KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

(For the candidates admitted from 2017 onwards)

DEPARTMENT OF COMPUTER SCIENCE,CA & IT

SUBJECT : **PROGRAMMING IN JAVA**

SEMESTER : **II** **L T P C**

SUBJECT CODE: **18CAU201** **CLASS** : **I BCA** **4 0 0 4**

COURSE OBJECTIVES:

- know the java path setting and programming techniques
- basic java programming and Applet programming
- understand the fundamental of Packages and access modifiers and interface in java
- understand the fundamental of Exception Handling and AWT component and AWT classes

COURSE LEARNING OUTCOMES:

Activities in this module map directly to the following three outcomes proposed in our grant proposal:

- Demonstrate in-depth understanding of the cyber security First Principles.
- Explore the use of various operating systems commands on different platforms.
- Have a better understanding of essential problem solving and programming concepts.

UNIT-I

Introduction to Java : Java Architecture and Features, Understanding the semantic and syntax differences between C++ and Java, Compiling and Executing a Java Program, Variables, Constants, Keywords Data Types, Operators (Arithmetic, Logical and Bitwise) and Expressions, Comments, Doing Basic Program Output, Decision Making Constructs (conditional statements and loops) and Nesting, Java Methods (Defining, Scope, Passing and Returning Arguments, Type Conversion and Type and Checking, Built-in Java Class Methods),

UNIT-II

Arrays, Strings and I/O Creating & Using Arrays (One Dimension and Multi-dimensional), Referencing Arrays Dynamically, Java Strings: The Java String class, Creating & Using String Objects, Manipulating Strings, String Immutability & Equality, Passing Strings To & From Methods, String Buffer Classes. Simple I/O using System. out and the Scanner class, Byte and Character streams, Reading/Writing from console and files. **Object-Oriented Programming Overview** Principles of Object-Oriented Programming, Defining & Using Classes, Controlling Access to Class Members, Class Constructors, Method Overloading, Class Variables & Methods, Objects as parameters, final classes, Object class, Garbage Collection.

UNIT-III

Inheritance, Interfaces, Packages, Enumerations, Auto boxing and Metadata

Inheritance: (Single Level and Multilevel, Method Overriding, Dynamic Method Dispatch, Abstract Classes), Interfaces and Packages, Extending interfaces and packages, Package and Class Visibility, Using Standard Java Packages (util, lang, io, net), Wrapper Classes, Auto boxing/Unboxing, Enumerations and Metadata.

UNIT-IV

Exception Handling, Threading, Networking and Database Connectivity Exception types, uncaught exceptions, throw, built-in exceptions, Creating your own exceptions; Multi-threading: The Thread class and Runnable interface, creating single and multiple threads, Thread prioritization, synchronization and communication, suspending/resuming threads. Using java.net package, Overview of TCP/IP and Datagram programming. Accessing and manipulating databases using JDBC.

UNIT-V

Java Applets: Introduction to Applets, Writing Java Applets, Working with Graphics,

Incorporating Images & Sounds. Event Handling Mechanisms, Listener Interfaces, Adapter and Inner Classes. The design and Implementation of GUIs using the AWT controls, Swing

components of Java Foundation Classes such as labels, buttons, text fields, layout managers, menus, events and listeners; Graphic objects for drawing figures such as lines, rectangles, ovals, using different fonts. Overview of servlets.

Suggested Readings:

1. Ken Arnold, James Gosling, David Homes, 2005,*The Java Programming Language*, 4th Edition.
2. James Gosling, Bill Joy, Guy L Steele Jr,Gilad Bracha, Alex Buckley, 2014, *The Java Language Specification*, Java SE 8th Edition (Java Series), Published by Addison Wesley.
3. Joshua Bloch, 2008, *Effective Java*, 2nd Edition,Publisher: Addison-Wesley.
4. Cay S. Horstmann, Gary Cornell, 2012, *Core Java 2 Volume 1* ,9th Edition,Printice Hall.
5. Cay S. Horstmann, Gary Cornell, 2013,*Core Java 2 Volume 2 - Advanced Features*, 9th Edition,Printice Hall.
6. Bruce Eckel, 2002,*Thinking in Java*, 3rd Edition, PHI.
7. E. Balaguruswamy, 2009,*Programming with Java*, 4th Edition, McGraw Hill.
8. Paul Deitel, Harvey Deitel, 2011, *Java: How to Program*, 10th Edition, Prentice Hall.
9. David J. Eck, 2009,*Introduction to Programming Using Java*, Published by CreateSpace Independent Publishing Platform.
10. John R. Hubbard, 2004,*Programming with JAVA*, Schaum's Series, 2nd Edition.

WEBSITES

1. java.sun.com/docs/books/tutorial/
2. www.en.wikipedia.org/wiki/Java
3. www.java.net/

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed University Established Under Section 3 of UGC Act 1956)

Coimbatore - 641021.

(For the candidates admitted from 2018 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

SUBJECT : **PROGRAMMING IN JAVA**
SEMESTER : **II** **L T P C**
SUBJECT CODE : **18CAU201** **CLASS** : **I BCA** **4 0 0 4**

LECTURE PLAN

STAFF NAME: M.THILLAINAYAKI

S.No	Lecture Duration (Hr)	Topics	Support Materials
		UNIT-I	
1.	1	• **Introduction to Java:** Java Architecture and Features	T2:13-15, W1,W2,W3
2.	1	• Understanding the semantic and syntax, Differences between C++ and Java, Compiling and Executing a Java Program	W4, T1:23-27, T2:16-17
3.	1	• Variables, Constants, Keyword, Data Types	R1:3-5, T2:22-24, T1:35-40, T2:19-22
4.	1	• Operators (Arithmetic, Logical and Bitwise) and Expressions, Comments	T2:31-37, R1:6
5.	1	• Doing Basic Program Output, Decision Making Constructs (conditional statements and loops) and Nesting	W5, W6, T2:41-47
6.	1	• Java Methods Defining, Scope, Passing and Returning Arguments, Type Conversion and Type and Checking	T2:54-57, T2:57-61, R1:50-57 , T2:24
7.	1	• Built-in Java Class Methods	W6
8.	1	• Recapitulation and Discussion of important questions	
Text Book	**T1→** Herbert Schildt, Java the Complete Reference, 8th Edition.		
	T2→ISRD Group, Introduction to object oriented programming through Java.		
Reference Book	**R1→**Ken Arnold, James Gosling, David Homes, 2005, *The Java Programming*		

Websites	*Language*, 4th Edition. **W1 →**https://www.javatpoint.com/features-of-java **W2→**http://www.careerbless.com/java/basics/JavaArchitecture.php **W3→**www./javainterviewpoint.com/java-virtual-machine-architecture-in-java **W4→** https://www.javatpoint.com/cpp-vs-java **W5→**https://www.w3schools.in/java-tutorial/ **W6→**https://beginnersbook.com/java-tutorial-for-beginners-with-examples/ **W7→** https://www.quora.com/Where-can-I-get-the-list-of-all-java-predefined-classes-and-methods		
Total No of Hours Planned For Unit – I : 8 Hours			
UNIT-II			
1.	1	• **Arrays, Strings and I/O** Creating & Using Arrays, Referencing Arrays Dynamically	T2:25-27, W4
2.	1	• Java Strings The Java String class, Creating & Using String Objects, Manipulating Strings, String Immutability & Equality	T2:177-188
3.	1	• Passing Strings To & From Methods, String Buffer Classes.	T2:188-189, R1:222-226
4.	1	• Simple I/O using System. out and the Scanner class	T1:563-572
5.	1	• Byte and Character streams, Reading/Writing from console and files.	R1:355-367
6.	1	• **Object-Oriented Programming Overview:** Class Members, Class Constructors, Method Overloading	T2:70-72, R1:35-37, W4
7.	1	• Class Variables & Methods, Objects as parameters, final classes, Object class, Garbage Collection.	T2:56-65, R1:313-318, W4
8.	1	• Recapitulation and Discussion of important questions	
Text Book	**T1→** Herbert Schildt, Java the Complete Reference, 8th Edition.		

Reference Book	**T2→**ISRD Group, Introduction to object oriented programming through Java. **R1→**Ken Arnold, James Gosling, David Homes, 2005, *The Java Programming Language*, 4th Edition.		
Websites	**W4→** https://www.javatpoint.com/cpp-vs-java		
Total No of Hours Planned For Unit – II : 8 Hours			
UNIT-III			
1.	1	• **Inheritance, Interfaces, Packages, Enumerations, Auto boxing and Metadata**	T1:161,187,196,270,259, 275
2.	1	• Inheritance: Single Level and Multilevel , Method Overriding	T1:161-173, T2:100
3.	1	• Dynamic Method Dispatch, Abstract Classes	T1:178-180, T2:107
4.	1	• Interfaces and Packages	R1:105-110, 329
5.	1	• Extending interfaces and packages	R1:110-112, 329
6.	1	• Package and Class Visibility, Using Standard Java Packages , Wrapper Classes	W4, T2:204
7.	1	• Auto boxing/Unboxing, Enumerations and Metadata.	T1:267 - 286
8.	1	• Recapitulation and Discussion of important questions	
Text Book	**T1→** Herbert Schildt, Java the Complete Reference, 8th Edition.		
Reference Book	**T2→**ISRD Group, Introduction to object oriented programming through Java. **R1→**Ken Arnold, James Gosling, David Homes, 2005, *The Java Programming Language*, 4th Edition.		
Websites	**W4→** https://www.javatpoint.com/cpp-vs-java		
Total No of Hours Planned For Unit – III 8 Hours			
UNIT-IV			
1.	1	• Exception types, uncaught exceptions, Throw, built-in exceptions, Creating your own exceptions	T1:209, T2:122-123, T1:216-221

2.	1	• Multi-threading: The Thread class and Runnable interface, Creating single and multiple threads	T1:230, T2:140-141, T1:232-237
3.	1	• Thread prioritization	T1:229-240
4.	1	• Synchronization and communication suspending/resuming threads	T2:155-158
5.	1	• Using java.net package	R1:536-537
6.	1	• Overview of TCP/IP and Datagram programming.	W7
7.	1	• Accessing and manipulating databases using JDBC.	T2:441-449
8.	1	• Recapitulation and Discussion of important questions	
Text Book	**T1→** Herbert Schildt, Java the Complete Reference, 8th Edition. **T2→**ISRD Group, Introduction to object oriented programming through Java.		
Reference Book	**R1→**Ken Arnold, James Gosling, David Homes, 2005, *The Java Programming Language*, 4th Edition.		
Websites	**W7→** https://www.cs.auckland.ac.nz/~jmor159/364/ppt/AJavaNetworking.ppt		
Total No of Hours Planned For Unit – IV 8 Hours			
UNIT-V			
1.	1	• Java Applets: Introduction to Applets, Writing Java Applets, Working with Graphics, Incorporating Images & Sounds.	T2:292-293, T1:688-689, T1:749-754, T2:300-301, T1:830-831,R1:548
2.	1	• Event Handling Mechanisms, Listener Interfaces, Adapter and Inner Classes	T1:707-719, T1:720-723, T1:729-731
3.	1	• Swing components of Java Foundation Classes	T1:965,969-975,967
4.	1	• Graphic objects for drawing, Overview of servlets	T1:749-754, T1:993-998,T2:477-484
5.	1	• Recapitulation and Discussion of	

		important questions	
6.	1	**Discussion of previous ESE Question papers**	
7.	1	**Discussion of previous ESE Question papers**	
8.	1	**Discussion of previous ESE Question papers**	
Text Book **Reference Book**	**T1→** Herbert Schildt, Java the Complete Reference, 8th Edition. **T2→**ISRD Group, Introduction to object oriented programming through Java. **R1→**Ken Arnold, James Gosling, David Homes, 2005, *The Java Programming Language*, 4th Edition.		
Total No of Hours Planned For Unit – V 8 Hours			
Total No. of Hours Planned: 40			

Text Book:

T1→ Herbert Schildt, Java the Complete Reference, 8th Edition.

T2→ ISRD Group, Introduction to object oriented programming through Java.

Reference Book:

R1→Ken Arnold, James Gosling, David Homes, 2005, *The Java Programming Language*, 4th Edition.

Suggested Readings:

1. James Gosling, Bill Joy, Guy L Steele Jr, Gilad Bracha, Alex Buckley, 2014, *The Java Language Specification*, Java SE 8th Edition (Java Series), Published by Addison Wesley.
2. Joshua Bloch, 2008, *Effective Java*, 2nd Edition, Publisher: Addison-Wesley.
3. Cay S. Horstmann, Gary Cornell, 2012, *Core Java 2 Volume 1*, 9th Edition, Prentice Hall.
4. Cay S. Horstmann, Gary Cornell, 2013, *Core Java 2 Volume 2 - Advanced Features*, 9th Edition, Prentice Hall.
5. Bruce Eckel, 2002, *Thinking in Java*, 3rd Edition, PHI.
6. E. Balaguruswamy, 2009, *Programming with Java*, 4th Edition, McGraw Hill.

7. Paul Deitel, Harvey Deitel, 2011, *Java: How to Program*, 10th Edition, Prentice Hall.

8. David J. Eck, 2009, *Introduction to Programming Using Java*, Published by CreateSpace Independent Publishing Platform.

9. John R. Hubbard, 2004, *Programming with JAVA*, Schaum's Series, 2nd Edition.

WEBSITES

W1 →https://www.javatpoint.com/features-of-java

W2→http://www.careerbless.com/java/basics/JavaArchitecture.php

W3→www./javainterviewpoint.com/java-virtual-machine-architecture-in-java

W4→ https://www.javatpoint.com/cpp-vs-java

W5→https://beginnersbook.com/java-tutorial-for-beginners-with-examples/

W6→ https://www.quora.com/Where-can-I-get-the-list-of-all-java-predefined-classes-and-methods

W7→ https://www.cs.auckland.ac.nz/~jmor159/364/ppt/AJavaNetworking.ppt

UNIT – I

SYLLABUS

Introduction to Java : Java Architecture and Features, Understanding the semantic and syntax differences between C++ and Java, Compiling and Executing a Java Program, Variables, Constants, Keywords Data Types, Operators (Arithmetic, Logical and Bitwise) and Expressions, Comments, Doing Basic Program Output, Decision Making Constructs (conditional statements and loops) and Nesting, Java Methods (Defining, Scope, Passing and Returning Arguments, Type Conversion and Type and Checking, Built-in Java Class Methods)

Introduction to Java

Java Architecture:

1. Compilation and interpretation in Java

Java combines both the approaches of compilation and interpretation. First, java compiler compiles the source code into byte code. At the run time, Java Virtual Machine (JVM) interprets this byte code and generates machine code which will be directly executed by the machine in which java program runs. So java is both compiled and interpreted language.

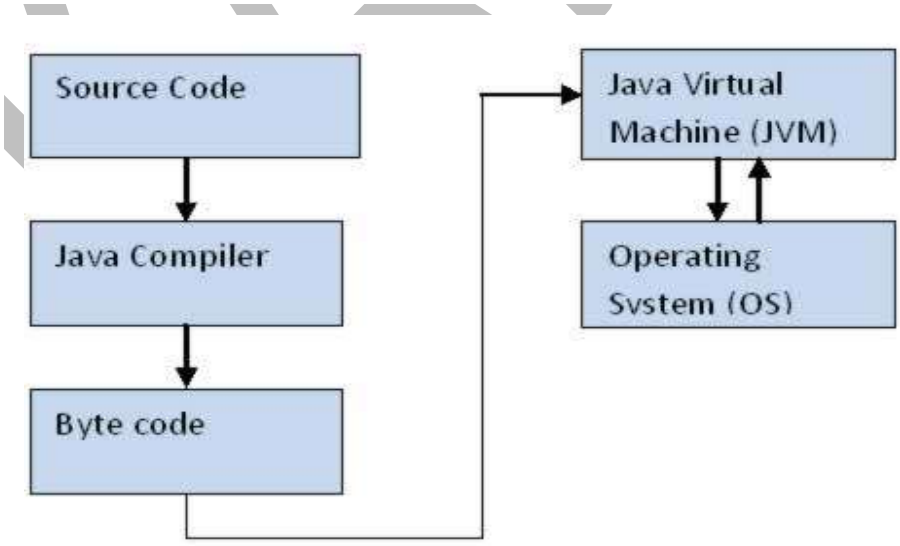


Figure 1.1: Java Architecture

2. Java Virtual Machine (JVM)

JVM is a component which provides an environment for running Java programs. JVM interprets the byte code into machine code which will be executed the machine in which the Java program runs. Java was developed with the concept of **WORA (*Write Once Run Anywhere)*** which runs on a **VM**. **The compiler** will be compiling the **java** file into a java **.class** file. The **.class** file is input to JVM which Loads and executes the **class file**. Below goes the Architecture of JVM.

Java Environment

The Java Virtual Machine

At the heart of Java's network-orientation is the Java virtual machine, which supports all three prongs of Java's network-oriented architecture: platform independence, security, and network-mobility.

The Java virtual machine is an abstract computer. Its specification defines certain features every Java virtual machine must have, but leaves many choices to the designers of each implementation. For example, although all Java virtual machines must be able to execute Java byte codes, they may use any technique to execute them. Also, the specification is flexible enough to allow a Java virtual machine to be implemented either completely in software or to varying degrees in hardware. The flexible nature of the Java virtual machine's specification enables it to be implemented on a wide variety of computers and devices.

A Java virtual machine's main job is to load class files and execute the byte codes they contain. As you can see in Figure 1-3, the Java virtual machine contains a *class loader*, which loads class files from both the program and the Java API. Only those class files from the Java API that are actually needed by a running program are loaded into the virtual machine. The byte codes are executed in *an* execution engine.

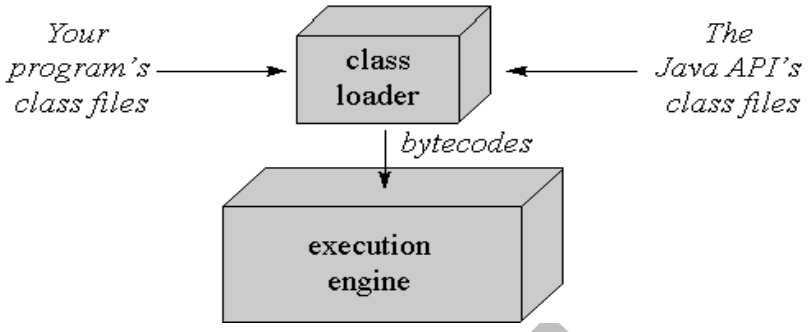


Figure 1-2. A basic block diagram of the Java virtual machine.

The execution engine is one part of the virtual machine that can vary in different implementations. On a Java virtual machine implemented in software, the simplest kind of execution engine just interprets the byte codes one at a time. Another kind of execution engine, one that is faster but requires more memory, is a *just-in-time compiler*. In this scheme, the byte codes of a method are compiled to native machine code the first time the method is invoked.

Java architecture

Java's architecture arises out of four distinct but interrelated technologies:

- the Java programming language
- the Java class file format
- the Java Application Programming Interface
- the Java virtual machine

When you write and run a Java program, you are tapping the power of these four technologies. You express the program in source files written in the Java programming language, compile the source to Java class files, and run the class files on a Java virtual machine. When you write your program, you access system resources (such as I/O, for example) by calling methods in the classes that implement the Java Application Programming Interface, or Java API. As your program runs, it fulfills your program's Java API calls by invoking methods in class files that implement the Java API. You can see the relationship between these four parts in Figure 1-1.

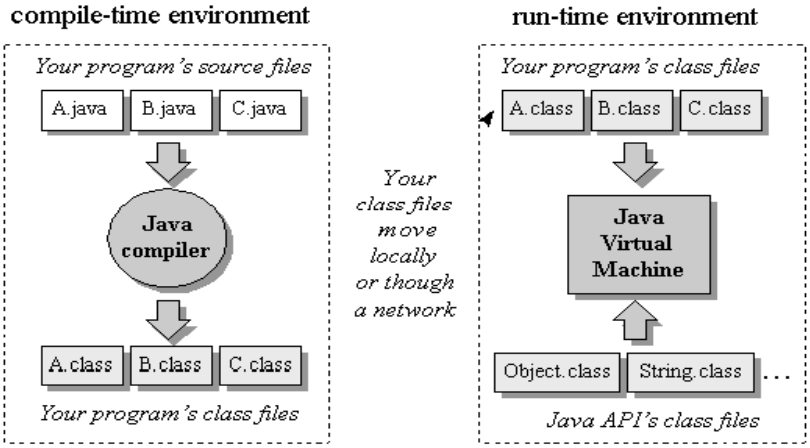


Figure 1-3. The Java programming environment.

Together, the Java virtual machine and Java API form a "platform" for which all Java programs are compiled. In addition to being called the *Java runtime system*, the combination of the Java virtual machine and Java API is called the *Java Platform* (or, starting with version 1.2, the *Java 2 Platform*). Java programs can run on many different kinds of computers because the Java Platform can itself be implemented in software. As you can see in Figure 1- 2, a Java program can run anywhere the Java Platform is present.

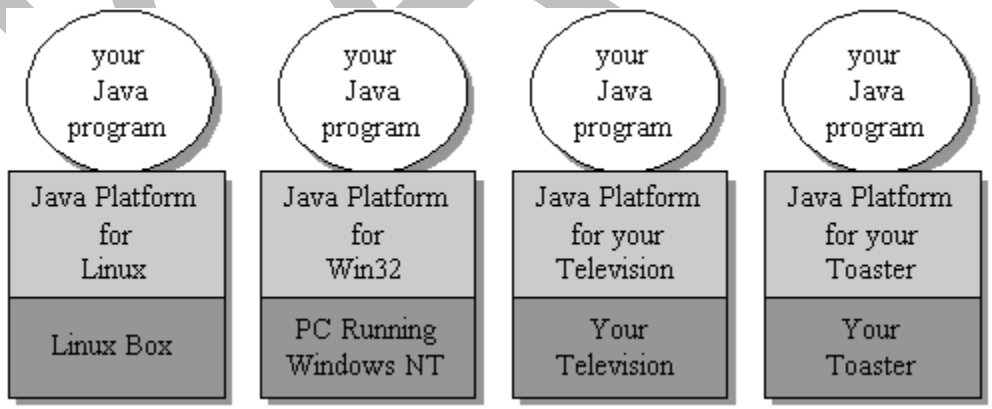


Figure 1-4. Java programs run on top of the Java Platform.

1.3.4 Java development kit

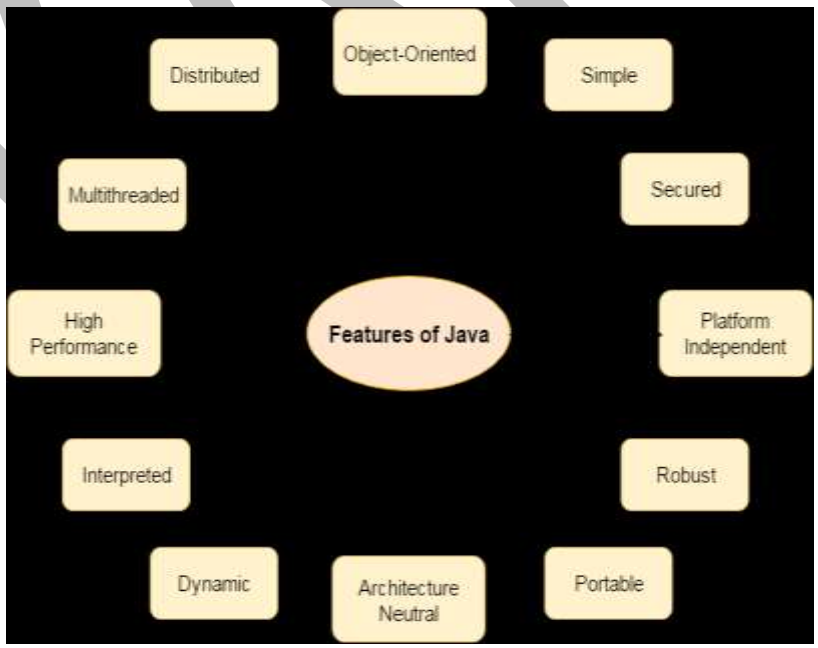
The **Java Development Kit** (**JDK**) is a Sun Microsystems product aimed at Java developers. Since the introduction of Java, it has been by far the most widely used Java SDK. On 17 November 2006, Sun announced that it would be released under the GNU General Public License (GPL), thus making it free software. This happened in large part on 8 May 2007[3]; Sun contributed the source code to the OpenJDK.

The JDK has as its primary components a collection of programming tools, including:

- java – the loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment. The old deployment launcher, jre, no longer comes with Sun JDK.
- javac – the compiler, which converts source code into Java byte code
- applet viewer – this tool can be used to run and debug Java applets without a web browser

Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.



1. Simple

2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

1. Simple

- According to Sun, Java language is simple because:
- Syntax is based on C++ (so easier for programmers to learn it after C++).
- Removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.
- No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

2. Object-oriented

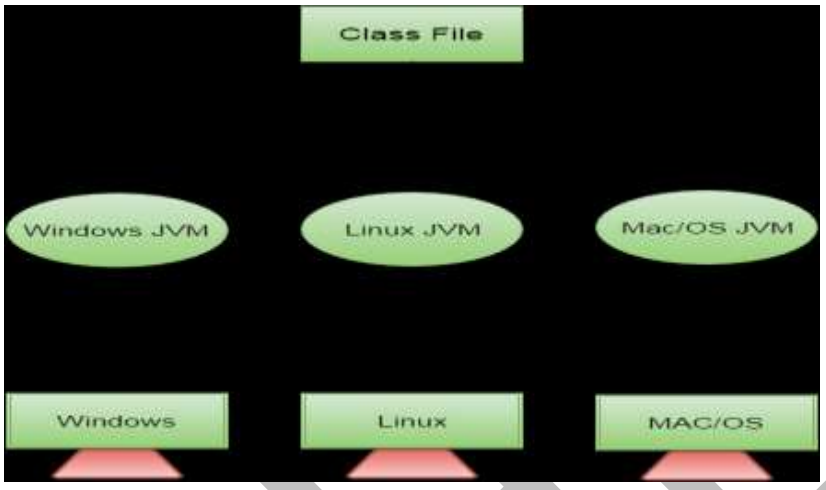
- Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Object-oriented programming (OOPs) is methodologies that simplify software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction

6. Encapsulation

3. Platform Independent



A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

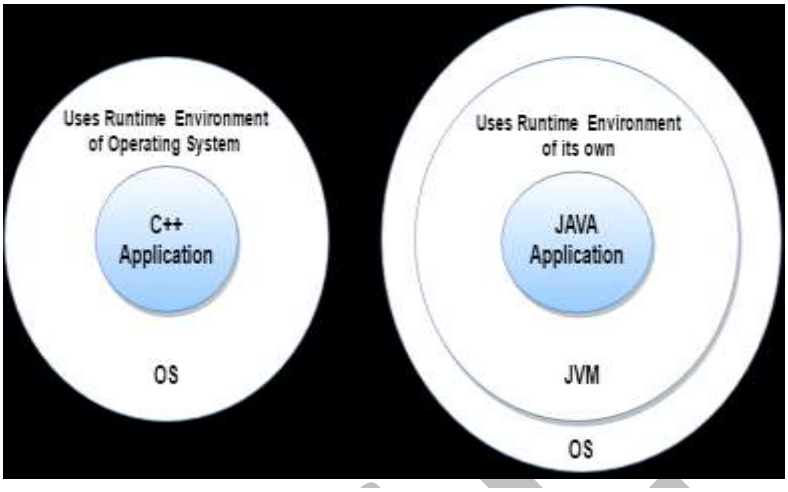
1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac / OS etc. Java code is compiled by the compiler and converted into byte code. This byte code is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere (WORA).

4. Secured

Java is secured because:

- **No explicit pointer**
- **Java Programs run inside virtual machine sandbox**



- **Classloader:** adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** determines what resources a class can access such as reading and writing to the local disk.

These securities are provided by java language. Some security can also be provided by application developer through SSL, JAAS, and Cryptography etc.

5. Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

6. Architecture-neutral

There are no implementation dependent features e.g. size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

7. Portable

We may carry the java bytecode to any platform.

8. High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

9. Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

10. Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

Introduction to OOP

Object Oriented Programming or OOP is the technique to create programs based on the real world. Unlike procedural programming, here in the OOP programming model programs are organized around objects and data rather than actions and logic. Objects represent some concepts or things and like any other objects in the real Objects in programming language have certain behavior, properties, type, and identity. In OOP based language the principal aim is to find out the objects to manipulate and their relation between each other.

Class - It is the central point of OOP and that contains data and codes with behavior. In Java everything happens within class and it describes a set of objects with common behavior. The class definition describes all the properties, behavior, and identity of objects present within that class. As far as types of classes are concerned, there are predefined classes in languages like C++ and Pascal. But in Java one can define his/her own types with data and code.

Object - Objects are the basic unit of object orientation with behavior, identity. As we mentioned above, these are part of a class but are not the same. An object is expressed by the

variable and methods within the objects. Again these variables and methods are distinguished from each other as instant variables, instant methods and class variable and class methods.

Methods - We know that a class can define both attributes and behaviors. Again attributes are defined by variables and behaviors are represented by methods. In other words, methods define the abilities of an object.

Inheritance - This is the mechanism of organizing and structuring software program. Though objects are distinguished from each other by some additional features but there are objects that share certain things common. In object oriented programming classes can inherit some common behavior and state from others. Inheritance in OOP allows to define a general class and later to organize some other classes simply adding some details with the old class definition. This saves work as the special class inherits all the properties of the old general class and as a programmer you only require the new features. This helps in a better data analysis, accurate coding and reduces development time.

Abstraction - The process of abstraction in Java is used to hide certain details and only show the essential features of the object. In other words, it deals with the outside view of an object (interface).

Encapsulation - This is an important programming concept that assists in separating an object's state from its behavior. This helps in hiding an object's data describing its state from any further modification by external component. In Java there are four different terms used for hiding data constructs and these are public, private, protected and package. As we know an object can associated with data with predefined classes and in any application an object can know about the data it needs to know about. So any unnecessary data are not required by an object can be hidden by this process. It can also be termed as information hiding that prohibits outsiders in seeing the inside of an object in which abstraction is implemented.

Polymorphism - It describes the ability of the object in belonging to different types with specific behavior of each type. So by using this, one object can be treated like another and in this way it

can create and define multiple level of interface. Here the programmers need not have to know the exact type of object in advance and this is being implemented at runtime.

C++ vs. Java

There are many differences and similarities between C++ programming language and Java. A list of top differences between C++ and Java are given below:

S.NO	Comparison Index	C++	Java
1	Platform-independent	C++ is platform-dependent.	Java is platform-independent.
2	Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
3	Goto	C++ supports goto statement.	Java doesn't support goto statement.
4	Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
5	Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.

6	Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java.
7	Compiler and Interpreter	C++ uses compiler only.	Java uses compiler and interpreter both.
8	Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
9	Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
10	Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
11	Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
12	Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by

		function.	default.
13	unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
14	Inheritance Tree	C++ creates a new inheritance tree always.	Java uses single inheritance tree always because all classes are the child of Object class in java. Object class is the root of inheritance tree in java.

Compiling and Executing a Java Program

1. Write source code

The following Java program is developed under Microsoft Windows. The process on other operating system should be similar but will not be covered here. Select a directory which should contain your code. I will use the directory c:\temp\java which will be called "javadir".

Open a text editor which supports plain text, e.g. notepad under Windows and write the following source code. You can start notepad via Start->Run-> notepad and pressing enter.

```
// The smallest Java program possible

public class HelloWorld {

        public static void main(String[] args) {

                System.out.println("Hello");}}
```

Save the source code in your directory "javadir" under the name "HelloWorld.java".The name of a Java source file must always equals the class name (within the source code) and end with .java. In our case the filename must be HelloWorld.java because the class is called HelloWorld.

2. Compile the code

Switch to the command line, e.g. under Windows Start-> Run -> cmd. Switch to the "javadir" directory with the command cd javadir, for example in my case cd c:\temp\java. Use the command dir to see that the source file is in the directory.

Type javac **Hello.java**.

Check the content of the directory with the command "dir". The directory contains now a file "HelloWorld.class". If you see this file you have successfully compiled your first Java source code into byte-code.

3. Run the code

Switch again to the command line, e.g. under Windows Start-> Run -> cmd. Switch to the directory jardir.

Type **java Hello**.

The system should write "Hello World" on the command line.

4. Using the classpath

You can use the classpath to run the program from another place in your directory.

Switch to the command line, e.g. under Windows Start-> Run -> cmd. Switch to any directory you want.

Type java Hello.

If you are not in the directory in which the compiled class is stored then the system should result an error message **Exception in thread "main" java.lang.NoClassDefFoundError: test/TestClass**

Type **java -classpath "mydirectory" HelloWorld** . Replace "mydirectory" with the directory which contains the test directory. You should again see the "HelloWorld" output.

Variable declaration

A variable is a container that stores a meaningful value that can be used throughout a program. For example, in a program that calculates tax on items you can have a few variables - one variable that stores the regular price of an item and another variable that stores the total price of an item after the tax is calculated on it. Variables store this information in a computer's memory and the value of a variable can change all throughout a program**.**

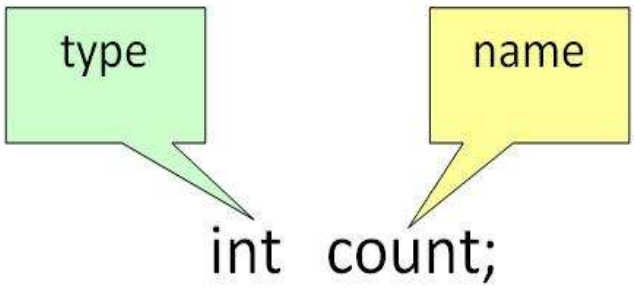
One variable in your program can store numeric data while another variable can store text data. Java has special keywords to signify what type of data each variable store. Use these keywords when declaring your variables to set the **data type** of the variable.

Java data types		
Keyword	**Type of data the variable will store**	**Size in memory**
boolean	true/false value	1 bit
byte	byte size integer	8 bits
char	a single character	16 bits

double	double precision floating point decimal number	64 bits
float	single precision floating point decimal number	32 bits
int	a whole number	32 bits
long	a whole number (used for long numbers)	64 bits
short	a whole number (used for short numbers)	16 bits

Variable Declaration:

To declare a variable, you must specify the data type & give the variable a unique name.



Examples of other Valid Declarations are,

int a,b,c;

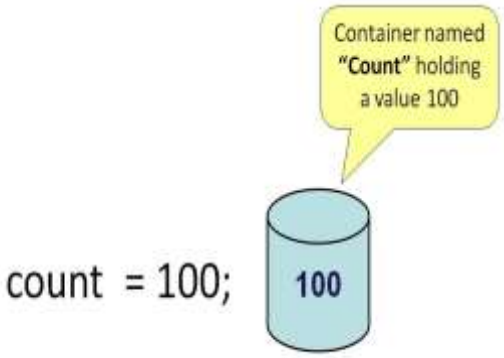
float pi;

double d;

char a;

2) Variable Initialization:

To initialize a variable you must assign it a valid value.



Example of other Valid Initializations are

pi =3.14f;

do =20.22d;

a='v';

You can combine variable declaration and initialization.

```
int count = 100;
```

Example:

int a=2,b=4,c=6;

float pi=3.14f;

double do=20.22d;

char a='v';

Data types

Java programming language is a language in which all the variables must be declared first and then to be used. That means to specify the name and the type of the variable. This specifies that Java is a strongly-typed programming language. Like

int pedal = 1;

This shows that there exists a field named 'pedal' that holds a data as a numerical value '1'. The values contained by the variables determines its data type and to perform the operations on it. **There are seven more primitive data types which are supported by Java language programming in addition to int.**

A primitive data type is a data type which is predefined in Java. Following are the eight primitive data types:

int

It is a 32-bit signed two's complement integer data type. It ranges from -2,147,483,648 to 2,147,483,647. This data type is used for integer values. However for wider range of values use

byte

The byte data type is an 8-bit signed two's complement integer. It ranges from -128 to127 (inclusive). We can save memory in large arrays using byte. We can also use byte instead of int to increase the limit of the code.

short

The short data type is a 16-bit signed two's complement integer. It ranges from -32,768 to 32,767. **short** is used to save memory in large arrays.

long

The long data type is a 64-bit signed two's complement integer. It ranges from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Use this data type with larger range of values.

float

The float data type is a single-precision 32-bit IEEE 754 floating point. It ranges from 1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative). Use a float (instead of double) to save memory in large arrays. We do not use this data type for the exact values such as currency. For that we have to use java.math.BigDecimal class.

double

This data type is a double-precision 64-bit IEEE 754 floating point. It ranges from 4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative). This data type is generally the default choice for decimal values.

boolean

The boolean data type is 1-bit and has only two values: **true** and **false**. We use this data type for conditional statements. true and false are not the same as True and False. They are defined constants of the language.

char

The char data type is a single 16-bit, unsigned Unicode character. It ranges from 0 to 65,535. They are not same as ints, shorts etc.

The following table shows the default values for the data types:

Keyword	Description	Size/Format
byte	Byte-length integer	8-bit two's complement
short	Short integer	16-bit two's

		complement
int	Integer	32-bit two's complement
long	Long integer	64-bit two's complement
float	Single-precision floating point	32-bit IEEE
double	Double-precision floating point	64-bit IEEE
char	A single character	16-bit Unicode character
boolean	A boolean value (true or false)	true or false

Java Tokens

In a Java program, all characters are grouped into symbols called **tokens**. Larger language features are built from the first five categories of tokens (the sixth kind of token is recognized, but is then discarded by the Java compiler from further processing). We must learn how to identify all six kinds of tokens that can appear in Java programs. In EBNF we write one simple rule that captures this structure:

token = identifier | keyword | separator | operator | literal | comment

The different types of Tokens are:

1. Identifiers: names the programmer chooses
2. Keywords: names already in the programming language

3. Separators (also known as punctuators): punctuation characters and paired-delimiters
4. Operators: symbols that operate on arguments and produce results
5. Literals (specified by their **type**)
    - Numeric: **int** and **double**
    - Logical: **boolean**
    - Textual: **char** and **String**
    - Reference: **null**
6. Comments
    - Line

Operators in Java

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-

1. Arithmetic Operators
2. Logical Operators
3. Bitwise Operators

Let's take a look at them in detail.

Arithmetic Operators: They are used to perform simple arithmetic operations on primitive data types.

1. **:** Multiplication
2. **/ :** Division
3. **% :** Modulo
4. **+ :** Addition
5. **:** Subtraction

```
// Java program to illustrate

// arithmetic operators

public class operators

{

   public static void main(String[] args)

   {

      int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
```

```
        String x = "Thank", y = "You";
        // + and - operator
        System.out.println("a + b = "+(a + b));
        System.out.println("a - b = "+(a - b));
        // + operator if used with strings
        // concatenates the given strings.
        System.out.println("x + y = "+x + y);
        // * and / operator
        System.out.println("a * b = "+(a * b));
        System.out.println("a / b = "+(a / b));
        // modulo operator gives remainder
        // on dividing first operand with second
        System.out.println("a % b = "+(a % b));
        // if denominator is 0 in division
        // then Arithmetic exception is thrown.
        // uncommenting below line would throw
        // an exception
        // System.out.println(a/c);
    }
}
```

Output:

a+b = 30

a-b = 10

x+y = ThankYou

a*b = 200

a/b = 2

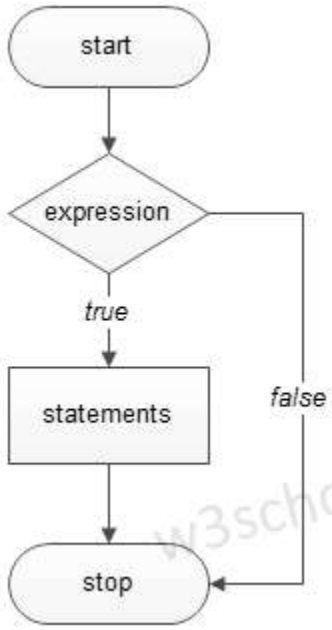
a%b = 0

Decision-making statements

A Java decision-making statement allows you to make decision, based upon the result of a condition.

All the programs in Java have set of statements, which are executed sequentially in the order in which they appear. This happens when jumping of statements or repetition of certain calculations is not necessary. However there may arise some situations where programmers have to change the order of execution of statements based on certain conditions which involves kind of decision-making statements. In this chapter you will learn about how the control flow statements works.

The flowchart of Decision making technique in Java can be expressed as:



Java has such decision making capabilities within its program by the use of following decision making statements:

Decision Making Statements in Java

- if Statement
  - if statement
  - if-else statement
  - else-if statement
- Conditional Operator
- switch statement

Java if Statements

If a statement in Java is used to control the program flow based on some condition, it's used to execute some statement code block if expression is evaluated to true, otherwise it will get skipped. This is an simplest way to modify the control flow of the program.

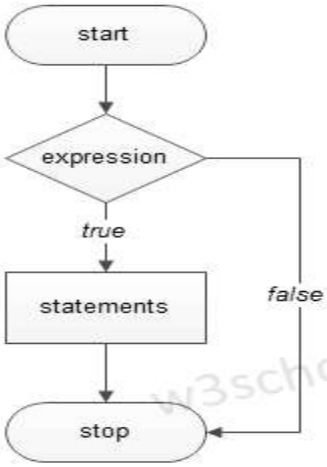
The basic format of if statement is:

Syntax:

```
if(test_expression)
{
   statement 1;
   statement 2;
   ...
}
```

'Statement n' can be a statement or a set of statements and if the test expression is evaluated to true, the statement block will get executed or it will get skipped.

Figure – Flowchart of if Statement:



w3schools.in

Example of a Java Program to Demonstrate If statements

Example:

```
public class Sample{
  public static void main(String args[]){
    int a=20, b=30;
    if(b>a)
      System.out.println("b is greater");
    }}
```

Program Output:

```
run:
b is greater
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java if-else Statement

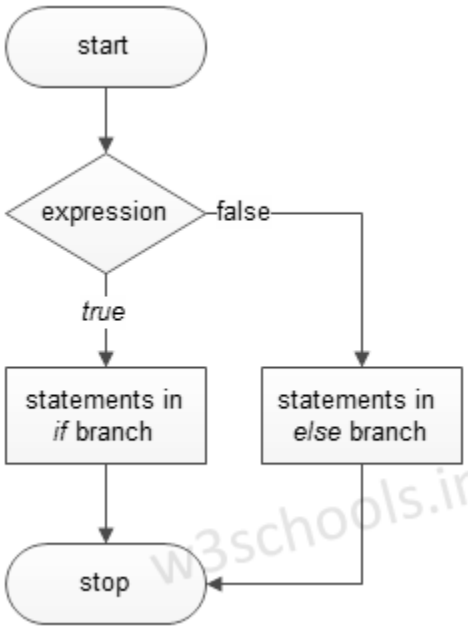
If else a statement in Java is also used to control the program flow based on some condition, only the difference is: it's used to execute some statement code block if expression is evaluated to true, otherwise executes else statement code block.

The basic format of if else statement is:

Syntax:

```
if(test_expression)
{
  //execute your code
}
else
{
  //execute your code
}
```

Figure – Flowchart of if else Statement:



Example of a Java Program to Demonstrate If else statements

```
public class Sample {
 public static void main(String args[]) {
  int a = 80, b = 30;
  if (b & gt; a) {
   System.out.println("b is greater");
  } else {
   System.out.println("a is greater");
  } }}
```

Program Output:

```
run:
a is greater
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java else-if Statements

else if statements in Java is like another if condition, it's used in program when if statement having multiple decisions.

The basic format of else if statement is:

Syntax:

```
if(test_expression)
{
  //execute your code
}
else if(test_expression n)
{
```

```
  //execute your code
}
else
{
  //execute your code
}
```

Example of a Java Program to Demonstrate else If statements

Example:

```
public class Sample {
   public static void main(String args[]) {
      int a = 30, b = 30;
      if (b > a) {
         System.out.println("b is greater");
      }
      else if(a > b){
         System.out.println("a is greater");
      }
      else {
         System.out.println("Both are equal");
      }}}
```

Program Output:

```
run:
Both are equal
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java switch Statements

Java switch statement is used when you have multiple possibilities for the if statement.

The basic format of switch statement is:

Syntax:

```
switch(variable)
{
case 1:
  //execute your code
break;

case n:
  //execute your code
break;

default:
  //execute your code
break;
}
```

After the end of each block it is necessary to insert a break statement because if the programmers do not use the break statement, all consecutive blocks of codes will get executed from each and every case onwards after matching the case block.

Example of a Java Program to Demonstrate Switch Statement

Example:

```
public class Sample {
 public static void main(String args[]) {
  int a = 5;
```

```
 switch (a) {
  case 1:
   System.out.println("You chose One");
   break;
  case 2:
   System.out.println("You chose Two");
   break;
  case 3:
   System.out.println("You chose Three");
   break;
  case 4:
   System.out.println("You chose Four");
   break;
  case 5:
   System.out.println("You chose Five");
   break;
  default:
   System.out.println("Invalid Choice. Enter a no between 1 and 5");
   break;
 } }}
```

Program Output:

KAHE

```
run:
You chose Five
BUILD SUCCESSFUL (total time: 0 seconds)
```

When none of the case is evaluated to true, then default case will be executed, and break statement is not required for default statement.

Java Loops

Sometimes it is necessary in the program to execute the statement several times, and Java loops execute a block of commands a specified number of times, until a condition is met. In this chapter you will learn about all the looping statements of Java along with their use.

What is Loop?

A computer is the most suitable machine to perform repetitive tasks and can tirelessly do a task tens of thousands of times. Every programming language has the feature to instruct to do such repetitive tasks with the help of certain form of statements. The process of repeatedly executing a collection of statement is called looping. The statements gets executed many number of times based on the condition. But if the condition is given in such a logic that the repetition continues any number of times with no fixed condition to stop looping those statements, then this type of looping is called infinite looping.

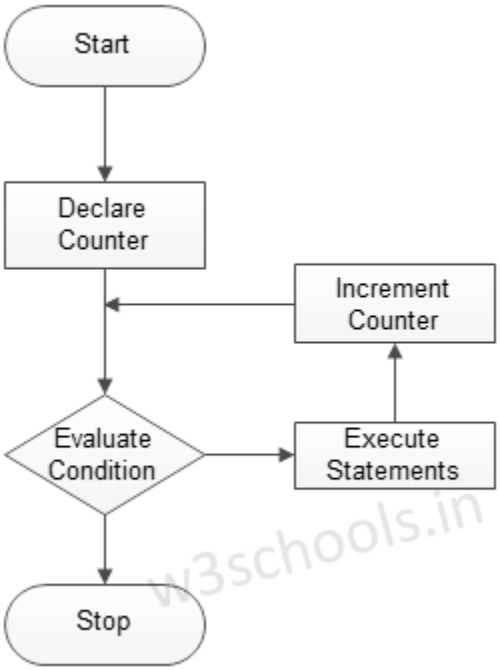
Java supports many looping features which enable programmers to develop concise Java programs with repetitive processes.

Java supports following types of loops:

- while loops
- do while loops
- for loops

All are slightly different and provides loops for different situations.

Figure – Flowchart of Looping:



Java Loop Control Statements

A Loop control statement is used to change normal sequence of execution of loop.

Statement	Syntax	Description
break statement	break;	Is used to terminate loop or switch statements.
continue statement	continue;	Is used to suspend the execution of current loop iteration and transfer control to the loop for the next iteration.
goto statement	goto labelName;labelName: statement;	It's transfer current program execution sequence to some other part of the program.

Java while loops

Java while loops statement allows to repeatedly running the same block of code, until a condition is met.

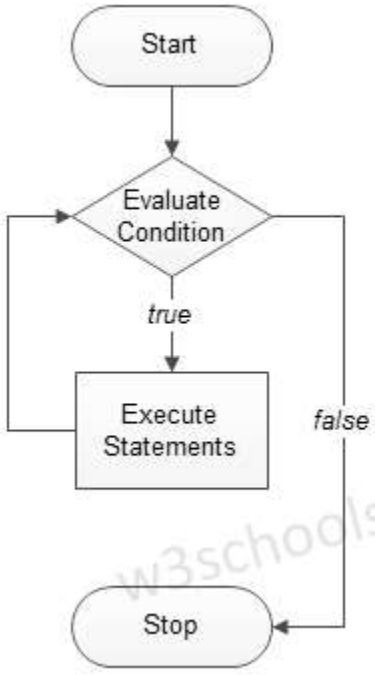
While loop is most basic loop in Java. It has one control condition, and executes as long the condition is true. The condition of the loop is tested before the body of the loop is executed, hence it is called an entry-controlled loop.

The basic format of while loop statement is:

Syntax:

```
While (condition)
{
  statement(s);
  incrementation;
}
```

Figure – Flowchart of while loop:



Example of a Java Program to Demonstrate while loop

Example:

```
public class Sample {


   public static void main(String args[]) {
      /* local variable Initialization */
      int n = 1, times = 5;
      /* while loops execution */
      while (n <= times) {
         System.out.println("Java while loops:" + n);
         n++;
      }   }}
```

Program Output:

```
run:
Java while loops:1
Java while loops:2
Java while loops:3
Java while loops:4
Java while loops:5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java do while loops

Java do while a loop is very similar to the while loops, but it always executes the code block at least once and further more as long as the condition remains true. This is exit-controlled loop.

The basic format of do while loop statement is:

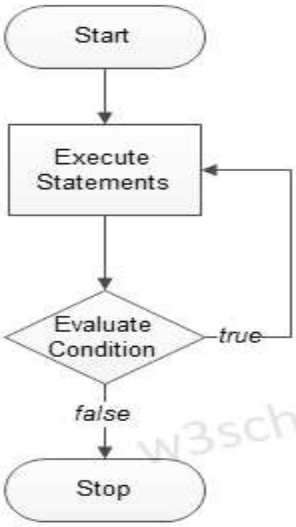
Syntax:

do

{ statement(s);

}while(condition);

Figure – Flowchart of do while loop:



Example of a Java Program to Demonstrate do while loop

```
public class Sample {
   public static void main(String args[]) {
      /* local variable Initialization */
      int n = 1, times = 0;
      /* do-while loops execution */
      do {
         System.out.println("Java do while loops:" + n);
         n++;
      } while (n <= times);    }}
```

Program Output:

```
run:
Java do while loops:1
BUILD SUCCESSFUL (total time: 1 second)
```

Java for loops

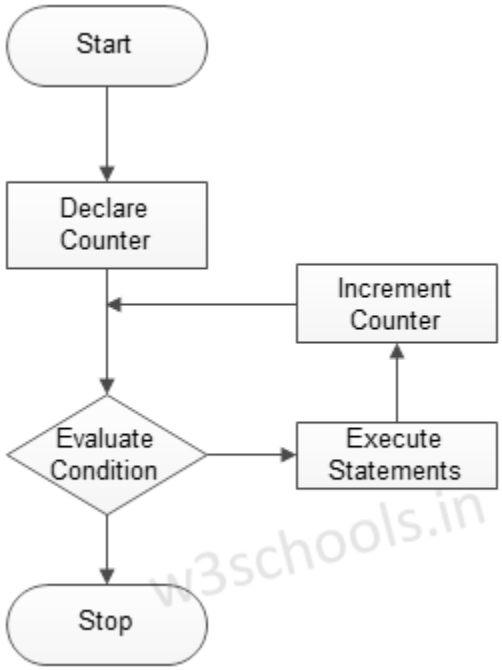
Java for loops is very similar to Java while loops in that it continues to process a block of code until a statement becomes false, and everything is defined in a single line.

The basic format of for loop statement is:

Syntax:

```
for ( init; condition; increment )
{
  statement(s);
}
```

Figure – Flowchart of for loop:



Example of a Java Program to Demonstrate for loop

Example:

```
public class Sample {
   public static void main(String args[]) {
      /* local variable Initialization */
      int n = 1, times = 5;
      /* for loops execution */
      for (n = 1; n <= times; n = n + 1) {
         System.out.println("Java for loops:" + n);      }   }}
```

Program Output:

```
run:
Java for loops:1
Java for loops:2
Java for loops:3
Java for loops:4
Java for loops:5
BUILD SUCCESSFUL (total time: 0 seconds)
```

What is a Method in Java?

In Java programming language, a method is a section of the program that contains a set of instructions or code. In a Java program, similar to a cake recipe, a method has a set of instructions. When the method is called, the set of instructions within the method is executed.

```
public int addNumbers (int x, int y)
{
int z = 0;
z = x+y;
System.out.println z;
return z;
}
```

Here, the name of the **method** is addNumbers. When the method addNumbers is called, the code within the method is executed, and the variable z is printed.

Parameters of a Method

When following the method to make cake, the ingredients like sugar and butter are combined and processed to make the final product. Similarly, Java methods have **parameters** (like ingredients) or data that are inputted or passed into the method. The method uses these parameter values to do the necessary data manipulation and processing. The method then usually returns the final value after all the necessary data processing is successfully performed.

Example:

```
public int subtractNumbers (int m, int n)
{
int p = 0;
p = m-n;
System.out.println p;
return p;
}
```

In this example, m and n are parameters. The Java method subtractNumbers finds the difference between m and n and saves the result in a new variable p. The values of the parameters m and n are used to generate the new variable p that is printed out on the computer screen.

The parameters of a method are declared within parentheses following the method name. If there is more than one parameter, they are separated by commas. Both the data type and the variable name (int m, int n) are specified for the parameters.

Returning a value from a Java method

When the method to make cake is completed using the necessary ingredients, the final result is a new product that is the cake. By using the parameters that are passed into the method, the method generates a new product or result. The result returned by the method is also available for use by the java program to which this method belongs.

Example:

```
public int subtractNumbers (int m, int n)
{
int p = 0;
p = m-n;
System.out.println p;
return p;
}
```

In this example, the variable p is returned by the method. The return statement is a java keyword return followed by the variable name.

```
return p;
```

When the method is declared, the return type of the variable is listed just before the name of the method. Here, the name of the method is subtractNumbers, and the data type of the variable being returned, p, is int, so the method declaration states:

```
public int subtractNumbers (int m, int n)
```

Where **int** is the return type of the method subtractNumbers.

Java Variable Type Conversion & Type Casting

A variable of one type can receive the value of another type. Here there are 2 cases - **case 1)** Variable of smaller capacity is be assigned to another variable of bigger capacity.

```
double d ;
int i = 10;
d = i;
```

This process is Automatic, and non-explicit is known as ***Conversion***
case 2) Variable of larger capacity is be assigned to another variable of smaller capacity

```
double d  = 10;
int i;
i = (int) d
```

Type Cast Operator

In such cases you have to explicitly specify the **type cast operator. This process is known as *Type Casting.***

In case, you do not specify a type cast operator, the compiler gives an error. Since this rule is enforced by the compiler, it makes the programmer aware that the conversion he is about to do may cause some loss in data and prevents **accidental losses.**

Example: To Understand Type Casting

```
class Demo{

public static void main(String args[]){

byte x;

int a=270;

double b =128.128;

System.out.println("int converted to byte");

x=(byte) a;

System.out.println("a and x "+ a +" "+x);
```

```
System.out.println("double converted to int");

a=(int) b;

System.out.println("b and a "+ b +" "+a);

System.out.println("\n double converted to byte");

x= b;

System.out.println("b and x "+b +" "+x);

} }
```

Built-in Java Class Methods

i) String Methods

ii) Number Methods

iii) Character methods

iv) Array methods Etc...

i) String Methods

1) compareTo() Method

The **java string compareTo()** method compares the given string with current string lexicographically. It returns positive number, negative number or 0.

It compares strings on the basis of Unicode value of each character in the strings.

If first string is lexicographically greater than second string, it returns positive number (difference of character value). If first string is less than second string lexicographically, it returns negative number and if first string is lexicographically equal to second string, it returns 0.

1. **if** s1 > s2, it returns positive number
2. **if** s1 < s2, it returns negative number
3. **if** s1 == s2, it returns 0

Signature

1. **public int** compareTo(String anotherString)

Parameters

anotherString: represents string that is to be compared with current string

Returns an integer value

Java String compareTo() method example

```
public class CompareToExample{
public static void main(String args[]){
String s1="hello";
String s2="hello";
String s3="meklo";
String s4="hemlo";
String s5="flag";
System.out.println(s1.compareTo(s2));//0 because both are equal
System.out.println(s1.compareTo(s3));//-5 because "h" is 5 times lower than "m"
System.out.println(s1.compareTo(s4));//-1 because "l" is 1 times lower than "m"
System.out.println(s1.compareTo(s5));//2 because "h" is 2 times greater than "f"
}}
```

Output:

```
0
-5
-1
2
```

2) equals() Method

The **java string equals()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

The String equals() method overrides the equals() method of Object class.

Signature

1. **public boolean** equals(Object anotherObject)

Parameter

anotherObject : another object i.e. compared with this string.

Returns

true if characters of both strings are equal otherwise **false**.

Overrides

equals() method of java Object class.

Java String equals() method example

```
public class EqualsExample{
public static void main(String args[]){
String s1="javatpoint";
String s2="javatpoint";
String s3="JAVATPOINT";
String s4="python";
System.out.println(s1.equals(s2));//true because content and case is same
System.out.println(s1.equals(s3));//false because case is not same
System.out.println(s1.equals(s4));//false because content is not same
}}
```

Output:

```
true
false
false
```

3) Concat() Method

The **java string concat()** method *combines specified string at the end of this string*. It returns combined string. It is like appending another string.

Signature

The signature of string concat() method is given below:

```
public String concat(String anotherString)
```

Parameter

anotherString : another string i.e. to be combined at the end of this string.

Returns combined string

Java String concat() method example

```
public class ConcatExample{
public static void main(String args[]){
String s1="java string";
s1.concat("is immutable");
System.out.println(s1);
s1=s1.concat(" is immutable so assign it explicitly");
System.out.println(s1);
}}
```

Output:

```
java string
java string is immutable so assign it explicitly
```

4) charAt() Method

Returns a character value by index...

Example:

```
String str1 = "Selenium";

System.out.println(str1.charAt(1));//e

System.out.println(str1.charAt(7));//m
```

5) equalsIgnoreCase()

It compares two strings and ignores letters (Upper case or Lower case...) Example:

String str1 = "Selenium";

String str2 = "SELENIUM";

String str3 = "UFT";

System.out.println(str1.equalsIgnoreCase(str2));//true
System.out.println(str2.equalsIgnoreCase(str3));//False

6) toUpperCase() Method

It Converts values to Upper case...

Example:

String str1 = "Selenium";

String str2 = "SELENIUM";

String str3 = "SELEnium";

String str4 = "selenium123";

System.out.println(str1.toUpperCase());//SELENIUM
System.out.println(str2.toUpperCase());//SELENIUM
System.out.println(str3.toUpperCase());//SELENIUM
System.out.println(str4.toUpperCase());//SELENIUM123 }

7) toLowerCase() Method

It converts values to Lower Case...

Example: String str1 = "selenium";

String str2 = "SELENIUM";

String str3 = "SELEnium";

String str4 = "selenium123";

System.out.println(str1.toLowerCase());//selenium System.out.println(str2.toLowerCase());//slenium
System.out.println(str3.toLowerCase());//selenium System.out.println(str4.toLowerCase());//selenium123

8) trim() Method

Removes spaces from both sides of a String...

Example:

String str1 = " Selenium ";

System.out.println(str1);

System.out.println(str1.trim());

9) subString() Method

Returns part of the string based on index position/s

Example:

String str1 = "Welcome to Selenium Testing"; System.out.println(str1.substring(11));//Selenium Testing System.out.println(str1.substring(20));//Testing System.out.println(str1.substring(11, 19));//Selenium System.out.println(str1.substring(8, 10));//to

10) endsWith() Method

It checks if the string Ends with specified suffix or not? And supports 2-way comparison (True/False)

String str1 = "Welcome to Selenium Testing"; System.out.println(str1.endsWith("Selenium Testing")); //True System.out.println(str1.endsWith("Testing")); //True
System.out.println(str1.endsWith("Selenium")); //False

11) length property

Returns length of a String

String str1 = "Selenium Testing";

String str2 = "Testing";

System.out.println(str1.length());//16

System.out.println(str2.length());//7

ii) Number methods

1) compareTo() methods

// Integer Class wraps a value of primitive Data Type int is an Object

//An Object of Integer contains a single field whose type is int...

Assignment to Sirisha - Compare two numbers - Numbers with decimal places... Number1 = Number2 then 0

Number1 > Number2 then 1

Number1 < Number2 then -1

Example:

```
int x = 5;

Integer a = x;

System.out.println(a.compareTo(5));//0

System.out.println(a.compareTo(4));//1

System.out.println(a.compareTo(7));//-1
```

2) equals() Method

It compares two numbers and it supports 2-way comparison

```
int a =10;

Integer b = a;

System.out.println(b.equals(10));//True

System.out.println(b.equals(7));//False

System.out.println(b.equals(14));//False
```

3) abs() Method

Returns absolute value....

```
double a = 10.234;

double b = 10.789;

double c =-20.345;

System.out.println(Math.abs(a));//10.234 System.out.println(Math.abs(b));//10.789
System.out.println(Math.abs(c));//20.345
```

4) round() Method

It Rounds the value to nearest Integer...

```
double a = 10.234;
```

```
double b = 10.789;
double c =-20.345;
System.out.println(Math.round(a));//10
System.out.println(Math.round(b));//11
System.out.println(Math.round(c));//-20
```

5) min() Method

It returns minimum value between two numbers...

Example:

```
int a = 5;
int b = 7;
double c = 10.234;
double d = 10.794;
System.out.println(Math.min(a, b));//5
System.out.println(Math.min(c, d));//10.234
System.out.println(Math.min(10, 17));//10
System.out.println(Math.min(1.34, 2.3));//1.34
```

6) max() Method

It returns maximum vale between two numbers...

Example:

```
int a = 5;
int b = 7;
double c = 10.234;
double d = 10.794;
System.out.println(Math.max(a, b));//7
System.out.println(Math.max(c, d));//10.794
```

System.out.println(Math.max(10, 17));//17

System.out.println(Math.max(1.34, 2.3));//2.3

7) random() Method

It generates a random Number...

Example:

System.out.println(Math.random());

iii) Character Methods

1) isLetter() method

Checks if the value is Alpha byte or not? And it returns Boolean Result / Logical Result (True/False)

Example: char a = 'Z';

char b = '1';

System.out.println(Character.isLetter(a));//True System.out.println(Character.isLetter(b));//False
System.out.println(Character.isLetter('A'));//True System.out.println(Character.isLetter('7'));//False
System.out.println(Character.isLetter('*'));//False

2) isDigit() Method

Checks if the value is Number or not? and it reruns Boolean / Logical Result

char a = 'Z';

char b = '1';

System.out.println(Character.isDigit(a));//False System.out.println(Character.isDigit(b));//True
System.out.println(Character.isDigit('A'));//False System.out.println(Character.isDigit('1'));//True
System.out.println(Character.isDigit('&'));//False

3) isLowerCase() Method

4) isUpperCase()

iv) Array Methods

1) length()

2) toString() etc...

POSSIBLE QUESTIONS
2 MARKS

1. Give any 4 differences between C++ and Java.

2. Write a Java code for Basic Program Output.

3. How to Compile and Execute a Java Program.

4. What is the result of the following program?

```
public class test
{
public static void main (string args[ ])
{
int i = -1;
i = i>>1 ;
System.out. println(i) ;
}
}
```

5. What gives java it's "write once and run anywhere" nature?

6. What is random() method?

7. What is Java Development Kit (JDK) ?

8. What is Data types?

9. What is Tokens?

10. Mention the Operators in Java?

11. What is Loop?

12. Define Type Conversion with example.

6 MARKS

1. List and explain the features of java and say why it is important?

2. Explain type conversions in java with suitable example program

3. Discuss operators in java with example.

4. Explain the following with suitable program. I) Nested if II)Switch

5. Define Methods. Illustrate Call by value and call by reference

6. Discuss Loops in java with example.

7. Discuss Java Methods with example program.

8. Illustrate the working of control statements in JAVA with appropriate examples.

9. Explain in detail about the features and architecture of JAVA.

10. Explain the following with example. i) Variable declaration ii) Expressions

11. Mention the Features of Java.

12. Explain in detail about Built-in Java Class Methods with examples

KARPAGAM ACADEMY OF HIGHER EDUCATION

COIMBATORE - 21

DEPARTMENT OF COMPUTER SCIENCE,CA & IT

CLASS : I BCA

BATCH : 2018-2021

Part -A Online Examinations (1 mark questions)

SUBJECT: Programming in Java SUBJECT CODE: 18CAU201

UNIT I

S. no	Questions	opt1	opt2	opt3	opt4	Answer
1	_____ is the automatic memory management	Memory release	Garbage collection	Memory management	Garbage compaction	Garbage collection
2	Java _____converts java source code into byte code	interpreter	compiler	assembler	preprocessor	compiler
3	_____ mechanism is java help to locate errors	Multithreading	Platform independent	Exception Handling	Runtime Binding	Exception Handling
4	The compiled code of Java program is called _____	binary code	octal code	byte code	hexadecimal code	byte code
5	Java was developed by _____	IBM	Microsoft	Sun Microsystems	Oracle Corporation	Sun Microsystems
6	Java is a _____ language	structured programming	object oriented	procedural oriented	machine	object oriented
7	OOPS follows_____	bottom_up	top_down	middle	top	bottom_up
8	Objects take up _____in the	Space	Address	Memory	bytes	Space
9	_____is a collection of objects of	Objects	methods	classes	messages	classes
10	Attributes are sometimes called_____	data members	methods	messages	functions	data members
11	The functions operate on the datas are	Methods	data members	messages	classes	Methods
12	The process of making an operator to exhibit	function overloading	operator overloading	method overloading	message overloading	operator overloading
13	.Variables are declared in_____	only in main()	anywhere in the scope	before the main() only	only at the beginning	anywhere in the scope
14	._____refers to permit	Dynamic initialization	Dynamic binding	Data binding	Dynamic message	Dynamic initialization
15	Keyword _____ indicates that method do	Static	Final	void	null	void
16	_____ is used to define the objects	class	functions	methods	none	class

17	An _____ is a single instance of a class that	class member	object	instances	none	object
18	A _____ is a message to take some action on an	member	variable	method	class	method
19	Java does not have _____ statement	goto	if	do	do while	goto
20	_____ is used to separate package names	:	,	.	!	.
21	The _____ is the basic unit of storage in a Java	identifier	variable	class	object	variable
22	byte belongs to _____ type.	character	Boolean	floating	integer	integer
23	In Java an int is _____ bits	16	64	52	32	32
24	byte is a signed _____ type	16 bit	8 bit	32 bit	64 bit	16 bit
25	The _____ statement is often used in switch	break	end	do	none	break
26	The keywords private and public are known as	Static	Dynamic	Visibility	const	Visibility
27	The class members that have been declared as	Private	Public	Static	protected	Private
28	The class members that have been declared as	Private	Public	Static	protected	Public
29	The class variables are known as _____	Functions	members	objects	none of the above	objects
30	The _____ command from J2SDK	Java	Appletviewer	Javac	javad	Javac
31	File produced by the java compiler contains	ASCII	Class	Pnemonics	ByteCodes	ByteCodes
32	The file produced by java compiler ends with	Java	html	class	applet	class
33	Objects are instantiated from_____	Java	methods	groups	class	class
34	Which of the following lines is not a Java	/** comments */	// comments	– comments	/* comments */	– comments
35	Which of the following statements is correct?	system.out.println('Welcome	System.out.println("Welcome	System.println('Welcome to	System.out.print('Welcome	System.out.println("Welcome
36	A block is enclosed inside _____.	Parentheses	Braces	Brackets	Quotes	Braces
37	Wich of the following is a correct signature for the	static void main(String[]	public static void	public void main(String[]	public static void	public static void

38	Which of the following lines is not a Java	/** comments */	// comments .	– comments	/* comments */	– comments
39	_____ translates the Java sourcecode to	javac	java	javap	jdk	javac
40	In java the functions are called as _____	fields	method	variables	none	method
41	_____ an object is also called as instantiating	deleting	creating	destroy	none	creating
42	Keyword _____ indicates that method do	Static	Final	void	null	void
43	Java interpreter is	JVM	Javac	Compiler	JAR	JVM
44	The _____ method terminates the program.	System.terminate(0);	System.halt(0);	System.exit(0);	System.stop(0);	System.exit(0);
45	Java has no _____ function.	malloc	free	both a & b	none	both a & b
46	Java supports _____ inheritance	single	multiple	both a & b	none	single
47	Java does not have _____	sturct	header files	union	all the above	all the above
48	_____ is a access specifier	static	void main	public	none	public
49	Which is invalid?	int a;	float x,y,z;	INT abc;	double a;	INT abc;
50	Which of these data type requires the most amount	long	Int	Short	byte	long
51	The equal comparison operator in Java is	<>	!=	==	^=	==
52	To add number to sum, you write (Note: Java is	number += sum;	number = sum + number;	sum = Number +	sum += number;	sum += number;
53	A_____ variable is known only in the method	Local	Global	Static	Auto	Local
54	The _____is	While	do_while	for	switch	do_while
55	The _____is an entry_entrolled loop	While	do_while	for	switch	do_while
56	_____refers to the use of same thing for different	Overloading	Dynamic binding	message loading	none	Overloading
57	_____ ___is a decision making	For	jump	break	if	if
58	The bool type data occupies	Two	one	three	four	one

59	The label must start with_____	Character	__	Number	alphanumeric	Character
60	_____statem	Jump	goto	continue	break	break
61	_____statement	For	if	goto	while	goto
62	_____statement is used to transfer the	Break	jump	goto	continue	continue
63	_____statement is a multiway branch	For	switch	if	while	switch
64	Test is performed at the _____of the for	Top	middle	end	program terminates	Top
65	Condition is checked at the _____of the	Beginning	end	middle	program terminates	end
66	Every relational expression always	0 or 1	1 or 2	_1 or 0	none	0 or 1
67	Which of the following loop statement uses 2	do_while loop	for loop	if loop	while loop	do_while loop
68	. Which of these operators is used to allocate memory	malloc	calloc	new	new malloc	new

UNIT-II

SYLLABUS

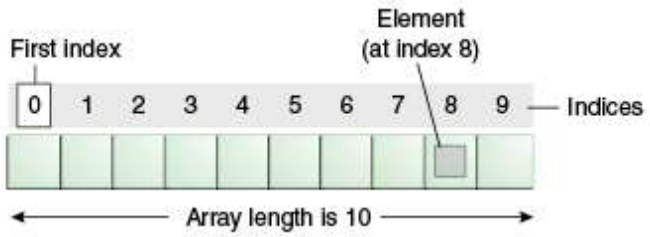
Arrays, Strings and I/O Creating & Using Arrays (One Dimensional and Multi-dimensional), Referencing Arrays Dynamically, Java Strings: The Java String class, Creating & Using String Objects, Manipulating Strings, String Immutability & Equality, Passing Strings To & From Methods, String Buffer Classes. Simple I/O using System. out and the Scanner class, Byte and Character streams, Reading/Writing from console and files. **Object-Oriented Programming Overview** Principles of Object-Oriented Programming, Defining & Using Classes, Controlling Access to Class Members, Class Constructors, Method Overloading, Class Variables & Methods, Objects as parameters, final classes, Object class, Garbage Collection.

Java Arrays

Normally, array is a collection of similar type of elements that have contiguous memory location.

Java array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based; first element of the array is stored at 0 index.

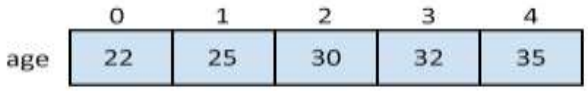


Example:

int age[5]={22,25,30,32,35};

Initializing each element separately in loop.

A Pictorial Representation of Array



Advantage of Java Array

Code Optimization: It makes the code optimized; we can retrieve or sort the data easily.

Random access: We can get any data located at any index position.

Disadvantage of Java Array

Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types of Array in java

There are two types of array.

1. Single Dimensional Array
2. Multidimensional Array

Single Dimensional Array in java

Syntax to Declare an Array in java

dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];

Instantiation of an Array in java

arrayRefVar=**new** datatype[size];

Example of single dimensional java array

Let's see the simple example of java array, where we are going to declare instantiate, initialize and traverse an array.

```
class Testarray{
public static void main(String args[]){
 int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
  //printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);  }}
```

Example:

Output:
10
20
70
40
50

Multidimensional array in java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in java

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)

dataType arrayRefVar[][]; (or)

dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

Example of Multidimensional java array

Let's see the simple example to declare instantiate, initialize and print the 2Dimensional array.

```
class Testarray3{
public static void main(String args[]){
  //declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
  //printing 2D array
for(int i=0;i<3;i++){
```

```
for(int j=0;j<3;j++){
System.out.print(arr[i][j]+" ");   }
System.out.println();  }  }}
```

Example:

Output:

1 2 3

2 4 5

4 4 5

Design a Class for Dynamic Arrays

In Java, the size of an array is fixed when it is created. Elements are not allowed to be inserted or removed. However, it is possible to implement a dynamic array by allocating a new array and copying the contents from the old array to the new one.

A *dynamic array* has variable size and allows elements to be added or removed. For this, we can allocate a fixed-size array and divide it into two parts:

- the first part stores the elements of the dynamic array and
- The second part is reserved, but not used.

Then we can add or remove elements at the end of the array by using the reserved space, until this space is completely consumed. After that, we create a bigger array and copy the contents of the old array to the new one.

- Logical size (size): the number of elements in the dynamic array
- Capacity: the physical size of the internal array (the maximum possible size without relocating storage)

We now design a class DynamicArray represents dynamic arrays of integers. It has two attributes:

- int[] data: an integer array, and
- int size: the logical size, the number of elements used

The capacity of this dynamic array is simply data. Length.

An important method we need is to add elements to the end of the dynamic array. This method should provide automatic extension if the capacity is not large enough to hold the added element.

In summary, we wish to design the class DynamicArray with the following members:

Attributes / Constructors / Methods:

- int[] data: the array storing the elements
- int size: the number of elements
- DynamicArray(): initialize this dynamic array with size 0
- DynamicArray(int capacity): initialize this dynamic array with the capacity
- int get(int index): get the element at the specified index
- int set(int index, int element): set the value of the element at the specified index
- boolean add(int element): add the element to the end of the array
- void ensureCapacity(int minCapacity): increase the capacity
- int size(): return the size of the dynamic array
- boolean isEmpty(): check whether the array is empty
- void clear(): clean up the elements

KAHE

Java String Class

String is a sequence of characters, for e.g. “Hello” is a string of 5 characters. In java, string is an immutable object which means it is constant and can cannot be changed once it has been created. In this tutorial we will learn about String class and String methods in detail along with many other Java String tutorials.

Creating a String

There are two ways to create a String in Java

1. String literal
2. Using new keyword

String literal

In java, Strings can be created like this: Assigning a String literal to a String instance:

```
String str1 = "Welcome";

String str2 = "Welcome";
```

Using New Keyword

As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this:

```
String str1 = new String("Welcome");
```

String str2 = new String("Welcome");

In this case compiler would create two different objects in memory having the same string.

A Simple Java String Example

```
public class Example{

  public static void main(String args[]){

      //creating a string by java string literal

      String str = "Beginnersbook";

      char arrch[]={'h','e','l','l','o'};

      //converting char array arrch[] to string str2

      String str2 = new String(arrch);

      //creating another java string str3 by using new keyword

      String str3 = new String("Java String Example");

      //Displaying all the three strings

      System.out.println(str);

      System.out.println(str2);

      System.out.println(str3);    }}
```

Output:

Beginnersbook

hello

Creating and using String Objects

String class

It is a predefined class in java.lang package can be used to handle the String. String class is immutable that means whose content cannot be changed at the time of execution of program.

String class object is immutable that means when we create an object of String class it never changes in the existing object.

Example:

```
classStringHandling{
publicstaticvoid main(String arg[]){
String s=newString("java");
s.concat("software");
System.out.println(s);}}
```

Output:

java

Explanation: Here we cannot change the object of String class so output is only java not java software.

Manipulating String

1. length()

length(): This method is used to get the number of character of any string.

Example

```
classStringHandling {
publicstaticvoid main(String arg[]) {
int l;
String s=newString("Java");
l=s.length();
System.out.println("Length: "+l);}}
```

Output

Length: 4

2. charAt(index)

charAt(): This method is used to get the character at a given index value.

Example

```
classStringHandling{

publicstaticvoid main(String arg[]){

char c;
String s=newString("Java");
c=s.charAt(2);
System.out.println("Character: "+c);}}
```

Output

Character: v

3. toUpperCase()

toUpperCase(): This method is use to convert lower case string into upper case.

Example

```
classStringHandling{

publicstaticvoid main(String arg[]){
String s="Java";
System.out.println("String: "+s.toUpperCase());}}
```

Output

String: JAVA

4. toLowerCase()

toLowerCase(): This method is used to convert lower case string into upper case.

Example **Output**

```
classStringHandling{
publicstaticvoid main(String arg[]){
String s="JAVA";
System.out.println("String: "+s.toLowerCase());}}
```

String: java

5. concat()

concat(): This method is used to combined two string.

Example **Output**

```
classStringHandling{
publicstaticvoid main(String arg[]){
String s1="Hitesh";
String s2="Raddy";
System.out.println("Combined String: "+s1.concat(s2));}}
```

Combined String: HiteshRaddy

6. equals()

equals(): This method is used to compare two strings, It return true if strings are same otherwise return false. It is case sensitive method.

Example **Output**

```
classStringHandling{
```

Compare String: false

```
publicstaticvoid main(String arg[]){
String s1="Hitesh";
String s2="Raddy";
String s3="Hitesh";
System.out.println("Compare String: "+s1.equals(s2));
System.out.println("Compare String: "+s1.equals(s3));}}
```

Compare String: true

7. equalsIgnoreCase()

equalsIgnoreCase(): This method is case insensitive method, It return true if the contents of both strings are same otherwise false.

Example

```
classStringHandling{
publicstaticvoid main(String arg[]){
String s1="Hitesh";
String s2="HITESH";
String s3="Raddy";
System.out.println("Compare String: "+s1.equalsIgnoreCase(s2));
System.out.println("Compare String: "+s1.equalsIgnoreCase(s3));}}
```

Output

Compare String: true

Compare String: false

8. compareTo()

compareTo(): This method is used to compare two strings by taking unicode values, It return 0 if the string are same otherwise return +ve or -ve integer values.

Example

```
classStringHandling{
```

Output

Strings are not same

```
publicstaticvoid main(String arg[]){
String s1="Hitesh";
String s2="Raddy";
int i;
i=s1.compareTo(s2);
if(i==0){
System.out.println("Strings are same");}
else{
System.out.println("Strings are not same");}}}
```

9.startsWith()

startsWith(): This method return true if string is start with given another string, otherwise it returns false.

Example **Output**

```
classStringHandling{
publicstaticvoid main(String arg[]){
String s="Java is programming language";
System.out.println(s.startsWith("Java"));}}
```

Output: true

10. endsWith()

endsWith(): This method return true if string is end with given another string, otherwise it returns false.

Example **Output**

```
classStringHandling{
publicstaticvoid main(String arg[]){
String s="Java is programming language";
System.out.println(s.endsWith("language"));}}
```

Output: true

11. subString()

subString(): This method is used to get the part of given string.

Example:1 **Output**

```
classStringHandling{

publicstaticvoid main(String arg[]){
String s="Java is programming language";
System.out.println(s.substring(8));// 8 is starting index}}
```

programming language

Example:2 **Output**

```
classStringHandling{

publicstaticvoid main(String arg[]){
String s="Java is programming language";
System.out.println(s.substring(8,12));}}
```

prog

12. trim()

trim(): This method remove space which are available before starting of string and after ending of string.

Example **Output**

```
classStringHandling{

publicstaticvoid main(String arg[]){
String s="    Java is programming language    ";
System.out.println(s.trim());}}
```

Java is programming language

13. split()

split(): This method is used to divide the given string into number of parts based on delimiter (special symbols like @ space ,).

Example **Output**

```
classStringHandling{
publicstaticvoid main(String arg[]){
String s="contact@tutorial4us.com";
String[] s1=s.split("@");// divide string based on @
for(String c:s1)//  foreach loop {
System.out.println(c);}}}
```

```
contact
@tutorial4us.com
```

14. replace()

replace(): This method is used to return a duplicate string by replacing old character with new character.

Note: In this method data of original string will never be modify.

Example **Output**

```
classStringHandling{
publicstaticvoid main(String arg[]){
String s1="java";
String s2=s1.replace('j','k');
System.out.println(s2);}}
```

```
kava
```

Immutable String in Java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

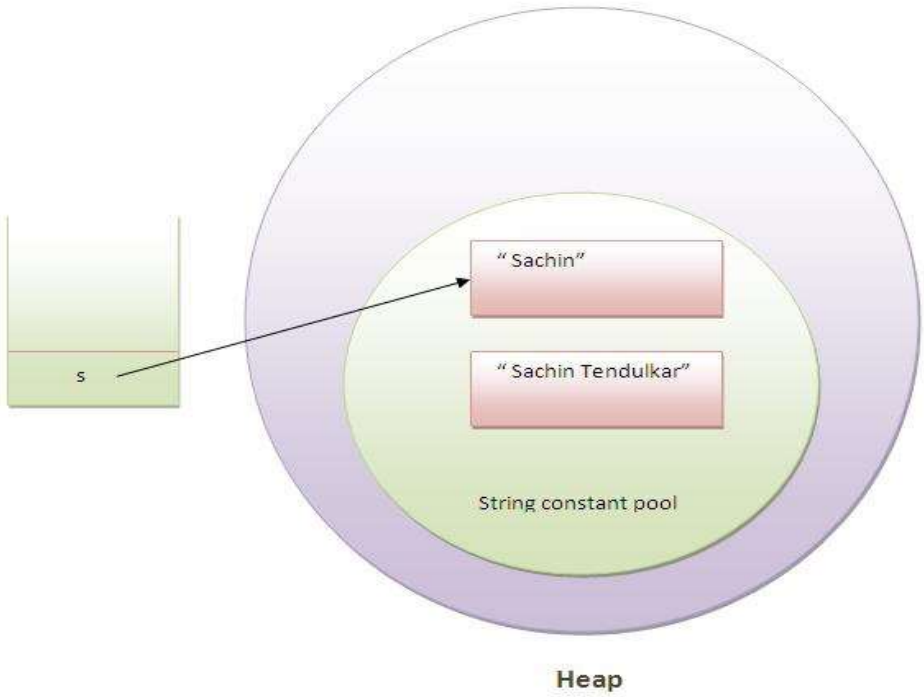
Let's try to understand the immutability concept by the example given below:

```
class Testimmutablestring{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the end
   System.out.println(s);//will print Sachin because strings are immutable objects  } }
```

Example:

Output:Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.



As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

```
class Testimmutablestring1{

 public static void main(String args[]){

   String s="Sachin";

   s=s.concat(" Tendulkar");

   System.out.println(s);   }  }
```

Example:

Output:Sachin Tendulkar

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

Passing Strings To & From Methods

Java StringBuffer class

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

Important Constructors of StringBuffer class

Constructor	Description
StringBuffer()	Creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)	Creates a string buffer with the specified string.

StringBuffer(int capacity)	Creates an empty string buffer with the specified capacity as length.

Important methods of StringBuffer class

What is mutable string?

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

1) StringBuffer append() method

The append() method concatenates the given argument with this string.

Example:

```
class StringBufferExample{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java  }  }
```

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

Example:

```
class StringBufferExample2{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
```

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello } }

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

Example:

```
class StringBufferExample3{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo } }
```

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

Example:

```
class StringBufferExample4{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello"); sb.delete(1,3);
System.out.println(sb);//prints Hllo } }
```

5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

Example:

```
class StringBufferExample5{
```

```
public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello");

sb.reverse();

System.out.println(sb);//prints olleH  }  }
```

6) StringBuffer capacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

Example:

```
class StringBufferExample6{

public static void main(String args[]){

StringBuffer sb=new StringBuffer();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2  }  }
```

Java Scanner class

There are various ways to read input from the keyboard; the java.util.Scanner class is one of them.

The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values.

Java Scanner class is widely used to parse text for string and primitive types using regular expression.

Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

Commonly used methods of Scanner class

There is a list of commonly used Scanner class methods:

Method	Description
public String next()	it returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

Java Scanner Example to get input from console

Let's see the simple example of the Java Scanner class which reads the int, string and double value as an input:

```
import java.util.Scanner;

class ScannerTest{

 public static void main(String args[]){

 Scanner sc=new Scanner(System.in);

 System.out.println("Enter your rollno");

  int rollno=sc.nextInt();

  System.out.println("Enter your name");

  String name=sc.next();

  System.out.println("Enter your fee");

  double fee=sc.nextDouble();

  System.out.println("Rollno:"+rollno+" nam
e:"+name+" fee:"+fee);

 sc.close();  }  }
```

Output:

```
Enter your rollno

111

Enter your name

Ratan

Enter

450000

Rollno:111 name:Ratan fee:450000
```

Java Scanner Example with delimiter

Let's see the example of Scanner class with delimiter. The \s represents whitespace.

```
import java.util.*;

public class ScannerTest2{

public static void main(String args[]){

String input = "10 tea 20 coffee 30 tea buisc
uits";

Scanner s = new Scanner(input).useDelimite
r("\\s");

 System.out.println(s.nextInt());

 System.out.println(s.next());

System.out.println(s.nextInt());

System.out.println(s.next());   s.close();   }}
```

Output:

10	20
tea	coffee

Simple I/O using System. out

Java I/O (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in java** by Java I/O API.

Stream

A stream is a sequence of data. In Java a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In java, 3 streams are created for us automatically. All these streams are attached with console.

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream

Let's see the code to print **output and error** message to the console.

```
System.out.println("simple message");
System.err.println("error message");
```

Let's see the code to get **input** from console.

```
int i=System.in.read();//returns ASCII code of 1st character
System.out.println((char)i);//will print the character
```

OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

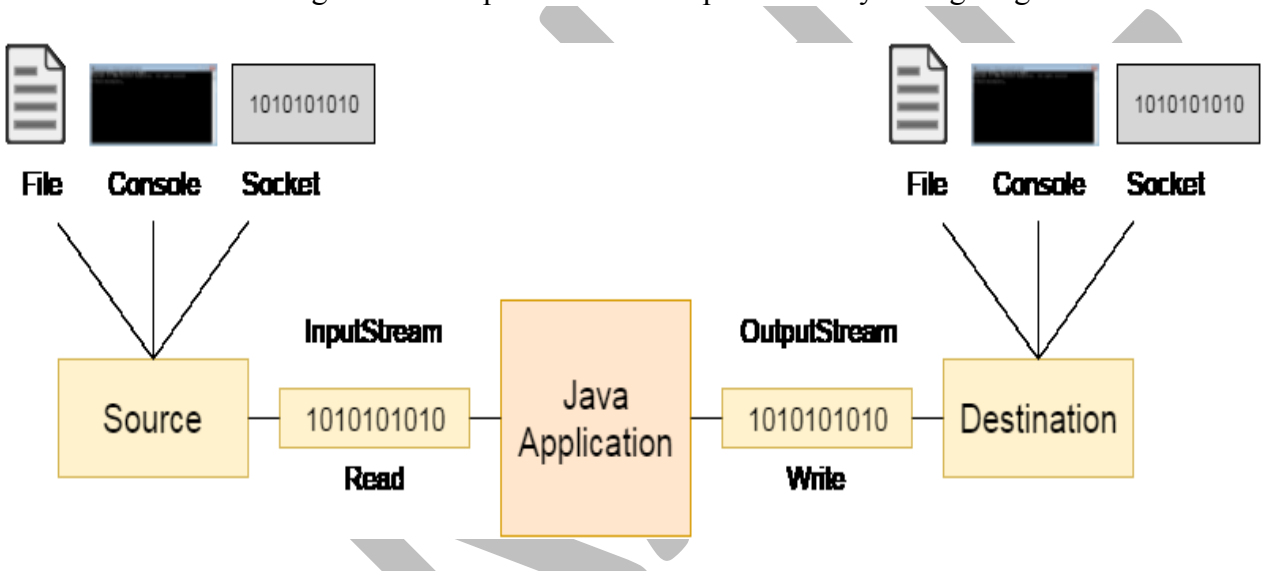
OutputStream

Java application uses an output stream to write data to a destination, it may be a file, an array, peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source, it may be a file, an array, peripheral device or socket.

Let's understand working of Java OutputStream and InputStream by the figure given below.



OutputStream class

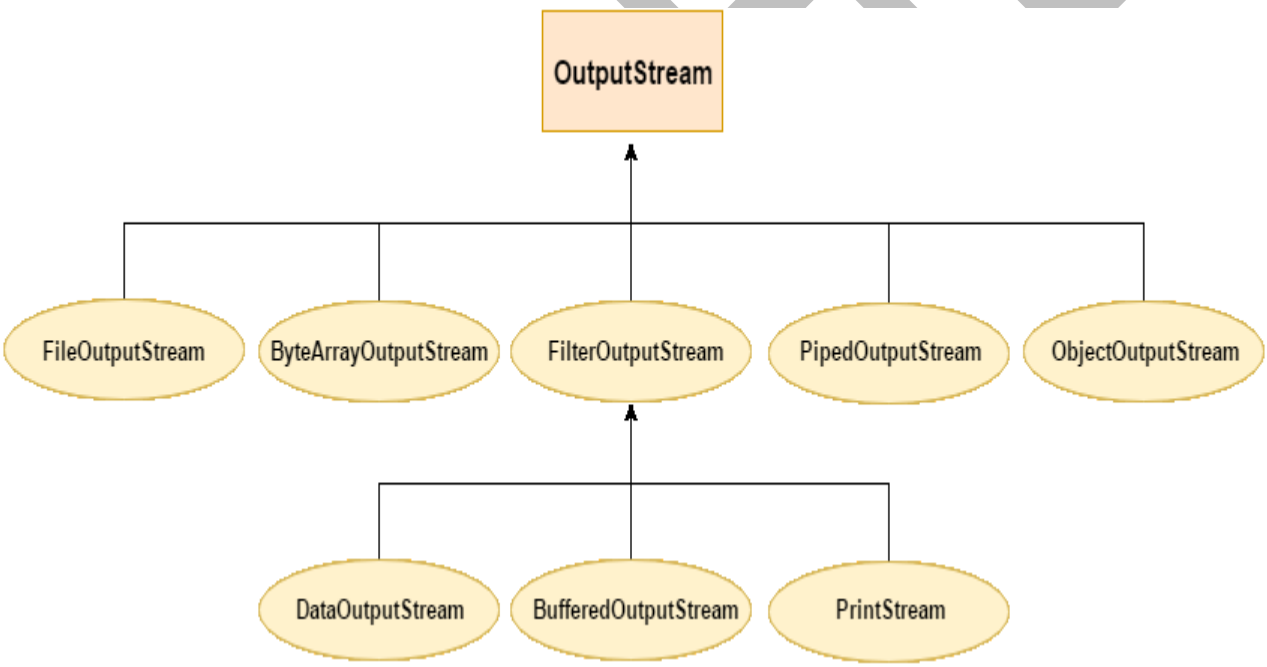
OutputStream class is an abstract class. It is the super class of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Useful methods of OutputStream

Method	Description

1) public void write(int)throws IOException	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException	flushes the current output stream.
4) public void close()throws IOException	is used to close the current output stream.

OutputStream Hierarchy



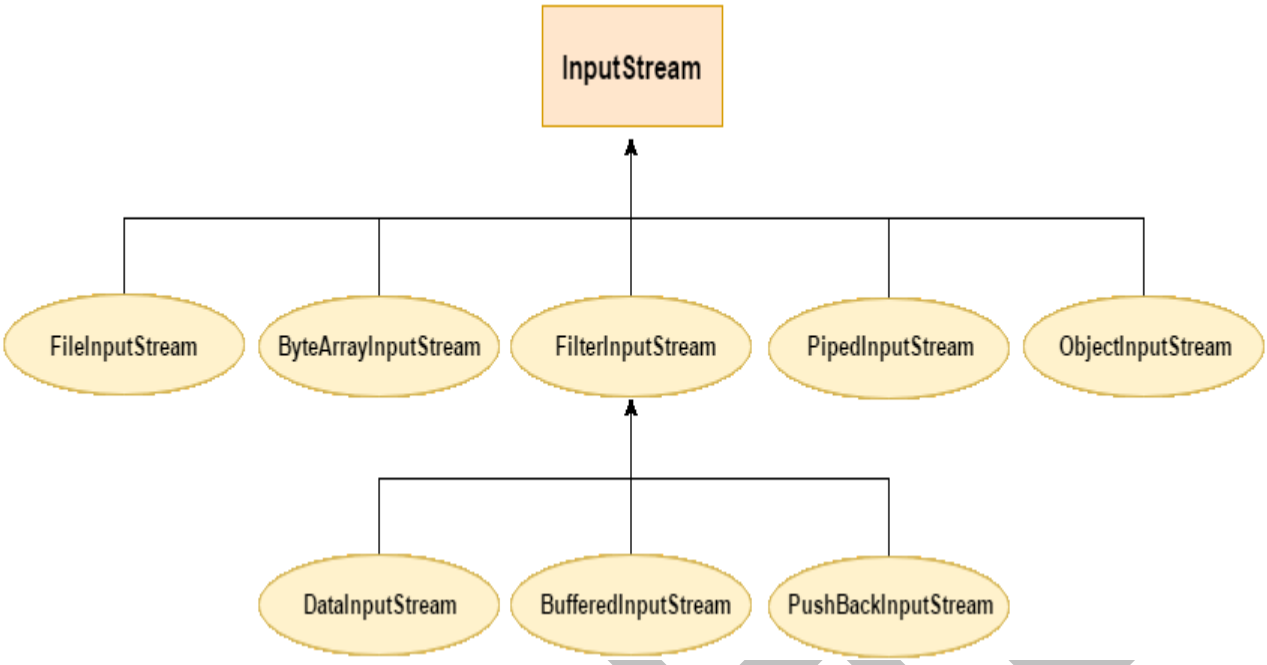
InputStream class

InputStream class is an abstract class. It is the super class of all classes representing an input stream of bytes.

Useful methods of InputStream

Method	Description
1) public abstract int read()throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of file.
2) public int available()throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close()throws IOException	is used to close the current input stream.

InputStream Hierarchy



Java FileOutputStream class

Character Stream Vs Byte Stream in Java

I/O Stream

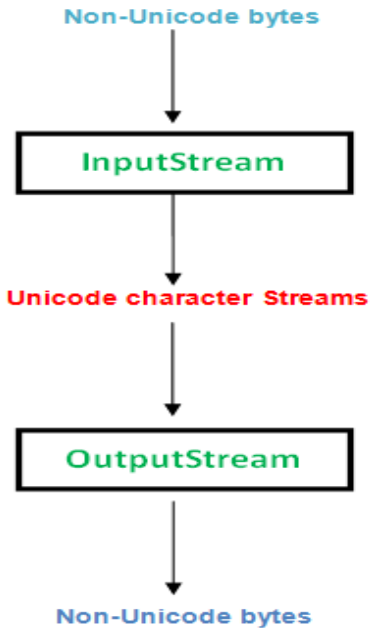
A stream is a method to sequentially access a file. I/O Stream means an input source or output destination representing different types of sources e.g. disk files.The java.io package provides classes that allow you to convert between Unicode character streams and byte streams of non-Unicode text.

Stream – A sequence of data.
Input Stream: reads data from source.
Output Stream: writes data to destination.

Character Stream

In Java, characters are stored using Unicode conventions (Refer this for details). Character stream automatically allows us to read/write data character by character. For example FileReader and FileWriter are

character streams used to read from source and write to destination.

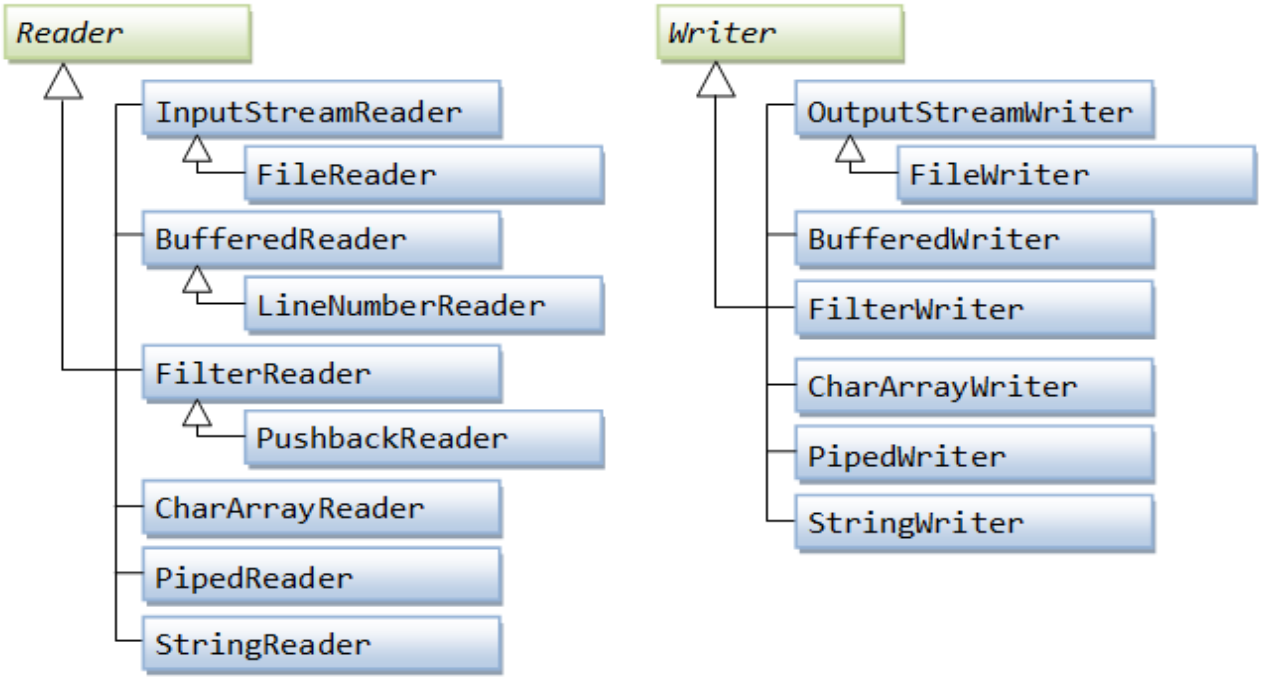



```
// Java Program illustrating that we can read a file in
// a human readable format using FileReader
importjava.io.*;   // Accessing FileReader, FileWriter, IOException
publicclassGfG{
    publicstaticvoidmain(String[] args) throwsIOException    {
        FileReader sourceStream = null;
        try
        {
            sourceStream = newFileReader("test.txt");
             // Reading sourcefile and writing content to
            // target file character by character.
            inttemp;
```

```
        while((temp = sourceStream.read()) != -1)
            System.out.println((char)temp);        }
    finally       {
        // Closing stream as no longer in use
        if(sourceStream != null)
            sourceStream.close();          }    }}
```

Output:

Shows contents of file test.txt

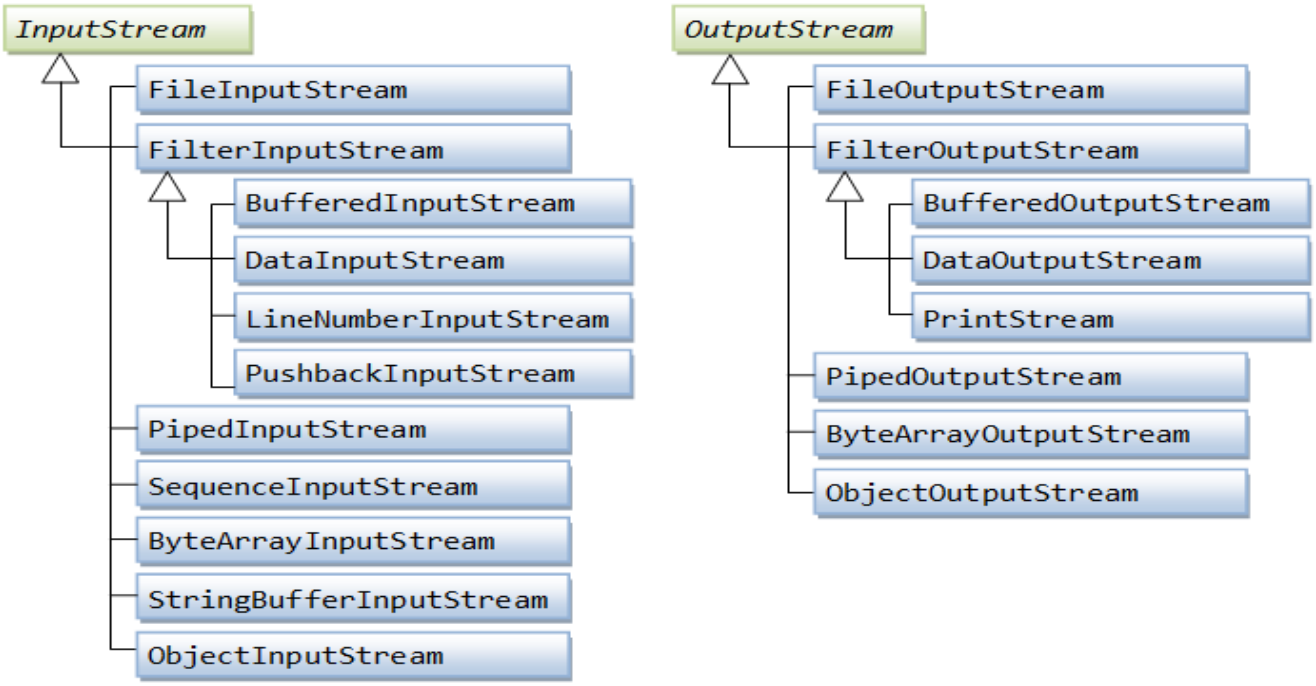


Byte Stream

Byte streams process data byte by byte (8 bits). For example FileInputStream is used to read from source and FileOutputStream to write to the destination.

// Java Program illustrating the Byte Stream to copy

```
// contents of one file to another file.
importjava.io.*;
publicclassBStream{
   publicstaticvoidmain(String[] args) throwsIOException    {
      FileInputStream sourceStream = null;
      FileOutputStream targetStream = null;
       try       {
         sourceStream = newFileInputStream("sorcefile.txt");
         targetStream = newFileOutputStream ("targetfile.txt");
          // Reading source file and writing content to target
         // file byte by byte
         inttemp;
         while((temp = sourceStream.read()) != -1)
            targetStream.write((byte)temp);                    }
      finally       {
         if(sourceStream != null)
            sourceStream.close();
         if(targetStream != null)
            targetStream.close();                  }    }}
```

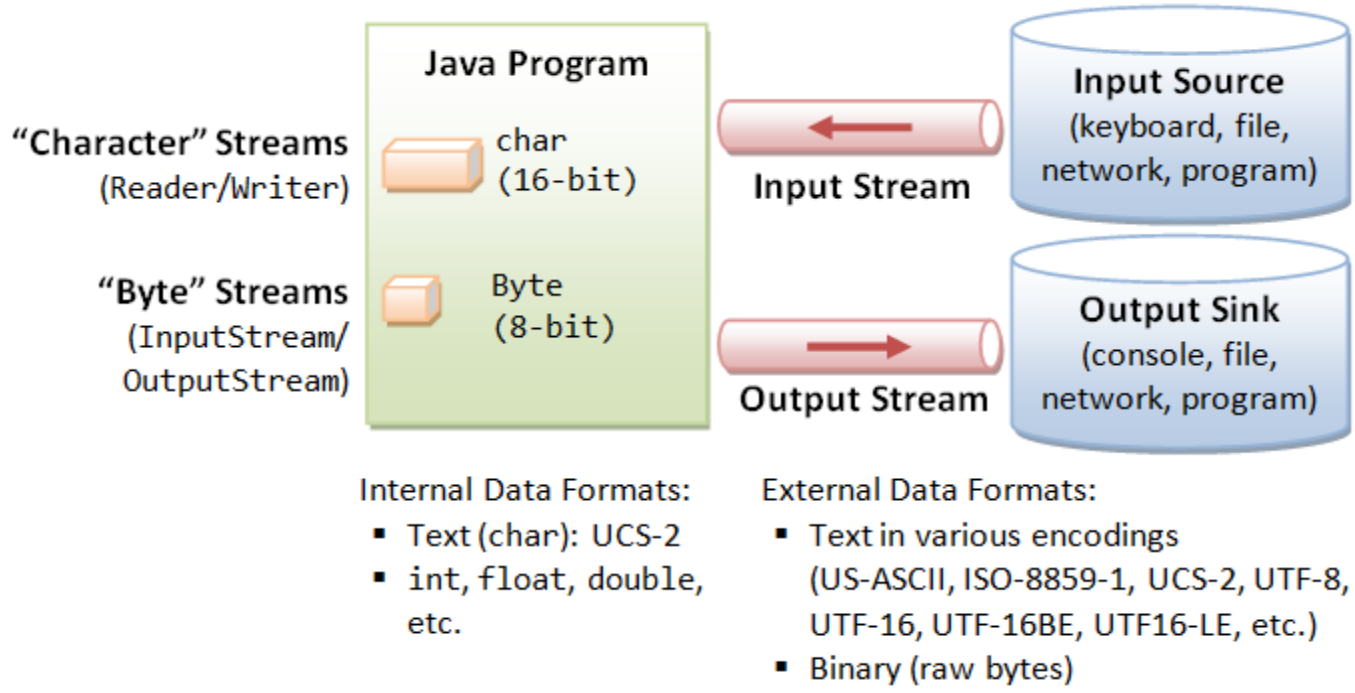


When to use Character Stream over Byte Stream?

- In Java, characters are stored using Unicode conventions. Character stream is useful when we want to process text files. These text files can be processed character by character. A character size is typically 16 bits.

When to use Byte Stream over Character Stream?

- Byte oriented reads byte by byte. A byte stream is suitable for processing raw data like binary files.



Notes:

- Names of character streams typically end with Reader/Writer and names of byte streams end with InputStream/OutputStream
- The streams used in example codes are unbuffered streams and less efficient. We typically use them with buffered readers/writers for efficiency. We will soon be discussing use BufferedReader/BufferedWriter (for character stream) and BufferedInputStream/BufferedOutputStream (for byte stream) classes.
- It is always recommended to close the stream if it is no longer in use. This ensures that the streams won't be affected if any error occurs.
- The above codes may not run in online compilers as files may not exist.

File I/O

In Java, we can read data from files and also write data in files.

We do these using streams. Java has many input and output streams that are used to read and write data. Same as a continuous flow of water is called water stream, in the same way input and output flow of data is called stream.

Stream

Java provides many input and output stream classes which are used to read and write.

Streams are of two types.

- **Byte Stream**
- **Character Stream**

Let's look at the two streams one by one.

Byte Stream

It is used in the input and output of byte.

We do this with the help of different Byte stream classes. Two most commonly used Byte stream classes are **FileInputStream** and **FileOutputStream**. Some of the Byte stream classes are listed below.

Byte Stream class	Description
BufferedInputStream	handles buffered input stream
BufferedOutputStream	handles buffered output stream
FileInputStream	used to read from a file
FileOutputStream	used to write to a file

CO

Character Stream class	Description
BufferedReader	handles buffered input stream

InputStream	Abstract class that describe input stream
OutputStream	Abstract class that describe output stream

Character Stream

It is used in the input and output of characters.

For input and output of characters, we have Character stream classes. Two most commonly used Character stream classes are **FileReader** and **FileWriter**. Below is the list of some Character Stream classes.

BufferedWriter	handles buffered output stream
FileReader	used to read from a file
FileWriter	used to write to a file
InputStreamReader	translate input from byte to character
OutputStreamReader	translate character to byte output
Reader	Abstract class that describe input stream
Writer	Abstract class that describe output stream

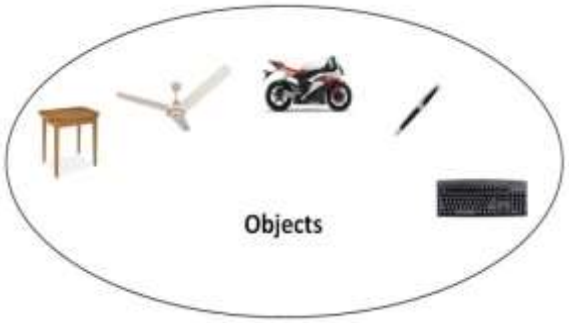
Java OOPs Concepts

Object Oriented Programming is a paradigm that provides many concepts such as **inheritance**, **data binding**, **polymorphism** etc.

Simula is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.

Smalltalk is considered as the first truly object-oriented programming language.

OOPs (Object Oriented Programming System)



Object means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

Encapsulation

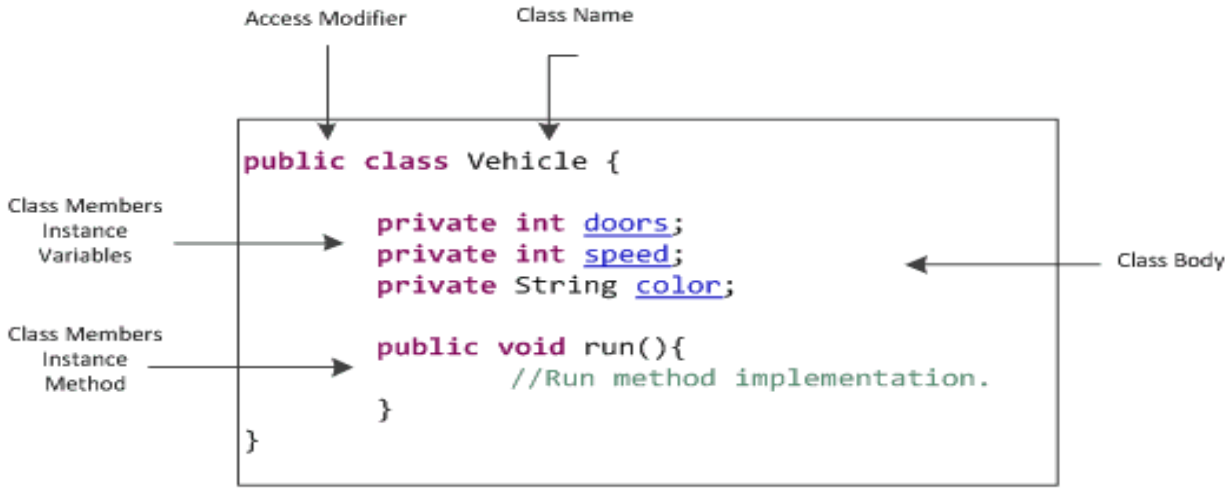
Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Defining & Using Classes

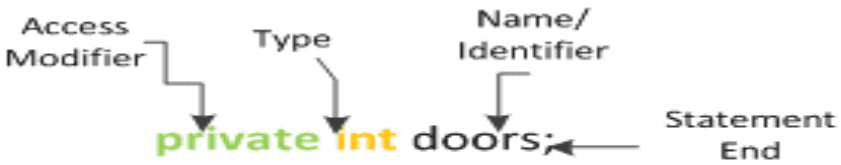
Declaration of Class:

A class is declared by use of the class keyword. The class body is enclosed between curly braces {and}. The data or variables, defined within a class are called instance variables. The code is contained within methods. Collectively, the methods and variables defined within a class are called members of the class.



Declaration of Instance Variables:

Variables defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables. Thus, the data for one object is separate and unique from the data for another. An instance variable can be declared public or private or default (no modifier). When we do not want our variable's value to be changed out-side our class we should declare them private. public variables can be accessed and changed from outside of the class. We will have more information in OOP concept tutorial. The syntax is shown below.



Access Modifiers in java

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.

The access modifier in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

Constructor in Java

Constructor in java is a *special type of method* that is used to initialize the object.

Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

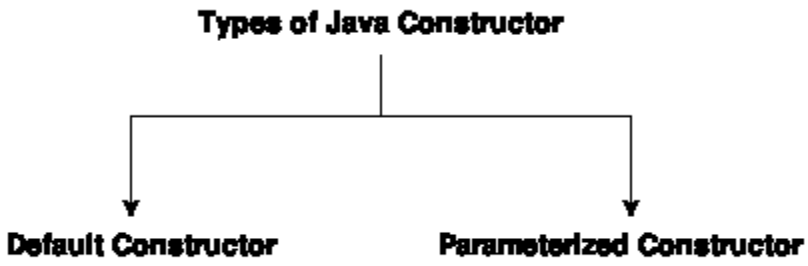
There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Java Default Constructor

A constructor that has no parameter is known as default constructor.

Syntax of default constructor:

1. <class_name>(){}

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1{

Bike1(){System.out.println("Bike is created");}

public static void main(String args[]){
```

Bike1 b=new Bike1(); } }

Output:

Bike is created

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type.

Example of default constructor that displays the default values

```
class Student3{
int id;
String name;
void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
Student3 s1=new Student3();
Student3 s2=new Student3();
s1.display();
s2.display(); }
```

Output:

0 null

0 null

Explanation: In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java parameterized constructor

A constructor that has parameters is known as parameterized constructor.

Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

Example of parameterized constructor

In this example, we have created the constructor of Student class that has two parameters. We can have any number of parameters in the constructor.

```
class Student4{
    int id;
    String name;
    Student4(int i,String n){
    id = i;
    name = n;      }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    s1.display();
    s2.display();     }  }
```

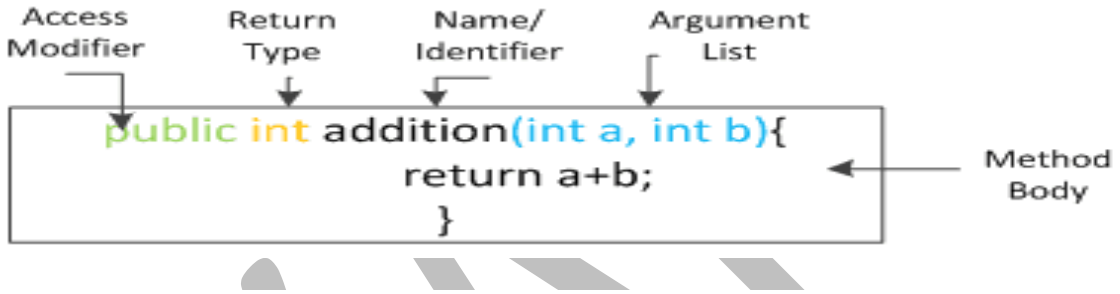
Output:

111 Karan

222 Aryan

Declaration of Methods:

A method is a program module that contains a series of statements that carry out a task. To execute a method, you invoke or call it from another method; the calling method makes a method call, which invokes the called method. Any class can contain an unlimited number of methods, and each method can be called an unlimited number of times. The syntax to declare method is given below.



Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

In java, Method Overloading is not possible by changing the return type of the method only.

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add () methods perform addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
 }
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));         }}
```

Output:

```
22
33
```

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differ in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
 class Adder{
static int add(int a, int b){return a+b;}
```

```
 static double add(double a, double b){return a+b;}
 }
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));  }}
```

Output:
22
24.9

Java passing object as parameter

Passing Object as Parameter:

```
package com.pritesh.programs;
class Rectangle {
   int length;
   int width;
Rectangle(int l, int b) {
       length = l;
       width = b;    }
   void area(Rectangle r1) {
       int areaOfRectangle = r1.length * r1.width;
System.out.println("Area of Rectangle : "  + areaOfRectangle);    }}
class RectangleDemo {
publicstatic void main(String args[]) {
Rectangle r1 = newRectangle(10, 20);
       r1.area(r1);    }}
```

Output of the program:

Area of Rectangle: 200

Explanation:

We can pass Object of any class as parameter to a method in java.

1. We can access the instance variables of the object passed inside the called method.

area = r1.length * r1.width

3. It is good practice to initialize instance variables of an object before passing object as parameter to method otherwise it will take default initial values.

Different Ways of Passing Object as Parameter:

Way 1 : By directly passing Object Name

```
void area(Rectangle r1) {
int areaOfRectangle = r1.length * r1.width;
      System.out.println("Area of Rectangle : "  + areaOfRectangle);    }
class RectangleDemo {
   public staticvoid main(String args[]) {
Rectangle r1 = new Rectangle(10, 20);
      r1.area(r1);    }
```

Way 2 : By passing Instance Variables one by one

```
package com.pritesh.programs;
class Rectangle {
int length;
int width;
void area(int length, int width) {
int areaOfRectangle = length * width;
```

```
    System.out.println("Area of Rectangle : "   + areaOfRectangle);   }}
class RectangleDemo {
   public staticvoid main(String args[]) {
      Rectangle r1 = new Rectangle();
      Rectangle r2 = new Rectangle();
      r1.length = 20;
      r1.width = 10;
      r2.area(r1.length, r1.width);    }}
```

Actually this is not a way to pass the object to method. but this program will explain you how to pass instance variables of particular object to calling method.

Way 3 : We can pass only public data of object to the Method.

Suppose we made width variable of a class private then we cannot update value in a main method since it does not have permission to access it.

```
private int width;
```

after making width private –

```
class RectangleDemo {
   public staticvoid main(String args[]) {
      Rectangle r1 = new Rectangle();
      Rectangle r2 = new Rectangle();
      r1.length = 20;
      r1.width = 10;
      r2.area(r1.length, r1.width);    }}
```

Final Keyword in Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable

2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or un initialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike{}
class Honda1 extends Bike{
void run(){System.out.println("running safely with 100kmph");}
public static void main(String args[]){
Honda1 honda= new Honda1();
honda.run();    }  }
```

Output: Compile Time Error

Object class in Java:

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

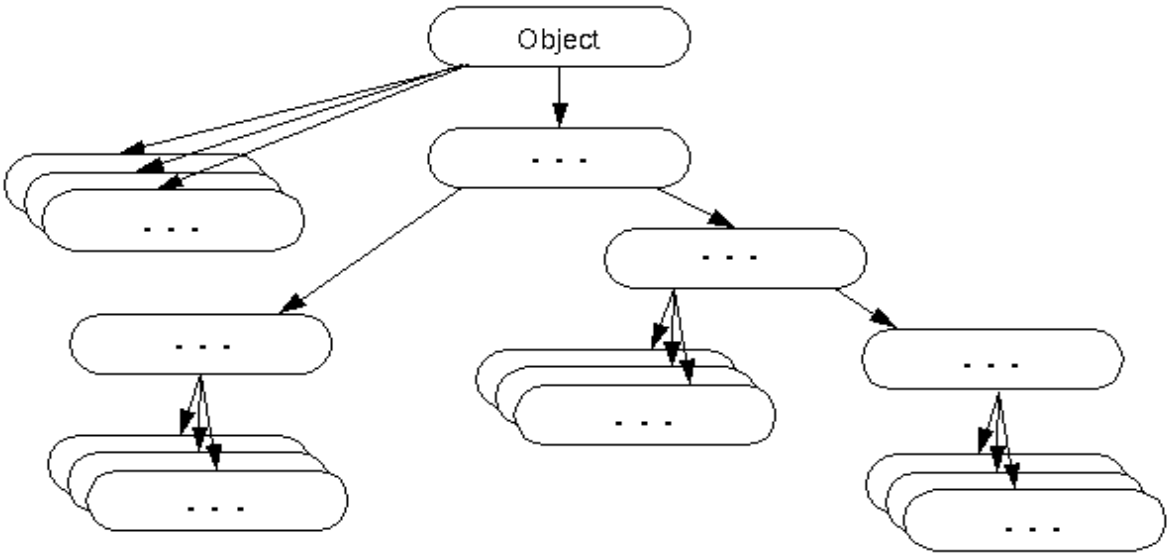
The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, we can use Object class reference to refer that object.

For example:

```
Object obj=getObject();//we don't know what object will be returned from this method
```

The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.



Methods of Object class:

The Object class provides many methods. They are as follows:

Method	Description
public final Class getClass()	Returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws	creates and returns the exact copy

CloneNotSupportedException	(clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

Java Garbage Collection:

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By annonymous object etc.

1) By nulling a reference:

```
Employee e=new Employee();
e=null;
```

2) By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is available for garbage collection
```

3) By annonymous object:

new Employee();

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:https://www.javatpoint.com/Garbage-Collectionhttps://www.javatpoint.com/Garbage-Collectionhttps://www.javatpoint.com/Garbage-Collection

protected void finalize(){}

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.https://www.javatpoint.com/Garbage-Collectionhttps://www.javatpoint.com/Garbage-Collectionhttps://www.javatpoint.com/Garbage-Collection

public static void gc(){}

Note: Garbage collection is performed by a daemon thread called Garbage Collector (GC). This thread calls the finalize() method before object is garbage collected.

Simple Example of garbage collection in java

```
public class TestGarbage1{
public void finalize(){System.out.println("object is garbage collected");}
public static void main(String args[]){
    TestGarbage1 s1=new TestGarbage1();
    TestGarbage1 s2=new TestGarbage1();
    s1=null;
    s2=null;
```

Output:

```
object is garbage collected
object is garbage collected
```

```
System.gc();   }  }
```

Note: Neither finalization nor garbage collection are guaranteed.

POSSIBLE QUESTIONS

2 MARKS

1. What is Java Array and its types?
2. What are the Advantage of Java Array?
3. Mention the Disadvantage of Java Array?
4. What is Dynamic Arrays?
5. Define Java String Class?
6. What is Immutable String in Java give example.
7. What is mutable string?
8. What is Stream?
9. What is OutputStream?
10. What is InputStream?
11. Define I/O Stream?
12. Define Character Stream and Byte Stream?
13. What are the Access Modifiers in java?
14. Mention the Types of java constructors.
15. What is the purpose of default constructor

6 MARKS

1. Differentiate between String and String Buffer classes. Also write a program to append a given string.

2. Describe Class Constructors with example.

3. Illustrate Method Overloading with suitable program.

4. Explain 1-dimensional and multi-dimensional arrays in detail.

5. Explain the following with suitable program i) Garbage Collection ii) final classes

6. Write any five methods available in StringBuffer and write a program to reverse a string using StringBuffer class.

7. Enlighten about the String Immutability & Equality with appropriate examples.

8. Explain the types of Array in a JAVA with examples.

9. Elaborate creation and operation on Strings with examples.

10. Explain OOPs (Object Oriented Programming System) in Java with example.

KARPAGAM ACADEMY OF HIGHER EDUCATION

COIMBATORE - 21

DEPARTMENT OF COMPUTER SCIENCE,CA & IT

CLASS : I BCA

BATCH : 2018-2021

Part -A Online Examinations (1 mark questions)

SUBJECT: Programming in Java SUBJECT CODE: 18CAU201

UNIT II

S.No	Questions	opt1	opt2	opt3	opt4	Answer
1	A _____never returns a value	class	function	method	constructor	constructor
2	_____can be used to initialize the fields in the	instance variables	constructors	methods	none	constructors
3	_____refers to the use of same thing for different	Overloading	Dynamic binding	message loading	none	Overloading
4	Single function name can be used to handle different types	function overloading	operator overloading	polymorphism	encapsulatio	function overloading
5	It is used to initialize the member variables when we	Constructors	destructors	Overloading	Overriding	Constructors
6	It takes no parameters	Default Constructors	Copy Constructors	Parameter Constructor	Function	Default Constructors
7	It is required when objects are required to perform a similar	Method Overriding	Polymorphism	Static Binding	Method Overloading	Method Overloading
8	_____ is an object that contains elements of same	Array	Structure	Class	Object	Array
9	What is the representation of the third element in an array	a[2]	a(2)	a[3]	a(3)	a[2]
10	Which of the following is correct?	int[] a = new int[2];	int a[] = new int[2];	int[] a = new int(2);	int a() = new int[2];	int[] a = new int[2];
11	Which of the following statements is valid?	int i = new int(30);	double d[] = new double[30];	char[] c = new char[4]{'a', 'b',	char[] c = new char();	double d[] = new double[30];
12	the length of a string by calling the ____ method	strlen()	len()	length( )	none	length( )
13	the character at a specified index within a string by	charAt( )	chatat()	char()	character()	charAt( )
14	To extract a single character from a string , the	charAt	Stringto	charone	None	charAt
15	To get the substring from a string _____ method is	getchars	substr	extract	substring	getchars
16	The _____ method compares the characters inside	= =	equivalent	equals	None	equals

17	The _____ operator compares two objects	= =	equivalent	equals	None	= =
18	The String method _____ can be used to	StringTo	CompareTo	Compare	CompareOf	CompareTo
19	If the integer result of CompareTo is negative, then	Equal	Less	Greater	None	Less
20	If the integer result of CompareTo is positive, then	Equal	Less	Greater	None	Greater
21	The search for a certain character or substring is done	index & indexof	index & lastindex	indexof & lastindexof	None	indexof & lastindexof
22	The replace method takes _____ characters as	1	2	3	4	2
23	_____ represents fixed length immutable character	String	Characters	Variable	Identifier	String
24	When you extends a class, you can change the behavior of a	method overriding.	object refernce	method overloading	polymorphis	method overriding.
25	To add a finalizer to a class, you simply define the _____	finalize( )	stop()	exit()	none	finalize( )
26	the new operator dynamically _____memory for an	free	allocates	delete	none	allocates
27	_____ dispatch is the mechanism by which a call to	Static method	Dynamic method	overload	none	Dynamic method
28	_____ is the one, which creates more than one methods	method overriding	method overloading	function call	inheritance	method overloading
29	static methods will not refer the _____	this	dot	new	public	this
30	._____ is used to allocate memory in the constructor	Delete	Binding	Free	new	new
31	Java supports a concept called _____ which is just	free	finalization	delete	new	finalization
32	A class that cannot be subclassed is called as	abstract	final	static	methods	final
33	._____ enables an object to initialize itself when it is	Destructor	constructor	overloading	none	constructor
34	Subclass constructors can call superclass constructors via the	final	protected	inherit	super	super
35	.The _____ is special because its name is the same	Destructor	static	constructor	none of the above	constructor
36	.A constructor that accepts no parameters is called the	Copy	default	multiple	none of the above	default
37	.Constructors are invoked automatically when the	Datas	classes	objects	none of the above	objects
38	.Constructors cannot be _____	Inherited	destroyed	both a & b	none of the above	Inherited

39	The constructors that can take arguments are called	Copy	multiple	parameterized	none of the above	parameterized
40	The concept of reading and writing data as	stream	file	java.io	reader	stream
41	Java also uses the _____ class to manipulate files	stream	File	String	Array	File
42	To support input and output package _____is used	java.util	java.awt	java.lang	java.io	java.io
43	_____ support 8_bit input and output	ByteStreams	InputStream	OutputStream	Writer	ByteStreams
44	_____ support 16_bit Unicode character	ByteStreams	InputStream	OutputStream	Character streams	Character streams
45	Streams can be chained with _____ to provide	DataInput	DataOutput	filters	serializable	filters
46	_____ class in java does not specify how information is	stream	File	String	Array	File
47	The _____ class also defines platform_dependent	stream	File	java.io	reader	File
48	_____ class defines Java's model of streaming byte	ByteStreams	InputStream	OutputStream	Character streams	InputStream
49	InputStream suports certain methods, all of which throw	ByteStreams	InputStream	OutputStream	Character streams	InputStream
50	The _____ class define byte input streams that are	InputStream	OutputStream	FileInputStream	FileOutputStream	FileInputStream
51	The _____ class define byte output streams that are	InputStream	OutputStream	FileInputStream	FileOutputStream	FileOutputStream
52	The FileInputStream class provides an implementation	read()	write()	update()	replace()	read()
53	The FileOutputStream class provides an implementation	read()	write()	update()	replace()	write()
54	The method provided by the Reader class is	skip()	write()	flush()	writeX()	skip()
55	The method provided by the Writer class is	read()	flush()	reset()	skip()	flush()
56	The _____ operator is used to access the instance	new	dot	this	super	dot
57	Methods are called on an instance of a class using the	new	dot	this	super	dot
58	_____ is used inside of any method to refer to the	new	dot	this	super	this
59	The _____ method terminates the program.	System.terminate(0);	System.halt(0);	System.exit(0);	System.stop(0);	System.exit(0);

60	The file produced by java compiler ends with _____	Java	html	class	applet	class

UNIT-III

SYLLABUS

Inheritance, Interfaces, Packages, Enumerations, Auto boxing and Metadata

Inheritance: (Single Level and Multilevel, Method Overriding, Dynamic Method Dispatch, Abstract Classes), Interfaces and Packages, Extending interfaces and packages, Package and Class Visibility, Using Standard Java Packages (util, lang, io, net), Wrapper Classes, Auto boxing/Unboxing, Enumerations and Metadata.

Inheritance in Java

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

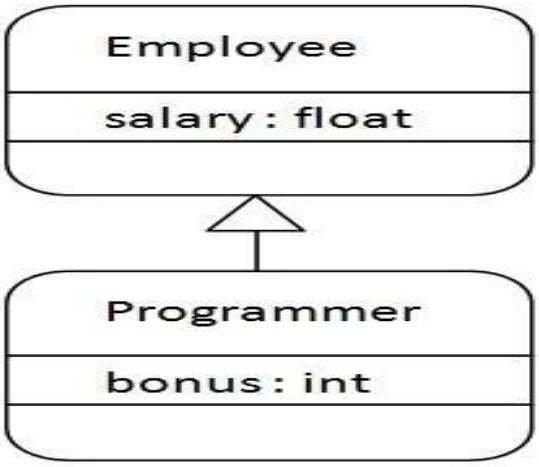
Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called parent or super class and the new class is called child or subclass.

Java Inheritance Example



As displayed in the above figure, Programmer is the subclass and Employee is the super class. Relationship between two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

```
class Employee{
float salary=40000;
}
class Programmer extends Employee{
int bonus=10000;
public static void main(String args[]){
  Programmer p=new Programmer();
  System.out.println("Programmer salary is:"+p.salary);
  System.out.println("Bonus of Programmer is:"+p.bonus);  }  }
```

Example:

Programmer salary is:40000.0

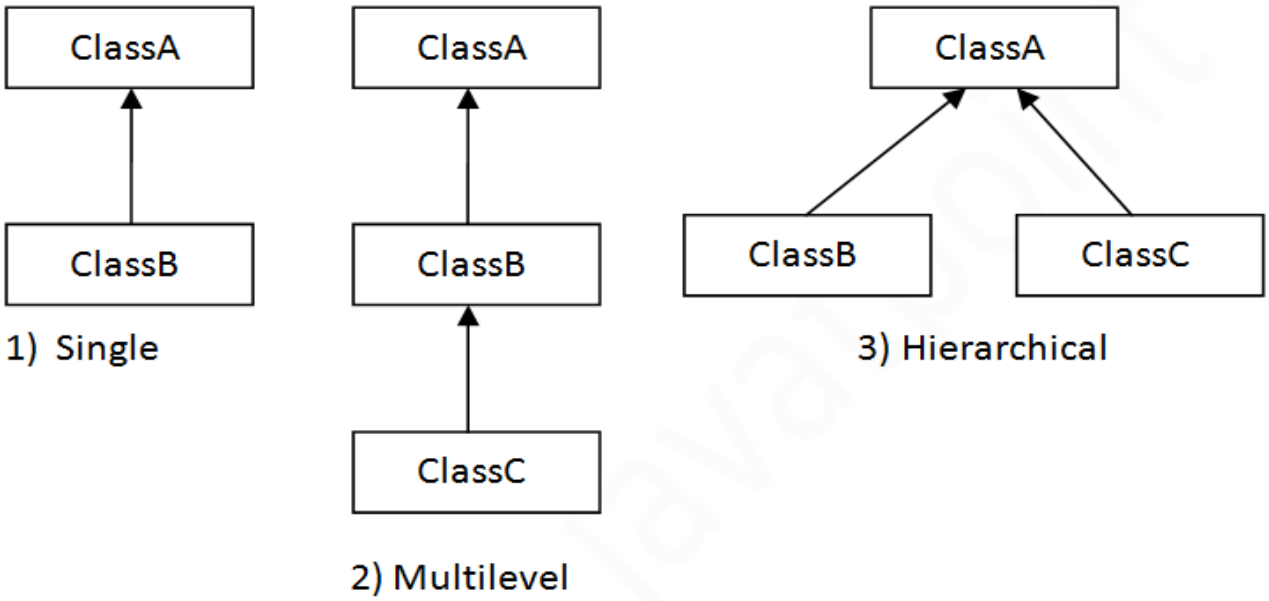
Bonus of programmer is:10000

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

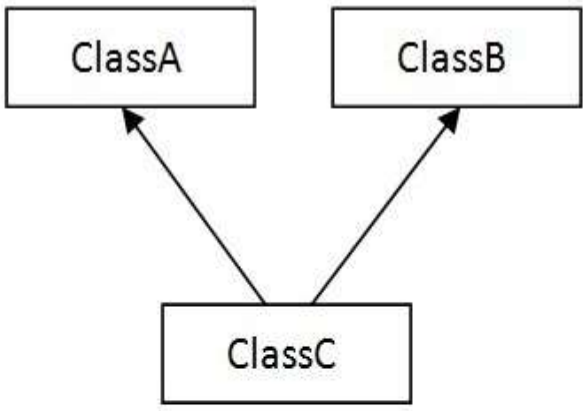
In java programming, multiple and hybrid inheritance is supported through interface only.



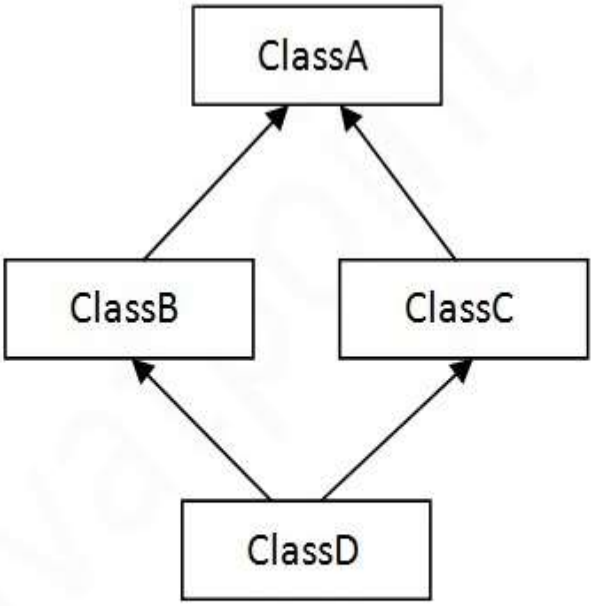
Note: Multiple inheritances are not supported in java through class.

When a class extends multiple classes i.e. known as multiple inheritance.

For Example:



4) Multiple



5) Hybrid

Single Inheritance Example

File: TestInheritance.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
```

```
d.eat();  }}
```

Output:

barking...

eating...

Multilevel Inheritance Example

File: TestInheritance2.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();  }}
```

Output:

weeping...

barking...

eating...

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

In other words, if subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. method must have same name as in the parent class
2. Method must have same parameter as in the parent class.
3. Must be IS-A relationship (inheritance).

Understanding the problem without method overriding

Let's understand the problem that we may face in the program if we don't use method overriding.

```
class Vehicle{
  void run(){System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{
  public static void main(String args[]){
  Bike obj = new Bike();
  obj.run();   }  }
```

Output: Vehicle is running

Problem is that I have to provide a specific implementation of run() method in subclass that is why we use method overriding.

Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

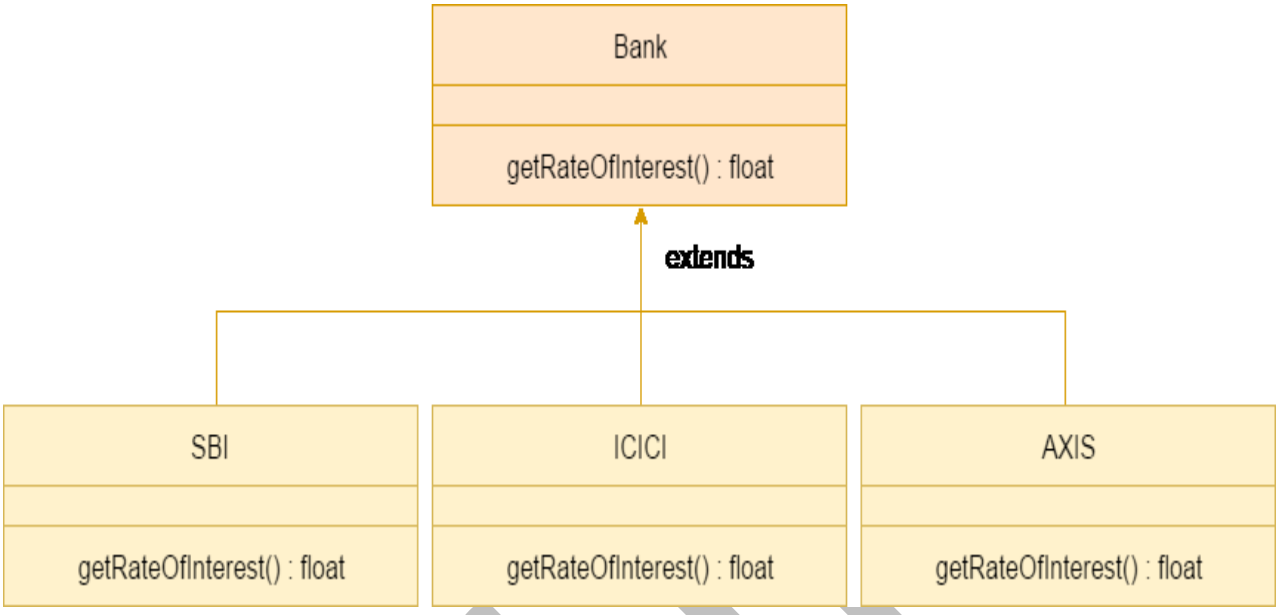
```
class Vehicle{
void run(){System.out.println("Vehicle is running");}
}
class Bike2 extends Vehicle{
void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){
Bike2 obj = new Bike2();
obj.run();
}
```

Output: Bike is running safely

Real example of Java Method Overriding

Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.




```
class Bank{
int getRateOfInterest(){return 0;}
}
 class SBI extends Bank{
int getRateOfInterest(){return 8;}
}
 class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
 class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
```

```
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest()); } }
```

Output:

SBI Rate of Interest: 8

ICICI Rate of Interest: 7

AXIS Rate of Interest: 9

Can we override static method?

No, static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

Why we cannot override static method?

Because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

Can we override java main method?

No, because main is a static method.

Difference between method Overloading and Method Overriding in java

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within*	Method overriding occurs *in*

	class.	*two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

Java Method overloading example

```
class OverloadingExample{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}  }
```

Java Method Overriding example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}
```

Polymorphism in Java

Polymorphism in java is a concept by which we can perform a *single action by different ways*. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in java:

1. Compile time polymorphism
2. Runtime polymorphism.

We can perform polymorphism in java by method overloading and method overriding.

If you overload static method in java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Runtime Polymorphism in Java

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Let's first understand the upcasting before Runtime Polymorphism.

Upcasting

When reference variable of Parent class refers to the object of Child class, it is known as upcasting.

For example:


```
class A{}
class B extends A{}

A a=new B();//upcasting
```

Java Runtime Polymorphism Example: Shape

```
class Shape{
void draw(){System.out.println("drawing...");}
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle...");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle...");}
}
class Triangle extends Shape{
void draw(){System.out.println("drawing triangle...");}
}
class TestPolymorphism2{
public static void main(String args[]){
Shape s;
s=new Rectangle();
s.draw();
s=new Circle();
s.draw();
s=new Triangle();
s.draw();
}
}
```

Output:

```
drawing rectangle...
drawing circle...
drawing triangle...
```

Abstract class in Java

A class that is declared with abstract keyword is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

```
abstract class A{ }
```

Abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

Example abstract method

```
abstract void printStatus();//no body and abstract
```

Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike{
  abstract void run();
```

```
}
class Honda4 extends Bike{
void run(){System.out.println("running safely..");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();  }  }
```

Output: running safely.

Another example of abstract class in java

File: TestBank.java

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}
 class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");   }}
```

Output:

Rate of Interest is: 7 %

Rate of Interest is: 8 %

Abstract class having constructor, data member, methods etc.

An abstract class can have data member, abstract method, method body, constructor and even main() method.

File: TestAbstraction2.java

```
//example of abstract class that have method body
abstract class Bike{
  Bike(){System.out.println("bike is created");}
  abstract void run();
  void changeGear(){System.out.println("gear changed");}
}
  class Honda extends Bike{
void run(){System.out.println("running safely..");}
}
class TestAbstraction2{
public static void main(String args[]){
 Bike obj = new Honda();
 obj.run();
 obj.changeGear();
}
}
```

Output:

```
bike is created
running safely..
gear changed
```

Rule: If there is any abstract method in a class, that class must be abstract.

```
class Bike12{
abstract void run();
}
```

Output: compile time error

Rule: If you are extending any abstract class that has abstract method, you must either provide the implementation of the method or make this class abstract.

Java Package

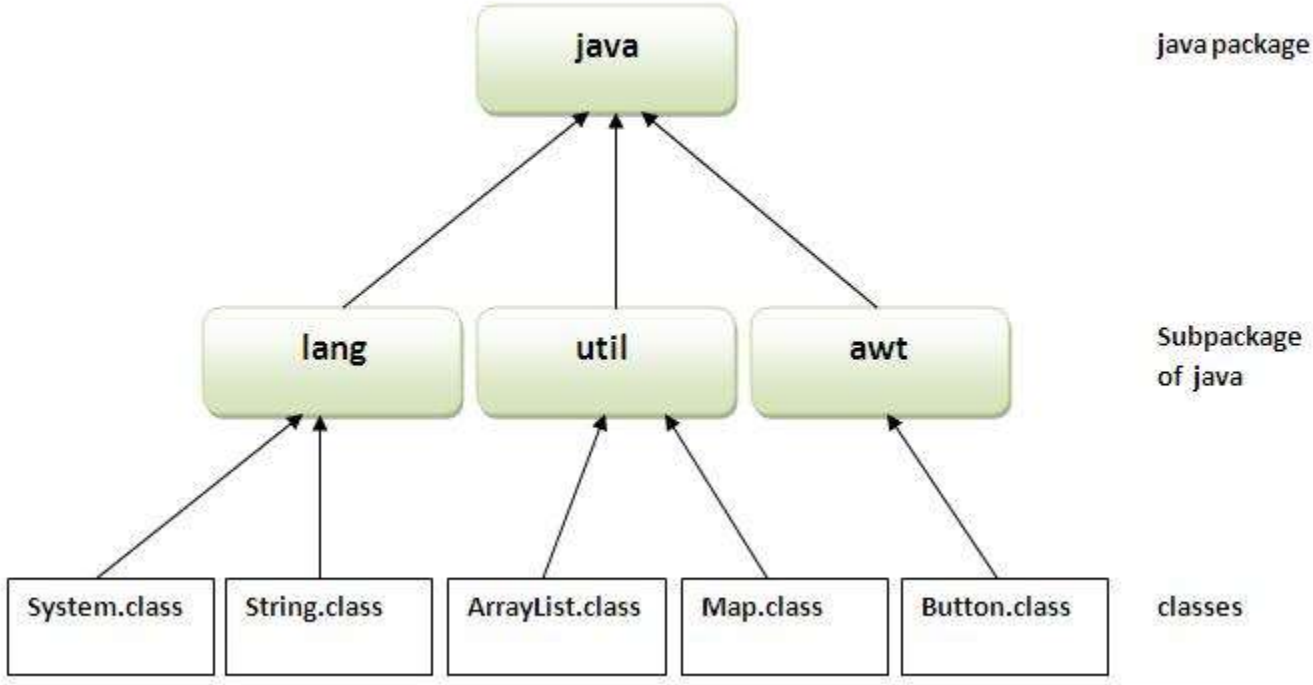
A **java package** is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc. Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");     }  }
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

```
javac -d directory javafilename
```

For **example**

```
javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to run java package program

You need to use fully qualified name e.g. mypack. Simple etc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

Output: Welcome to package

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

1) Using packagename.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;
 class B{
 public static void main(String args[]){
  A obj = new A();
  obj.msg();   }  }
```

Output: Hello

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java
package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.A;
class B{
public static void main(String args[]){
A obj = new A();
 obj.msg();   } }
```

Output: Hello

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

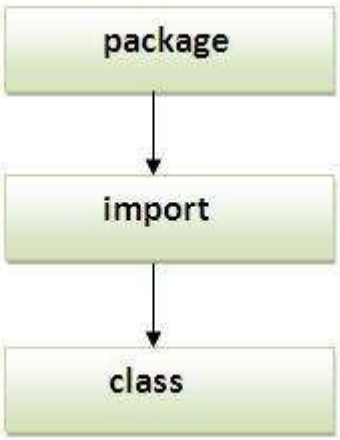
```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
class B{
public static void main(String args[]){
pack.A obj = new pack.A();//using fully qualified name
obj.msg();   }  }
```

Output: Hello

Note: If you import a package, subpackages will not be imported.

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Note: Sequence of the program must be package then import then class.



Subpackage in java

Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.

Let's take an example; Sun Microsystem has definded a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.

The standard of defining package is domain.company.package e.g. com.javatpoint.bean or org.sssit.dao.

Example of Subpackage

```
package com.javatpoint.core;
class Simple{
  public static void main(String args[]){
   System.out.println("Hello subpackage");    }  }
```

To Compile: javac -d . Simple.java

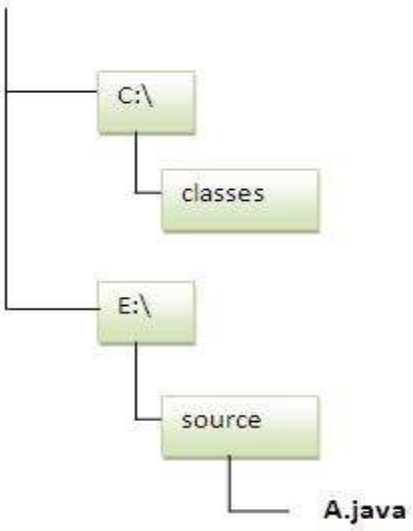
To Run: java com.javatpoint.core.Simple

Output: Hello subpackage

How to send the class file to another directory or drive?

There is a scenario; I want to put the class file of A.java source file in classes' folder of c: drive.

For example:




```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
   System.out.println("Welcome to package");     }  }
```

To Compile:

e:\sources> javac -d c:\classes Simple.java

To Run:

To run this program from e:\source directory, you need to set classpath of the directory where the class file resides.

e:\sources> set classpath=c:\classes;.;

e:\sources> java mypack.Simple

Another way to run this program by -classpath switch of java:

The -classpath switch can be used with javac and java tool.

To run this program from e:\source directory, you can use -classpath switch of java that tells where to look for class file. For example:

e:\sources> java -classpath c:\classes mypack.Simple

Output: Welcome to package

Ways to load the class files or jar files

There are two ways to load the class files temporary and permanent.

- Temporary
  - By setting the classpath in the command prompt
  - By -classpath switch
- Permanent
  - By setting the classpath in the environment variables
  - By creating the jar file, that contains all the class files, and copying the jar file in the jre/lib/ext folder.

Interface in Java

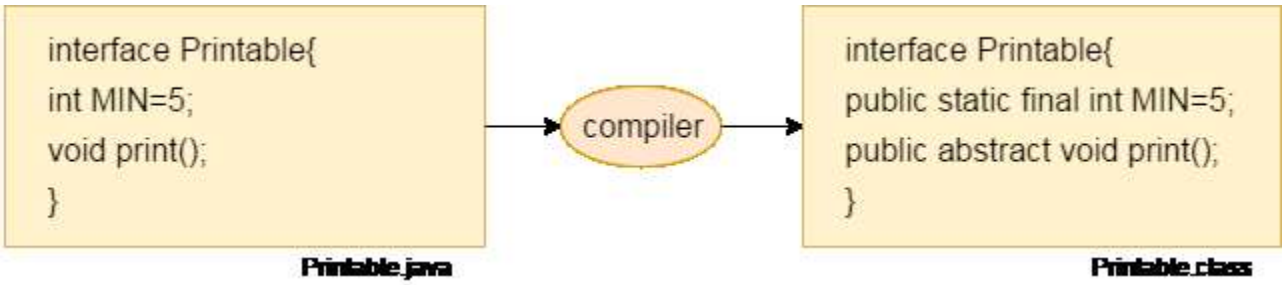
1. An **interface in java** is a blueprint of a class. It has static constants and abstract methods.
2. The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.
3. Java Interface also **represents IS-A relationship**.
4. It cannot be instantiated just like abstract class.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

In other words, Interface fields are public, static and final by default, and methods are public and abstract.



Wrapper class in Java

Wrapper class in java provides the mechanism *to convert primitive into object and object into primitive*.

Since J2SE 5.0, **autoboxing** and **unboxing** feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing.

The eight classes of *java.lang* package are known as wrapper classes in java.

The list of eight wrapper classes is given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float

double	Double

Wrapper class Example: Primitive to Wrapper

```
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
}}
```

Output:

20 20 20

Wrapper class Example: Wrapper to Primitive

```
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int
int j=a;//unboxing, now compiler will write a.intValue() internally
System.out.println(a+" "+i+" "+j);
}}
```

Output:

3 3 3

Autoboxing and Unboxing:

The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing. This is the new feature of Java5. So java programmer doesn't need to write the conversion code.

Advantage of Autoboxing and Unboxing:

No need of conversion between primitives and Wrappers manually so less coding is required.

Simple Example of Autoboxing in java:

```
class BoxingExample1{
  public static void main(String args[]){
    int a=50;
       Integer a2=new Integer(a);//Boxing

       Integer a3=5;//Boxing

       System.out.println(a2+" "+a3);
  }
}
1.
```

Output: 50 5

Simple Example of Unboxing in java:

The automatic conversion of wrapper class type into corresponding primitive type, is known as Unboxing.

Example of unboxing:

```
class UnboxingExample1{
public static void main(String args[]){
  Integer i=new Integer(50);
```

```
        int a=i;
        System.out.println(a);
    }
}
```

Output: 50

Java Enumeration (enum)

Enum in java is a data type that contains fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc. The java enum constants are static and final implicitly. It is available from JDK 1.5.

Java Enums can be thought of as classes that have fixed set of constants.

Points to remember for Java Enum

- enum improves type safety
- enum can be easily used in switch
- enum can be traversed
- enum can have fields, constructors and methods
- enum may implement many interfaces but cannot extend any class because it internally extends Enum class

Simple example of java enum

```
class EnumExample1{
public enum Season { WINTER, SPRING, SUMMER, FALL }
public static void main(String[] args) {
for (Season s : Season.values())
System.out.println(s);
 }}
```

Output:

WINTER

SPRING

SUMMER

FALL

Metadata:

The metadata means data about data i.e. we can get further information from the data.

POSSIBLE QUESTIONS

2 MARKS

1. What is Inheritance?
2. Define Interfaces?
3. What is Packages?
4. Define Enumerations?
5. Write about Auto boxing and Metadata.
6. What is Method Overriding?
7. Define Dynamic Method Dispatch?
8. What is Java Enumeration?
9. Write about Abstract classes.

6 MARKS

1. Explain Multilevel Inheritance with example program.
2. Explain the following: i)Auto boxing/Unboxing ii) Wrapper Classes
3. What is Inheritance? Briefly explain importance of abstract Classes in Java.
4. Describe interfaces & how to implement it with a Java Program?
5. What is a package? What are the benefits of using package? Write down the steps in creating a package and using it in a java program with an example.
6. List and Explain Inheritance with example.
7. Explain Multilevel Inheritance with an example Program
8. How will you declare a package and import it, Explain.
9. Discuss in detail about the core interfaces and collections in JAVA utility package.
10. Write a JAVA program to create and using inheritance

KARPAGAM ACADEMY OF HIGHER EDUCATION

COIMBATORE - 21

DEPARTMENT OF COMPUTER SCIENCE,CA & IT

CLASS : I BCA

BATCH : 2018-2021

Part -A Online Examinations **(1 mark questions)**

SUBJECT: Programming in Java **SUBJECT CODE: 18CAU201**

UNIT 3

S.No	Questions	opt1	opt2	opt3	opt4
1	_____ is an explicit specification of a set of methods	Interface	package	Statement	None
2	_____ are containers for classes that are used to keep the class namespace	Interface	package	Statement	None
3	All of the Java “built-in” classes included in the java distribution are stored in a package	Header	Java	Package	Files
4	_____ are the means of encapsulation and containing the namespace and scope of	Class	Package	Classes and Package	None
5	_____ act as containers for classes and other packages.	Container	Classes	Java	Packages
6	_____ is used to extend a class by creating a new class	constructors	method overloading	inheritance	none
7	When you extends a class, you can change the behavior of a method in the parent class.	method overriding.	object refernce	method overloading	polymorphi
8	The _____ operator creates a single instances of a named class and returns a	dot	new	super	this
9	The _____ operator is used to access the instance variables and method within an	new	dot	this	super
10	Methods are called on an instance of a class using the _____ operator	new	dot	this	super
11	_____ is used inside of any method to refer to the current object.	new	dot	this	super
12	The data, or variables, defined within a class are called ____.	instance variables	reference variables	methods	classes
13	_____ initializes an object	overloading	constructors	overriding	none
14	To add a finalizer to a class, you simply define the _____ method	finalize( )	stop()	exit()	none
15	the new operator dynamically _____memory for an object.	free	allocates	delete	none
16	_____control to transfer back to the caller of the method	continue	return	jump	goto

17	a _____ statement causes control to be transferred directly to the conditional	continue	return	jump	goto
18	a method in a subclass has the same name and type signature as a method in its	override	overload	function	none
19	_____ dispatch is the mechanism by which a call to an overridden method is	Static method	Dynamic method	overload	none
20	_____ is the one, which creates more than one methods with the same name but	method overriding	method overloading	function call	inheritance
21	static methods will not refer the _____	this	dot	new	public
22	_____ is used to allocate memory in the constructor	Delete	Binding	Free	new
23	Java supports a concept called _____ which is just opposite to initialization.	free	finalization	delete	new
24	A class that cannot be subclassed is called as _____ class.	abstract	final	static	methods
25	_____ enables an object to initialize itself when it is created	Destructor	constructor	overloading	none
26	Subclass constructors can call superclass constructors via the _____ keyword	final	protected	inherit	super
27	The _____ is special because its name is the same as the class name.	Destructor	static	constructor	none of the above
28	A constructor that accepts no parameters is called the _____ constructor	Copy	default	multiple	none of the above
29	Constructors are invoked automatically when the _____ are created	Datas	classes	objects	none of the above
30	Constructors cannot be _____	Inherited	destroyed	both a & b	none of the above
31	The constructors that can take arguments are called _____ constructors	Copy	multiple	parameteriz	none of the above
32	Code Reusability is characterized by	baseclass	Subclass	Derived class	Inheritance
33	Which of these keyword must be used to inherit a class?	super	this	extent	extends
34	Which of these keywords is used to refer to member of base class from a sub class?	upper	super	this	None
35	A class member declared protected becomes member of subclass of which	public member	private member	protected member	static member
36	Which of these is correct way of inheriting class A by class B?	class B + class A {}	class B inherits	class B extends A	class B extends
37	"X extends Y" is correct if X and Y are either	both classes	both interfaces	X is class	Both A and B

38	An interface can extend many interfaces	TRUE	FALSE	Only Class can	None
39	Which of these keywords is used to define interfaces in Java?	interface	Interface	intf	Intf
40	Which of these can be used to fully abstract a class from its implementation?	Objects	Packages	Interfaces	None of the Mentioned.
41	Which of these access specifiers can be used for an interface?	Public	Protected	private	All of the mentioned
42	Which of these keywords is used by a class to use an interface defined previously?	import	Import	implements	Implements
43	Which of the following is correct way of implementing an interface salary by class	class manager extends salary	class manager	class manager	None of the mentioned.
44	Which of the following is incorrect statement about packages?	Interfaces specifies what	Interfaces are specified	interface are implicitly	All variables
45	Which of the following package stores all the standard java classes?	lang	java	util	java.packages
46	Which of these packages contain classes and interfaces used for input & output operations	java.util	ava.lang	java.io	All of the mentioned
47	Which of these class is not a member class of java.io package?	String	StringReader	Writer	File
48	Which of these interface is not a member of java.io package?	DataInput	ObjectInput	ObjectFilter	FileFilter
49	Which of these class is not related to input and output stream in terms of functioning?	File	Writer	InputStream	Reader
50	Which of these is specified by a File object?	a file in disk	directory path	directory in disk	None of the above
51	Which of these is method for testing whether the specified element is a file or a directory?	IsFile()	isFile()	Isfile()	isfile()
52	Which of these classes is used for input and output operation when working with bytes?	InputStream	Reader	Writer	All of the mentioned
53	Which of these standard collection classes implements a dynamic array?	AbstractList	LinkedList	ArrayList	AbstractSet
54	Which of these method is used to reduce the capacity of an ArrayList object?	trim()	trimSize()	trimTosize()	trimToSize()
55	Which of these standard collection classes implements a linked list data structure?	AbstractList	LinkedList	HashSet	AbstractSet
56	Which of these classes implements Set interface?	ArrayList	HashSet	LinkedList	DynamicList
57	Which of these classes is not included in java.lang?	Byte	Integer	Array	Class
58	Which of these is a process of converting a simple data type into a class?	type wrapping	type conversion	type casting	None of the Mentioned

59	Which of these is a super class of wrappers Double & Integer?	Long	Digits	Float	Number
60	Which of these is wrapper for simple data type float?	float	double	Float	Double

answer
Interface
Package
Package
Class and Package
Packages
inheritance
method overriding.
new
dot
dot
this
instance variables
constructors
finalize( )
allocates
return

continue
override
Dynamic method
method overloading
this
new
finalization
final
constructor
super
constructor
default
objects
Inherited
parameterize
Inheritance
extends
super
private member
class B extends A
Both A and B

TRUE
interface
Interfaces
Public
implements
class manager
All variables are static
java
java.io
String
ObjectFilter
File
directory in disk
isFile()
InputStream
ArrayList
trimToSize()
LinkedList
HashSet
Array
type conversion

Number
Float

UNIT-IV

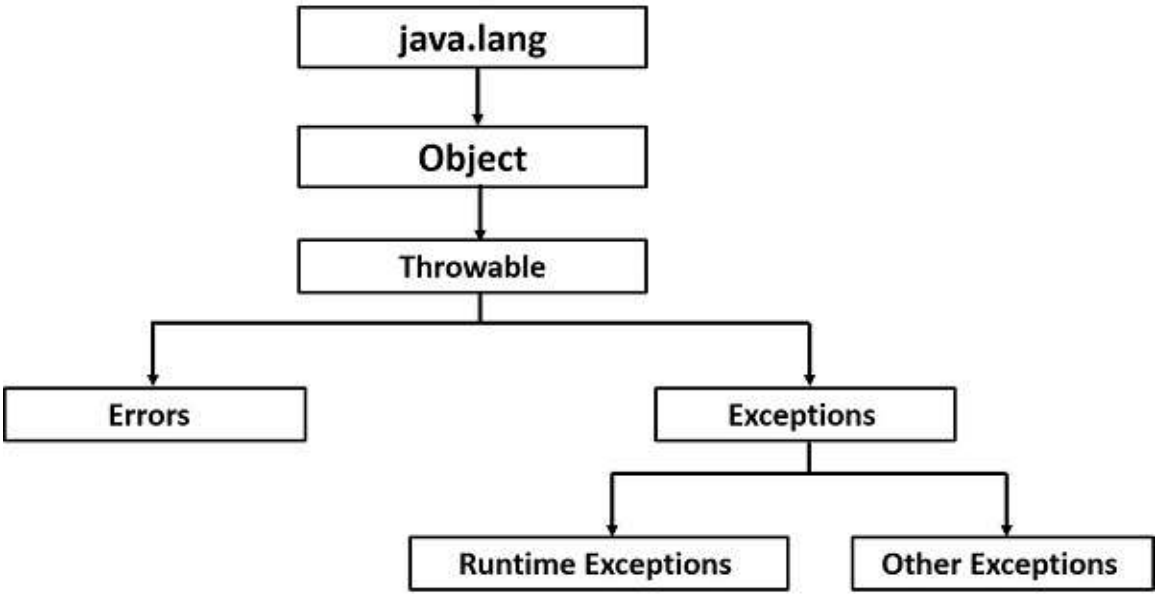
SYLLABUS

Exception Handling, Threading, Networking and Database Connectivity Exception types, uncaught exceptions, throw, built-in exceptions, Creating your own exceptions; Multi-threading: The Thread class and Runnable interface, creating single and multiple threads, Thread prioritization, synchronization and communication, suspending/resuming threads. Using java.net package, Overview of TCP/IP and Datagram programming. Accessing and manipulating databases using JDBC.

Exception Handling

Exception Handling in Java

Java - Exceptions. An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.



The **exception handling in java** is one of the powerful*mechanisms to handle the runtime errors* so that normal flow of the application can be maintained.

What is exception?

Dictionary Meaning: Exception is an abnormal condition.

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is exception handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

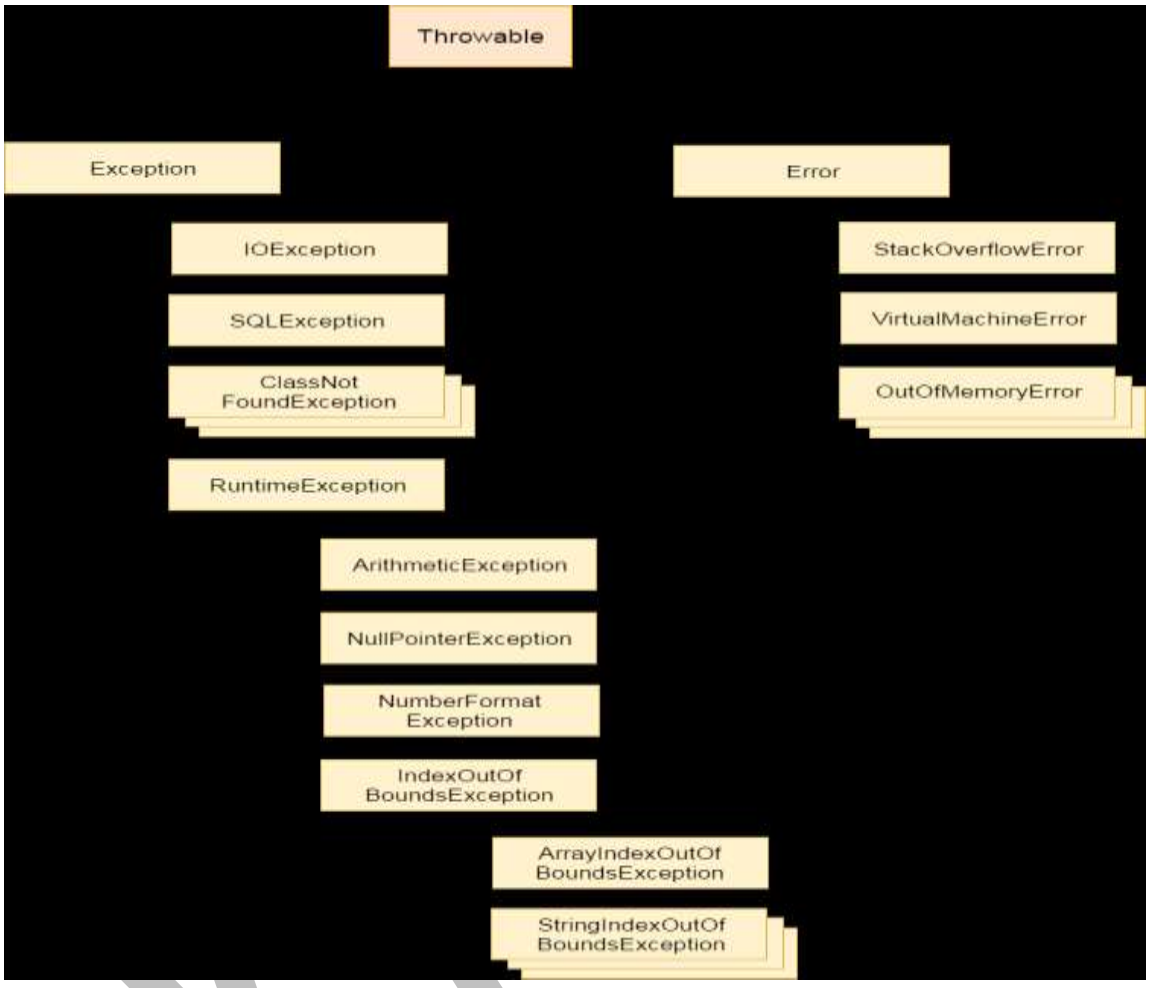
Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Suppose there is 10 statements in your program and there occurs an exception at statement5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

Hierarchy of Java Exception classes



Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between checked and unchecked exceptions

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Common scenarios where exceptions may occur

There are given some scenarios where unchecked exceptions can occur. They are as follows:

1) Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

2) Scenario where NullPointerException occurs

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
String s=null;

System.out.println(s.length());//NullPointerException
```

3) Scenario where NumberFormatException occurs

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

```
String s="abc";

int i=Integer.parseInt(s);//NumberFormatException
```

4) Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

1. **int** a[]=**new int**[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException

Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws

Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Syntax of java try-catch

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

Syntax of try-finally block

```
try{
//code that may throw exception
}finally{}
```

Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

Problem without exception handling

Let's try to understand the problem if we don't use try-catch block.

```
public class Testtrycatch1{
  public static void main(String args[]){
     int data=50/0;//may throw exception
     System.out.println("rest of the code..."); } }
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
```

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed).

There can be 100 lines of code after exception. So all the code after exception will not be executed.

Solution by exception handling

Let's see the solution of above problem by java try-catch block.

```
public class Testtrycatch2{
  public static void main(String args[]){
   try{
     int data=50/0;
   }catch(ArithmeticException e){System.out.println(e);}
   System.out.println("rest of the code..."); } }
```

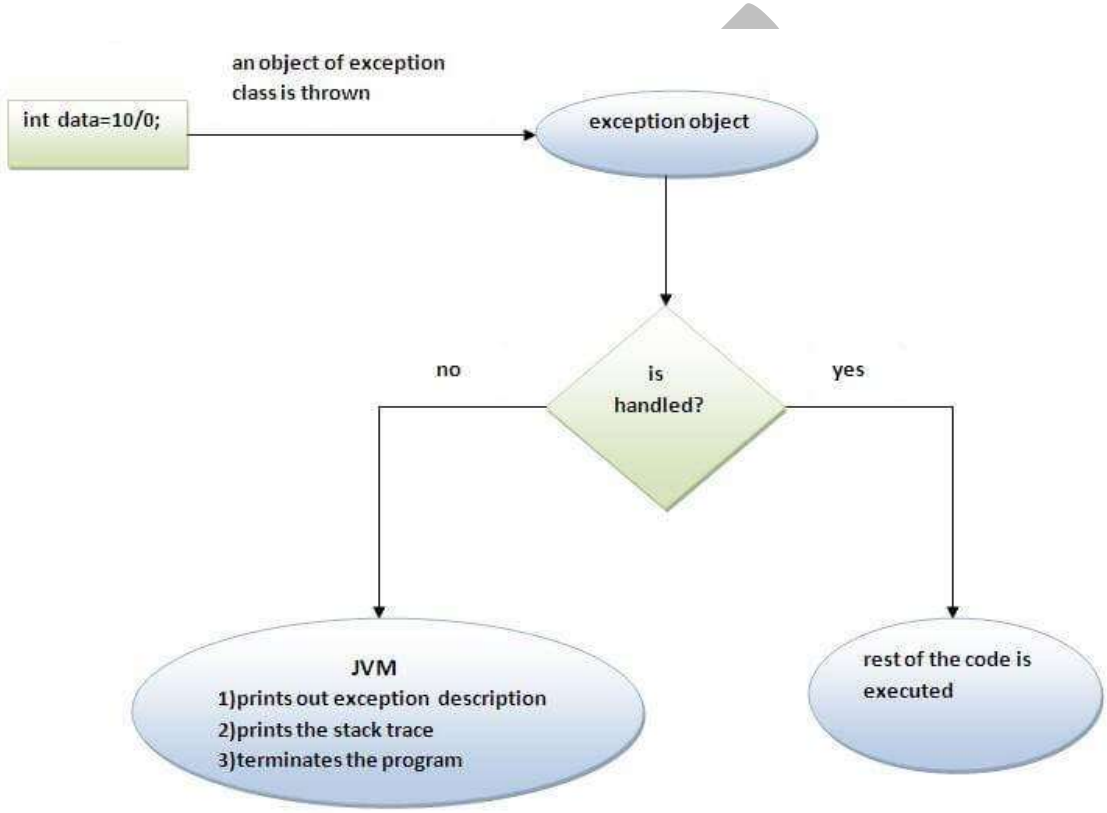
Output:

Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code... statement is printed.

Internal working of java try-catch block



The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

- ✓ Prints out exception description.
- ✓ Prints the stack trace (Hierarchy of methods where the exception occurred).
- ✓ Causes the program to terminate.

But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

throw exception;

Let's see the example of throw IOException.

throw new IOException("sorry device error);

Java throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
public class TestThrow1{
  static void validate(int age){
    if(age<18)
     throw new ArithmeticException("not valid");
    else
     System.out.println("welcome to vote");     }
  public static void main(String args[]){
     validate(13);
     System.out.println("rest of the code...");    }  }
```

Output:

Exception in thread main java.lang.ArithmeticException:not valid

Java throws keyword

The Java throws keyword is used to declare an exception. It gives information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws

```
return_type method_name() throws exception_class_name{

//method code

}
```

Which exception should be declared

Ans) checked exception only, because:

- **unchecked Exception:** under your control so correct your code.
- **error:** beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

Advantage of Java throws keyword

Now Checked Exception can be propagated (forwarded in call stack).

It provides information to the caller of the method about the exception.

Java throws example

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
import java.io.IOException;

class Testthrows1{
```

```
void m()throws IOException{
  throw new IOException("device error");//checked exception    }
void n()throws IOException{
  m();    }
void p(){
 try{
  n();     }catch(Exception e){System.out.println("exception handled");}    }
public static void main(String args[]){
  Testthrows1 obj=new Testthrows1();
  obj.p();
  System.out.println("normal flow...");    }  }
```

Output:

exception handled

normal flow...

Rule: If you are calling a method that declares an exception, you must either caught or declare the exception.

There are two cases:

Case1: You caught the exception i.e. handle the exception using try/catch.

Case2: You declare the exception i.e. specifying throws with the method.

Case1: You handle the exception

In case you handle the exception, the code will be executed fine whether exception occurs during the program or not.

```
import java.io.*;

class M{

 void method()throws IOException{

  throw new IOException("device error");   }  }

public class Testthrows2{

   public static void main(String args[]){

    try{

     M m=new M();

     m.method();

    }catch(Exception e){System.out.println("exception handled");}

      System.out.println("normal flow...");    }  }
```

Output:

```
exception handled

normal flow...
```

Case2: You declare the exception

A) In case you declare the exception, if exception does not occur, the code will be executed fine.

B) In case you declare the exception if exception occurs, an exception will be thrown at runtime because throws does not handle the exception.

A) Program if exception does not occur

```
import java.io.*;
class M{
 void method()throws IOException{
  System.out.println("device operation performed");   }  }
class Testthrows3{
   public static void main(String args[])throws IOException{//declare exception
     M m=new M();
     m.method();
     System.out.println("normal flow...");    }  }
```

Output: device operation performed

normal flow...

B) Program if exception occurs

```
import java.io.*;
class M{
 void method()throws IOException{
  throw new IOException("device error");   }  }
class Testthrows4{
   public static void main(String args[])throws IOException{//declare exception
     M m=new M();
```

```
m.method();
 System.out.println("normal flow...");    }  }
```

Output: Runtime Exception

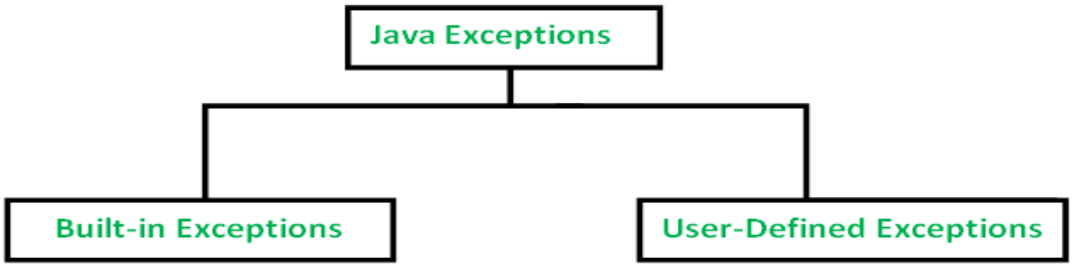
Difference between throw and throws in Java

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

Types of Exception in Java with Examples

Java defines several types of exceptions that relate to its various class libraries. Java also allows users to define their own exceptions.



Built-in Exceptions

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

1.Arithmetic Exception
It is thrown when an exceptional condition has occurred in an arithmetic operation.
2.ArrayIndexOutOfBoundException
It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
3.ClassNotFoundException
This Exception is raised when we try to access a class whose definition is not found
4.FileNotFoundException
This Exception is raised when a file is not accessible or does not open.
5.IOException
It is thrown when an input-output operation failed or interrupted
6.InterruptedException
It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.
7.NoSuchFieldException
It is thrown when a class does not contain the field (or variable) specified
8.NoSuchMethodException
It is thrown when accessing a method which is not found.
9.NullPointerException
This exception is raised when referring to the members of a null object. Null represents nothing
10. NumberFormatException
This exception is raised when a method could not convert a string into a numeric format.
11. RuntimeException
This represents any exception which occurs during runtime.

12. StringIndexOutOfBoundsException

It is thrown by String class methods to indicate that an index is either negative than the size of the string

Examples of Built-in Exception:

Arithmetic exception

```
// Java program to demonstrate ArithmeticException

classArithmeticException_Demo{

   publicstaticvoidmain(String args[])    {

      try{

         inta = 30, b = 0;

         intc = a/b;  // cannot divide by zero

         System.out.println ("Result = "+ c);       }

      catch(ArithmeticException e) {

         System.out.println ("Can't divide a number by 0");       }    }}
```

Output:

Can't divide a number by 0

NullPointer Exception

```
//Java program to demonstrate NullPointerException

classNullPointer_Demo{

   publicstaticvoidmain(String args[])

   {

      try{

         String a = null; //null value

         System.out.println(a.charAt(0));

      } catch(NullPointerException e) {

         System.out.println("NullPointerException..");       }    }}
```

Output:

NullPointerException..

StringIndexOutOfBound Exception

```
// Java program to demonstrate StringIndexOutOfBoundsException

classStringIndexOutOfBound_Demo{

   publicstaticvoidmain(String args[])

   {

      try{

         String a = "This is like chipping "; // length is 22

         charc = a.charAt(24); // accessing 25th element

         System.out.println(c);       }

      catch(StringIndexOutOfBoundsException e) {

         System.out.println("StringIndexOutOfBoundsException");       }    }}
```

Output:

StringIndexOutOfBoundsException

FileNotFound Exception

```
//Java program to demonstrate FileNotFoundException

importjava.io.File;

importjava.io.FileNotFoundException;

importjava.io.FileReader;

 classFile_notFound_Demo {

    publicstaticvoidmain(String args[])  {

      try{

          // Following file does not exist

         File file = newFile("E://file.txt");

          FileReader fr = newFileReader(file);
```

```
    } catch(FileNotFoundException e) {
      System.out.println("File does not exist");      }   }}
```

Output:

File does not exist

NumberFormat Exception

```
// Java program to demonstrate NumberFormatException
class  NumberFormat_Demo
{
   publicstaticvoidmain(String args[])   {
      try{
         // "akki" is not a number
         intnum = Integer.parseInt ("akki") ;
          System.out.println(num);
      } catch(NumberFormatException e) {
         System.out.println("Number format exception");      }   }}
```

Output:

Number format exception

ArrayIndexOutOfBounds Exception

```
// Java program to demonstrate ArrayIndexOutOfBoundException
classArrayIndexOutOfBound_Demo
{
   publicstaticvoidmain(String args[])   {
      try{
         inta[] = newint[5];
         a[6] = 9; // accessing 7th element in an array of
               // size 5       }
```

```
catch(ArrayIndexOutOfBoundsException e){
    System.out.println ("Array Index is Out Of Bounds");      }   }}
```

Output:
Array Index is Out Of Bounds

Java Custom Exception

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

Let's see a simple example of java custom exception.

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);   }  }
class TestCustomException1{
   static void validate(int age)throws InvalidAgeException{
     if(age<18)
      throw new InvalidAgeException("not valid");
     else
      System.out.println("welcome to vote");     }
       public static void main(String args[]){
      try{
      validate(13);
      }catch(Exception m){System.out.println("Exception occured: "+m);}
       System.out.println("rest of the code...");    }  }
```

Output:

Exception occured: InvalidAgeException:not valid

rest of the code...

Multithreading

Itis a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.

2) You **can perform many operations together so it saves time**.

3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

How to create thread?

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()

- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).

18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

Starting a thread:

start() method of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

1) Java Thread Example by extending Thread class

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");  }
public static void main(String args[]){
Multi t1=new Multi();
t1.start();   }  }
```

Output:

thread is running...

2) Java Thread Example by implementing Runnable interface

```
class Multi3 implements Runnable{

public void run(){

System.out.println("thread is running...");  }

public static void main(String args[]){

Multi3 m1=new Multi3();

Thread t1 =new Thread(m1);

t1.start();   }  }
```

Output:

thread is running...

If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitly create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.

Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

```
public static int MIN_PRIORITY

public static int NORM_PRIORITY

public static int MAX_PRIORITY
```

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread{
```

```
public void run(){
  System.out.println("running thread name is:"+Thread.currentThread().getName());
  System.out.println("running thread priority is:"+Thread.currentThread().getPriority());    }
public static void main(String args[]){
 TestMultiPriority1 m1=new TestMultiPriority1();
 TestMultiPriority1 m2=new TestMultiPriority1();
 m1.setPriority(Thread.MIN_PRIORITY);
 m2.setPriority(Thread.MAX_PRIORITY);
 m1.start();
 m2.start();   }  }
```

Output:

```
running thread name is:Thread-0
    running thread priority is:10
    running thread name is:Thread-1
    running thread priority is:1
```

Synchronization

Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization?

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.
    3. static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block
3. by static synchronization

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

From Java 5 the package java.util.concurrent.locks contains several lock implementations.

Understanding the problem without Synchronization

In this example, there is no synchronization, so output is inconsistent. Let's see the example:

```
class Table{
void printTable(int n){//method not synchronized
  for(int i=1;i<=5;i++){
    System.out.println(n*i);
    try{
     Thread.sleep(400);
    }catch(Exception e){System.out.println(e);}    }    }  }
 class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;  }
public void run(){
t.printTable(5);  }  }
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;  }
public void run(){
t.printTable(100);        }  }
class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
```

```
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();  }  }
```

Output:

```
5
 100
 10
 200
 15
 300
 20
 400
 25
 500
```

Java synchronized method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
//example of java synchronized method
class Table{
synchronized void printTable(int n){//synchronized method
for(int i=1;i<=5;i++){
```

```
System.out.println(n*i);
try{
Thread.sleep(400);
}catch(Exception e){System.out.println(e);} } } }
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t; }
public void run(){
t.printTable(5); } }
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t; }
public void run(){
t.printTable(100); } }
public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start(); } }
```

KAHE

Output:

5

10

15

20

25

100

200

300

400

500

Inter-thread communication in Java

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method

Description

public final void wait()throws InterruptedException

waits until object is notified.

public final void wait(long timeout)throws InterruptedException

waits for the specified amount of time.

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

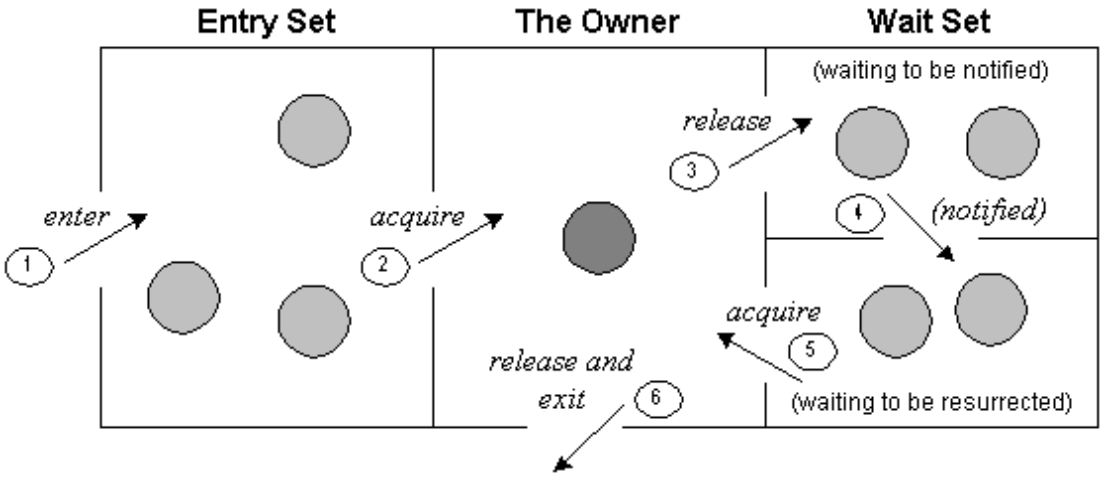
public final void notify()

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()

Understanding the process of inter-thread communication



The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.

2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

Why wait(), notify() and notifyAll() methods are defined in Object class not Thread class?

It is because they are related to lock and object has a lock.

Difference between wait and sleep?

Let's see the important differences between wait and sleep methods.

wait()	sleep()
wait() method releases the lock	sleep() method doesn't release the lock.
is the method of Object class	is the method of Thread class
is the non-static method	is the static method
is the non-static method	is the static method
should be notified by notify() or notifyAll() methods	after the specified amount of time, sleep is completed.

Example of inter thread communication in java

```
class Customer{
int amount=10000;
synchronized void withdraw(int amount){
System.out.println("going to withdraw...");
 if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");  }
 synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();  }  }
class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
```

```
public void run(){c.deposit(10000);}

}.start();  }}
```

Output:

```
going to withdraw...

    Less balance; waiting for deposit...

    going to deposit...

    deposit completed...

    withdraw completed
```

Suspending/resuming threads:

Suspending

The suspend( ) method of theThread class was deprecated by Java 2 several years ago. This was done because suspend( ) can sometimes cause serious system failures. Assume that a thread has obtained locks on critical data structures. If that thread is suspended atthat point, those locks are not relinquished. Other threads that may be waiting for those resources can be deadlocked.

Resuming

The resume( ) method is also deprecated. It does not cause problems, but cannot be used without the suspend( ) method as its counterpart.

Stopping

The stop( ) method of theThread class, too, was deprecated by Java 2. This was done because this method can sometimes cause serious system failures. Assume that a thread is writing to a critically important data structure and has completed only part of its changes.If that thread is stopped at that point, that data structure might be left in a corrupted state.

Because you can't now use the suspend( ),resume( ), or stop( ) methods to control a thread, you might be thinking that no way exists to pause, restart, or terminate a thread.But, fortunately, this is not true. Instead, a thread must be designed so that the run( ) method periodically checks to determine whether that thread should suspend, resume, or stop its own execution. Typically, this is accomplished by establishing a flag variable that indicates the execution state of the thread. As

long as this flag is set to "running," the run( ) method must continue to let the thread execute. If this variable is set to "suspend," the thread must pause. If it is set to "stop," the thread must terminate.

Example:Suspending, resuming, and stopping a thread.

```
// Suspending, resuming, and stopping a thread.
 using System;
using System.Threading;
class MyThread
{
public Thread thrd;
 public MyThread(string name)
 {
thrd = new Thread(new ThreadStart(this.run));
thrd.Name = name;
thrd.Start();
}
// This is the entry point for thread.
 void run() {
Console.WriteLine(thrd.Name + " starting.");
for(int i = 1; i <= 1000; i++){
Console.Write(i + " ");
if((i%10)==0) {
Console.WriteLine();
Thread.Sleep(250);  } }
```

```
Console.WriteLine(thrd.Name + " exiting."); } }
public class SuspendResumeStop {
public static void Main() {
MyThread mt1 = new MyThread("My Thread");
Thread.Sleep(1000); // let child thread start executing
mt1.thrd.Suspend();
Console.WriteLine("Suspending thread.");
Thread.Sleep(1000);
mt1.thrd.Resume();
Console.WriteLine("Resuming thread.");
Thread.Sleep(1000);
mt1.thrd.Suspend();
Console.WriteLine("Suspending thread.");
Thread.Sleep(1000);
mt1.thrd.Resume();
Console.WriteLine("Resuming thread.");
Thread.Sleep(1000);
Console.WriteLine("Stopping thread.");
mt1.thrd.Abort();
mt1.thrd.Join(); // wait for thread to terminate
Console.WriteLine("Main thread terminating."); } }
```

When you run this program,you will see the output shown here:

```
My Thread starting.
```

1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40

Supending thread.
Resuming Thread.

41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 64 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80

The java.net Package

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. sharing resources
2. centralize software management

KAHE

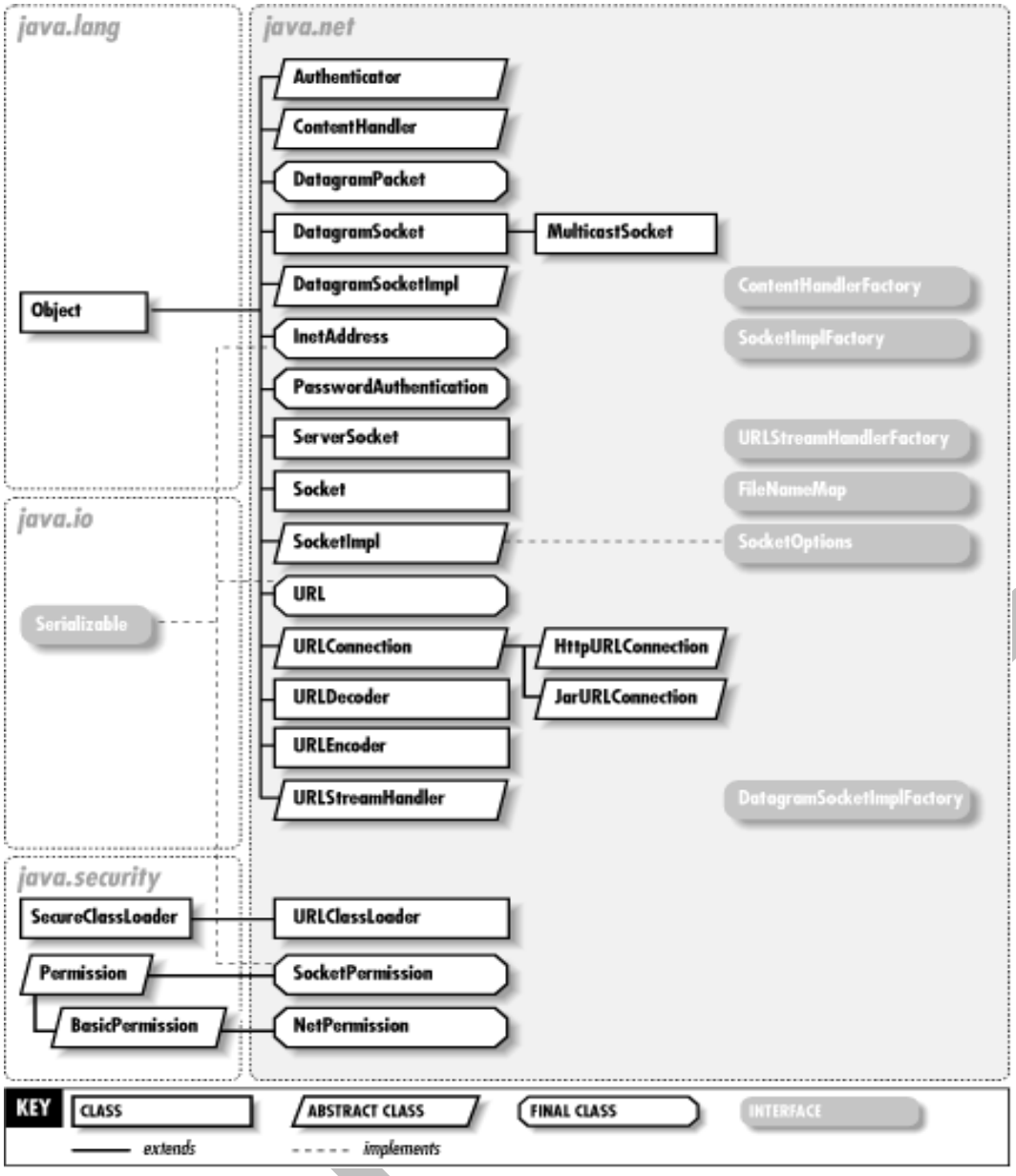


Figure 1. The classes of the java.net package

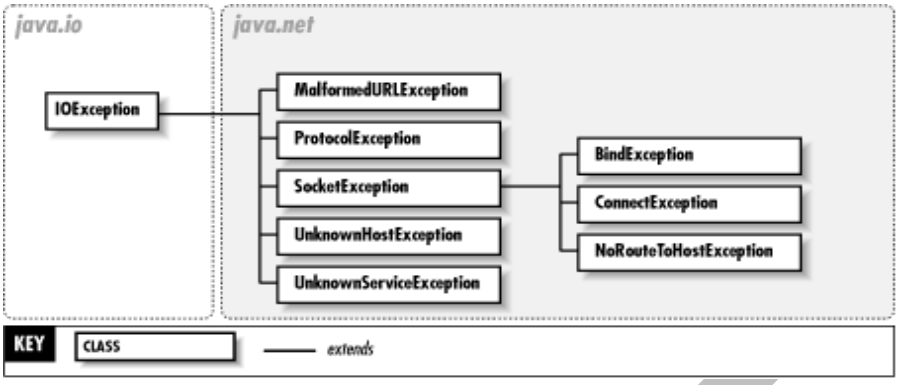


Figure 2. The exceptions of the java.net package

Java Networking Terminology

The widely used java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet

- SMTP
- POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communications.

Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.

Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.
3.

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Example of Java Socket Programming

File: MyServer.java

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}  }  }
```

File: MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
```

s.close(); }**catch**(Exception e){System.out.println(e);} } }

Java URL

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:

http://www.javatpoint.com/java-tutorial

A URL contains much information:

1. **Protocol:** In this case, http is the protocol.
2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.jsp is the file name.

Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated with this

	URL.

Example of Java URL class

```
//URLDemo.java
import java.io.*;
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
 System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("File Name: "+url.getFile());
 }catch(Exception e){System.out.println(e);}  }  }
```

Output:

```
Protocol: http
Host Name: www.javatpoint.com
Port Number: -1
File Name: /java-tutorial
```

Java URLConnection class

The **Java URLConnection** class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

How to get the object of URLConnection class

The openConnection() method of URL class returns the object of URLConnection class.
Syntax:

public URLConnection openConnection()**throws** IOException{}

Displaying source code of a webpage by URLConnecton class

The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method. The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

Example of Java URLConnection class

```
import java.io.*;

import java.net.*;

public class URLConnectionExample {

public static void main(String[] args){

try{

URL url=new URL("http://www.javatpoint.com/java-tutorial");

URLConnection urlcon=url.openConnection();

InputStream stream=urlcon.getInputStream();

int i;

while((i=stream.read())!=-1){

System.out.print((char)i);  }

}catch(Exception e){System.out.println(e);}  }}
```

Java HttpURLConnection class

The **Java HttpURLConnection** class is http specific URLConnection. It works for HTTP protocol only.

By the help of HttpURLConnection class, you can information of any HTTP URL such as header information, status code, response code etc.

The java.net.HttpURLConnection is subclass of URLConnection class.

How to get the object of HttpURLConnection class

The openConnection() method of URL class returns the object of URLConnection class. Syntax:

public URLConnection openConnection()**throws** IOException{}

You can typecast it to HttpURLConnection type as given below.

```
URL url=new URL("http://www.javatpoint.com/java-tutorial");

HttpURLConnection huc=(HttpURLConnection)url.openConnection();
```

Java InetAddress class

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.javatpoint.com, www.google.com, www.facebook.com etc.

Commonly used methods of InetAddress class

Method	Description
public static InetAddress getByName(String host) throws UnknownHostException	it returns the instance of InetAddress containing LocalHost IP and name.
public static InetAddress getLocalHost() throws UnknownHostException	it returns the instance of InetAdddress containing local host name and address.
public String getHostName()	it returns the host name of the IP address.

public String getHostAddress()	it returns the IP address in string format.

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

- **DatagramSocket() throws SocketEeption:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket(int port, InetAddress address) throws SocketEeption:** it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket class

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

UNIT -IV

POSSIBLE QUESTIONS

2 MARKS

1. What is an Exception?
2. Distinguish between init () and start () methods.
3. What is the difference between exception and error?
4. Differentiate wait and sleep methods in java?
5. Define Exception Handling.
6. What is Thread prioritization?
7. Difference between throw and throws in Java
8. Types of Exception in Java with Examples
9. Mention the Java Exception Handling Keywords.
10. What is Synchronization?
11. What are the advantages of Java Networking?

6 MARKS

1. Explain the use of thread methods yield(), stop() and sleep().
2. Discuss built-in exceptions with suitable example program.
3. What is an Exception? Explain how to throw, catch and handle Exceptions
4. Explain creating a thread, extending the thread class and an example of using the thread class.
5. Explain following keywords used in Exception Handling. (i) try (ii) catch(iii) throw
6. Discuss in detail about java.net package with example

7. What is a package? What are the benefits of using package? Write down the steps in creating a package and using it in a java program with an example.

8. List and Explain Inheritance with example.

9. Clarify in detail about the basics of exception with its types and an example program

10. Define threads. Explain multiplication table using multithreading with suitable program.

11. Clarify in detail about the types of exception with an example program to handle array out of bounds exception.

12. Describe creating multiple Threads with example program.

13. Explain Inter-thread communication in Java

KAHE

KARPAGAM ACADEMY OF HIGHER EDUCATION

COIMBATORE - 21

DEPARTMENT OF COMPUTER SCIENCE,CA & IT

CLASS : I BCA

BATCH : 2018-2021

Part -A Online Examinations (1 mark questions)

SUBJECT: Programming in Java SUBJECT CODE: 18CAU201

Unit-4

S.No	Questions	opt1	opt2	opt3	opt4	answer
1	An _____ is a condition that is caused by a runtime error in the	throw	exception	handle	catch	exception
2	Exception can be generated by the _____ or manually by the code	Throwable class	Java runtime	object	catch	Java runtime
3	All exception types are subclasses of the built_in class _____	Throwable	RuntimeException	StackTree	LocalizedMessage	Throwable
4	All exception classes are divided into _____ groups	3	4	2	6	2
5	The _____ defines the exceptions which are not expected to be caught	java.lang.Error	java.lang.Math	java.lang.Throwable	java.lang.IOException	java.lang.Error
6	When an exception occurs within a java method, the method creates an	catching the	throwing an exception	handle the exception	get the exception	throwing an exception
7	When java method throws an exception the java runtime system searches all	catching the	throwing an exception	handle the exception	get the exception	catching the
8	Exception performs _____ tasks	3	4	5	2	4
9	Unchecked exceptions are extensions of _____	throws	catch	RuntimeException	Error	RuntimeException
10	Checked exceptions are extensions of _____	throws	catch	Exception	Error	Exception
11	Each of Exception's predefined class provide _____ constructors	3	4	5	2	2
12	The errors are printed by _____	Stack Trace	StackTree	Message	Error	Stack Trace
13	AWT includes a very simple plain text,multiline editor called	Label	TextField	TextArea	Option.	TextArea
14	_____ class is a button that is used to toggle the state of a check	Label	Option	CheckBox	Button	CheckBox
15	_____ class is at the top of the exception class hierarchy.	Exception	Error	Throws	Throwable	Throwable
16	_____subclass of throwable defines exceptions that are not	Exception	Error	Throws	Throwable	Error

17	The _____ class is used for exceptional conditions that the user	Exception	Error	Throws	Throwable	Exception
18	The two subclass of throwable class are _____	Exception and Error	Exception and handler	throw and throwable	try and catch	Exception and Error
19	The _____ Keyword is used to specify a block of code that should	Catch	try	exception	block of code	try
20	_____ specifies the type of exception to be caught.	Catch	try	exception	block of code	Catch
21	_____ keyword is used to identify the list of possible exceptions	throw	try	catch	none	throw
22	Certain block of code necessarily has to be run no matter of what exceptions	throw	final	finally	none	finally
23	There are _____ ways of creating Throwable object	3	4	5	2	2
24	_____ is an important subclass of exception	RuntimeException	Aarithmetic Exception	NullException	Subclasses of	RuntimeException
25	What is the name of the method used to start a thread execution?	init();	start();	run();	resume();	start();
26	Which two are valid constructors for Thread?	Thread(Runnable r,	Thread()	Thread(int priority)	Both A and B	Both A and B
27	Which three are methods of the Object class?	notify();	notifyAll();	wait(long msecs);	All the Above	All the Above
28	Which cannot directly cause a thread to stop executing?	SetPriority() method	wait() method	notify() method	read() method	notify() method
29	Which two of the following methods are defined in class Thread?	start()	run()	wait()	Both A and B	Both A and B
30	Which guarantee that a thread will leave the running state?	wait()	sleep(1000)	aLiveThread.join()	All the Above	All the Above
31	Which of the following will directly stop the execution of a Thread?	wait()	notify()	notifyall()	exits synchronize	wait()
32	Which method must be defined by a class implementing the	void run()	public void run()	public void start()	void run(int priority)	public void run()
33	Which will contain the body of the thread?	run();	start();	stop();	main();	run();
34	Which method registers a thread in a thread scheduler?	run();	construct();	start();	register();	start();
35	called from a thread A on an object B: wait(2000);	after two seconds	after thread *A* is	after lock *B* is	None of thes	after two seconds
36	Which class or interface defines the wait(), notify(),and notifyAll()	Object	Thread	Runnable	Class	Object
37	public class MyRunnable implements Runnable -which of these will create	new Runnable(	new Thread(My	new Thread(ne	new MyRunnabl	new Thread(ne

38	What is true about threads?	Threads consumes	enables multi	reduces idle time	All	All
39	A thread can acquire a lock by using which reserved keyword?	volatile	synchronized	locked	none	synchronized
40	How many threads can a process contain?	1	2	multiple	none	multiple
41	What is sometimes also called a lightweight process?	Thread	Process	JVM	All	Thread
42	What is name of thread which calls main method	mainThread	Thread	Thread-0	main	main
43	Significance of synchronized variable	doesn't exist	used in multi	Prevents concurrent	None	doesn't exist
44	What is pre-emptive scheduling in threads	highest priority	low priority thread	medium priority	Anyone may happen	highest priority
45	What will happen if two threads try to read same resource without	not allowed in	doesn't create any	create race condition	None	doesn't create any
46	What are valid statements for daemon threads?	created as daemon	only daemon	low priority	All of these	All of these
47	Which tools could be used to analyse thread dumps	VisualVM	jstack	All	none	All
48	Which method can make Thread to go from running to waiting state	wait()	resume()	notify()	alive()	wait()
49	The JDBC-ODBC bridge is	Multithreaded	Singlethreaded	Both of the above	none of the above	Multithreaded
50	Which statements about JDBC are true?	JDBC is an API	Java DataBase	access relational	XML data sources	Java DataBase
51	Which type of driver provides JDBC access via one or more ODBC drivers?	Type 1 driver	Type 2 driver	Type 3 driver	Type 4 driver	Type 1 driver
52	JDBC stands for:	Java Database	Java Database	Java Database	None of the above	Java Database
53	Where is metadata stored in MySQL?	MySQL database	MySQL database	MySQL database	None of the above	MySQL database
54	Which of the following methods are needed for loading a database driver in	registerDriver()	Class.forName()	Both A and B	getConnecti	Both A and B
55	Which of the following statements is false as far as different type of	Regular Statement	Prepared Statement	Callable Statement	Interim Statement	Interim Statement
56	Which of the following allows non repeatable read in JDBC Connection?	TRANSACTION_R	TRANSACTION_REA	TRANSACTION_S	TRANSACTION_REP	TRANSACTION_REP
57	The JDBC-ODBC Bridge supports _____per connection?	multiple concurrent	Single concurrent	multiple concurrent	Single concurrent	multiple concurrent
58	Which type of Statement can execute parameterized queries?	PreparedStatement	ParameterizedStatement	CallableStatement	All kinds of Statements	PreparedStatement

59	Which type of Statement can execute parameterized queries?	PreparedStatement	ParameterizedStatement	CallableStatement	All kinds of Statements	PreparedStatement
60	How can you execute a stored procedure in the database?	execute() on a	executeProcedure() on a	execute() on a	run() on a ProcedureC	execute() on a

UNIT-V

SYLLABUS

Java Applets: Introduction to Applets, Writing Java Applets, Working with Graphics, Incorporating Images & Sounds. Event Handling Mechanisms, Listener Interfaces, Adapter and Inner Classes. The design and Implementation of GUIs using the AWT controls, Swing components of Java Foundation Classes such as labels, buttons, text fields, layout managers, menus, events and listeners; Graphic objects for drawing figures such as lines, rectangles, ovals, using different fonts. Overview of servlets.

INTRODUCTION TO APPLETS

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

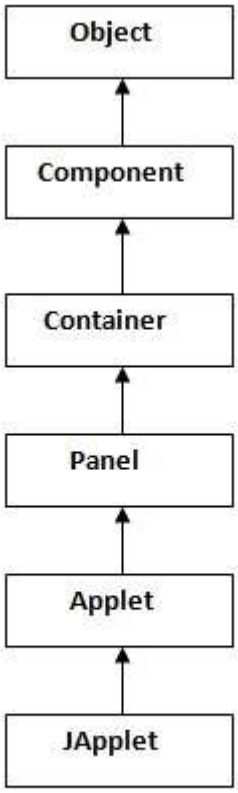
There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

Java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java

import java.applet.Applet;

import java.awt.Graphics;

public class First extends Applet

{

public void paint(Graphics g){

g.drawString("welcome",150,150);

}

 }
```

Note: class must be public because its object is created by Java Plugin software that resides on the browser.

myapplet.html

```
<html>

<body>

<applet code="First.class" width="300" height="300">

</applet>

</body>

</html>
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g){
g.drawString("welcome to applet",150,150);  }    }
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

c:\>javac First.java

c:\>appletviewer First.java

Java AWT

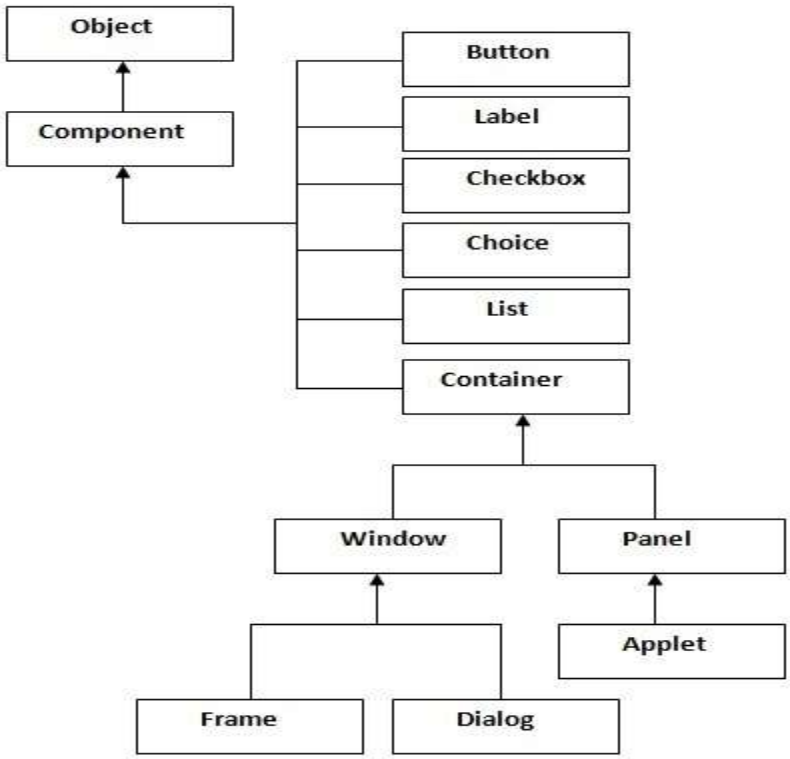
Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchies of Java AWT classes are given below.



Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The class that extends Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the containers that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class

Method	Description
public void add(Component c)	Inserts a component on this component.
public void setSize(int width,int height)	Sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	Defines the layout manager for the component.
public void setVisible(boolean status)	Changes the visibility of the component, by default false.

Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

AWT Example by Inheritance

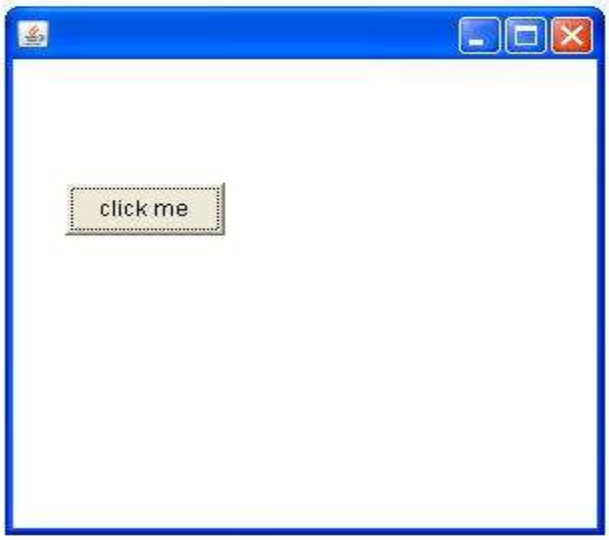
Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
```

```
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}
```

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.

KAHE



AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;

class First2{

First2(){

Frame f=new Frame();

Button b=new Button("click me");

b.setBounds(30,50,80,30);

f.add(b);

f.setSize(300,300);

f.setLayout(null);
```

```
f.setVisible(true);  }
public static void main(String args[]){
First2 f=new First2();
}}
```

Working with Graphics

The AWT supports a rich assortment of graphics method.All graphics are drawn reative to window. This can be a main window of an applet,a child window of an applet,or stand alone application of window.The origin of each window is at the top-left corner and is 0,0. Coordinates are specified in pixels. All output to a window takes place through a graphics context.

A graphics context is encapsulated by the Graphics class and is obtained in two ways:

• It is passed to a method, such as paint( ) or update( ), as an argument.

• It is returned by the getGraphics( ) method of Component.

The Graphics class defines a number of drawing functions.Each shape can be drawn edge only or filled.Several drawing methods are:

1. Drawing Lines

Lines are drawn by means of the drawLine() method,shown here:

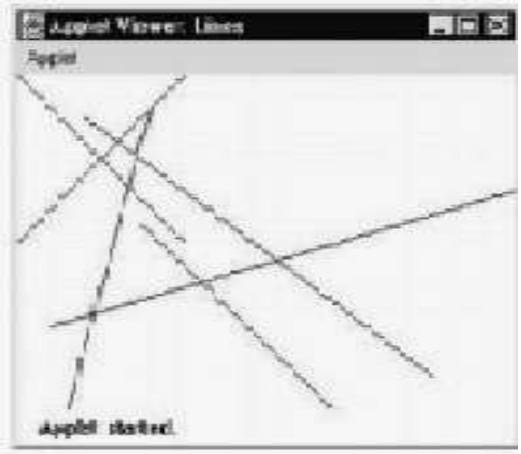
void drawLine(int startX,int startY,int endX,int endY)

Simple applet program to drawn a line

```
import java.applet.*;
import java.awt.*;
public class DrawingLines extends Applet
 {
  int width, height;
  public void init()
 {
    width = getSize().width;
    height = getSize().height;
    setBackground( Color.black );
  }
  public void paint( Graphics g )
{
    g.setColor( Color.green );
    for ( int i = 0; i < 10; ++i )
{
       g.drawLine( width, height, i * width / 10, 0 );     }  }}
```

KAHE

Sample output from this program is shown here:



2. Drawing Rectangles

The drawRect() and fillRect() methods display an outlined and filled rectangle, respectively.

Syntax:

```
void drawRect(int top,int left,int width,int height)

void fillRect(int top,int left,int width,int height)
```

The upper left corner of the rectangle is at top,left.The dimensions of the rectangle are specified by the width,height.To draw rounded rectangle,use drawRoundRect()or fillRoundRect().

Syntax:

```
void drawRoundRect(int top,int left,int width,int height,int xDiam,int yDiam)
void fillRoundRect(int top,int left,int width,int height,int xDiam,int yDiam)
```

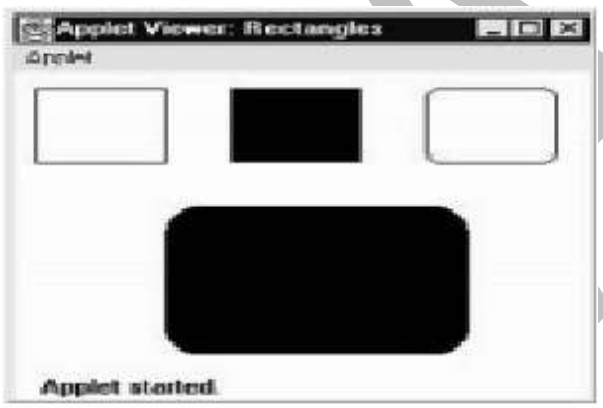
A rounded rectangle has rounded corners. The upper-left corner of the rectangle is at top,left. The dimensions of the rectangle are specified by width and height.The diameter of the rounding arc along the X axis is specified by Diam. The diameter of the rounding arc along the Y axis is specified by Diam.

applet program to draws several rectangles:

```
// Draw rectangles
```

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Rectangles" width=300 height=200>
</applet>
*/
public class Rectangles extends Applet
{
public void paint(Graphics g)
 {
g.drawRect(10, 10, 60, 50);
g.fillRect(100, 10, 60, 50);
g.drawRoundRect(190, 10, 60, 50, 15, 15);
g.fillRoundRect(70, 90, 140, 100, 30, 40);
}
}
```

Sample output from this program is shown here:



3. Drawing Ellipses and Circles

To draw an ellipse, use drawOval( ). To fill an ellipse, use fillOval( ).

Syntax:

void drawOval(int top, int left, int width, int height)

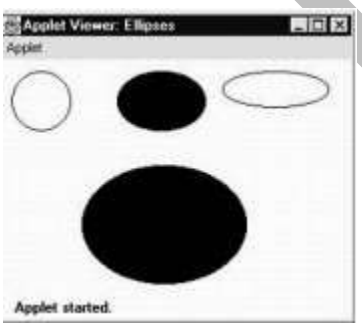
void fillOval(int top, int left, int width, int height)

The ellipse is drawn within a bounding rectangle whose upper-left corner is specified by top,left and whose width and height are specified by width and height. To draw a circle, specify a square as the bounding rectangle.

The following program draws several ellipses:

```
// Draw Ellipses
import java.awt.*;
import java.applet.*;
/*
<applet code="Ellipses" width=300 height=200>
</applet>
*/
public class Ellipses extends Applet {
public void paint(Graphics g) {
g.drawOval(10, 10, 50, 50);
g.fillOval(100, 10, 75, 50);
g.drawOval(190, 10, 90, 30);
g.fillOval(70, 90, 140, 100);
}
}
```

Sample output from this program is shown here:



4. Drawing Arcs

Arcs can be drawn with drawArc( ) and fillArc( ), shown here:

Syntax:

void drawArc(int top, int left, int width, int height, intstartAngle,intsweepAngle)
void fillArc(int top, int left, int width, int height, intstartAngle,intsweepAngle)

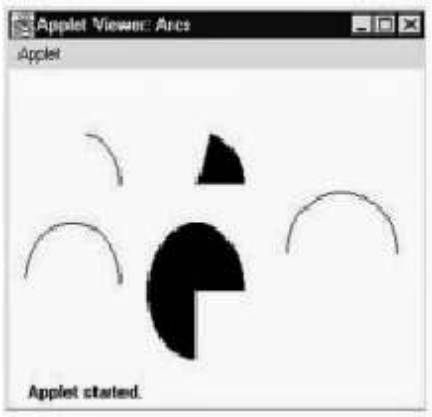
The arc is bounded by the rectangle whose upper-left corner is specified by top,left and whose width and height are specified by width and height. The arc is drawn from startAnglethrough the angular distance specified by sweepAngle. Angles are specified in degrees. Zero degrees are on the horizontal, at the three o'clock position. The arc is drawn counterclockwise ifsweepAngleis positive,and clockwise ifsweepAngleis negative. Therefore, to draw an arc from twelve o'clock to six o'clock, the start angle would be 90 and the sweep angle 180.

The following applet draws several arcs:

```
// Draw Arcs
import java.awt.*;
import java.applet.*;
/*
<applet code="Arcs" width=300 height=200>
</applet>
*/
public class Arcs extends Applet
{
public void paint(Graphics g)
 {
g.drawArc(10, 40, 70, 70, 0, 75);
g.fillArc(100, 40, 70, 70, 0, 75);
g.drawArc(10, 100, 70, 80, 0, 175);
g.fillArc(100, 100, 70, 90, 0, 270);
g.drawArc(200, 80, 80, 80, 0, 180);
}
```

}

Sample output from this program is shown here:



5. Drawing Polygons

It is possible to draw arbitrarily shaped figures using drawPolygon( ) and fillPolygon( ), shown here:

Syntax:

```
void drawPolygon(int x[ ], int y[ ], int numPoints)
void fillPolygon(int x[ ], int y[ ], int numPoints)
```

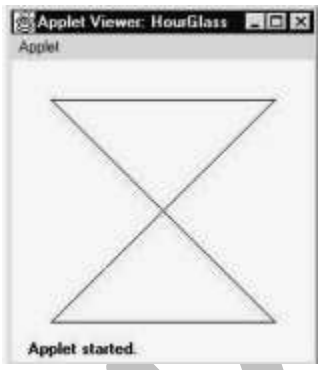
The polygon's endpoints are specified by the coordinate pairs contained within the x and y arrays. The number of points defined by x and y is specified by numPoints. There are alternative forms of these methods in which the polygon is specified by a Polygon object.

The following applet draws an polygon

```
// Draw Polygon
import java.awt.*;
import java.applet.*;
/*
<applet code="HourGlass" width=230 height=210>
</applet>
*/
```

```
public class HourGlass extends Applet
{
public void paint(Graphics g)
 {
int xpoints[] = {30, 200, 30, 200, 30};
int ypoints[] = {30, 30, 200, 200, 30};
int num = 5;
g.drawPolygon(xpoints, ypoints, num);
}
}
```

Sample output from this program is shown here:



Incorporating Images & Sounds:

Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Syntax of drawImage() method:

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)

is used draw the specified image.

How to get the object of Image?

The java.applet.Applet class provides getImage() method that returns the object of Image.

Syntax:

```
public Image getImage(URL u, String image){}
```

Other required methods of Applet class to display image:

public URL getDocumentBase(): is used to return the URL of the document in which applet is embedded.

public URL getCodeBase(): is used to return the base URL.

Example of displaying image in applet:

```
import java.awt.*;
import java.applet.*;
public class DisplayImage extends Applet {
Image picture;
public void init() {
   picture = getImage(getDocumentBase(),"sonoo.jpg");    }
   public void paint(Graphics g) {
   g.drawImage(picture, 30,30, this);   }
 }
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

How to play sound using Applet?

Following example demonstrates how to play a sound using an applet image using getAudioClip(), play() & stop() methods of AudioClip() class.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class PlaySoundApplet extends Applet implements ActionListener {
  Button play,stop;
  AudioClip audioClip;
  public void init() {
    play = new Button(" Play in Loop ");
    add(play);
    play.addActionListener(this);
    stop = new Button(" Stop ");
    add(stop);
    stop.addActionListener(this);
    audioClip = getAudioClip(getCodeBase(), "Sound.wav");
```

```
}
public void actionPerformed(ActionEvent ae) {
  Button source = (Button)ae.getSource();
  if (source.getLabel() == " Play in Loop ") {
    audioClip.play();
  } else if(source.getLabel() == "  Stop  "){
    audioClip.stop();     }   }}
```

Result

The above code sample will produce the following result in a java enabled web browser.

View in Browser.

EventHandling Mechanisms:

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button.

Example of EventHandling in applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener{
Button b;
TextField tf;
public void init(){
tf=new TextField();
tf.setBounds(30,40,150,20);
```

```
b=new Button("Click");
b.setBounds(80,150,60,50);
  add(b);add(tf);
b.addActionListener(this);
setLayout(null);
}
 public void actionPerformed(ActionEvent e){
  tf.setText("Welcome");   }   }
```

In the above example, we have created all the controls in init() method because it is invoked only once.

myapplet.html

```
<html>
<body>
<applet code="EventApplet.class" width="300" height="300">
</applet>
</body>  </html>
```

Listener Interfaces:

Java AWT Listeners are a group of interfaces from **java.awt.event** package. Listeners are capable to handle the events generated by the components like button, choice, frame etc. These listeners are implemented to the class which requires handling of events.

public class ButtonDemo1 extends Frame implements ActionListener

The class **ButtonDemo1** implements ActionListener as ButtonDemo1 includes some buttons which require event handling. The button events (known as action events) are handled by**ActionListener**.

Even though, some listeners handle the events of a few components, generally every component event is handled by a separate listener. For example, the **ActionListener** handles the events of **Button**, **TextField**, **List** and **Menu**. But these types are very rare.

Table giving list of few Java AWT Listeners and components whose events the listeners can handle.

LISTENER INTERFACE	COMPONENT
WindowListener	Frame
ActionListener	Button, TextField, List, Menu
ItemListener	Checkbox, Choice, List
AdjustmentListener	Scrollbar
MouseListener	Mouse (stable)
MouseMotionListener	Mouse (moving)
KeyListener	Keyboard

Adapter and Inner Classes:

Java Adapter Classes:

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So, it saves *code*.

The adapter classes are found in **java.awt.event**, **java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener

DragTargetAdapter	DragTargetListener

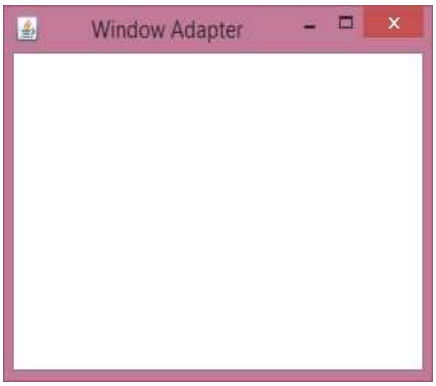
javax.swing.event Adapter classes

Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Java WindowAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample{
   Frame f;
   AdapterExample(){
      f=new Frame("Window Adapter");
      f.addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e) {
            f.dispose();            }         }
             f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);      }
public static void main(String[] args) {
   new AdapterExample();  }  }
```

Output:



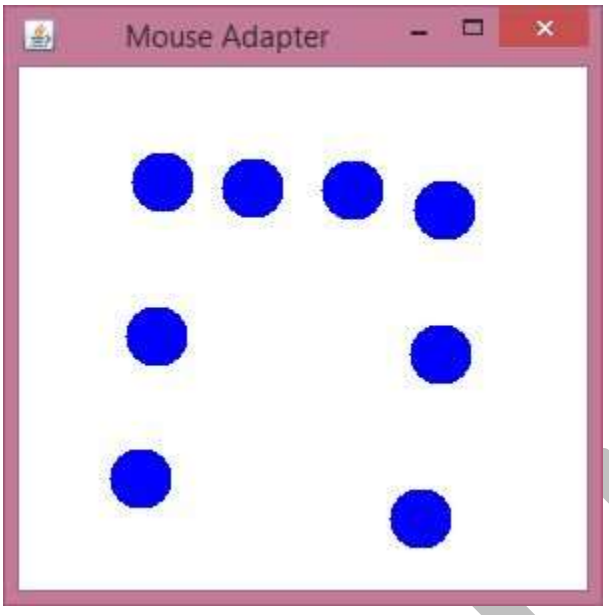
Java MouseAdapter Example

https://www.javatpoint.com/java-adapter-classeshttps://www.javatpoint.com/java-adapter-classeshttps://www.javatpoint.com/java-adapter-classes

```
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter{
   Frame f;
   MouseAdapterExample(){
      f=new Frame("Mouse Adapter");
      f.addMouseListener(this);

      f.setSize(300,300);
      f.setLayout(null);
      f.setVisible(true);
   }
   public void mouseClicked(MouseEvent e) {
      Graphics g=f.getGraphics();
      g.setColor(Color.BLUE);
      g.fillOval(e.getX(),e.getY(),30,30);
   }
    public static void main(String[] args) {
   new MouseAdapterExample();  }  }
```

Output:



Java MouseMotionAdapter Example

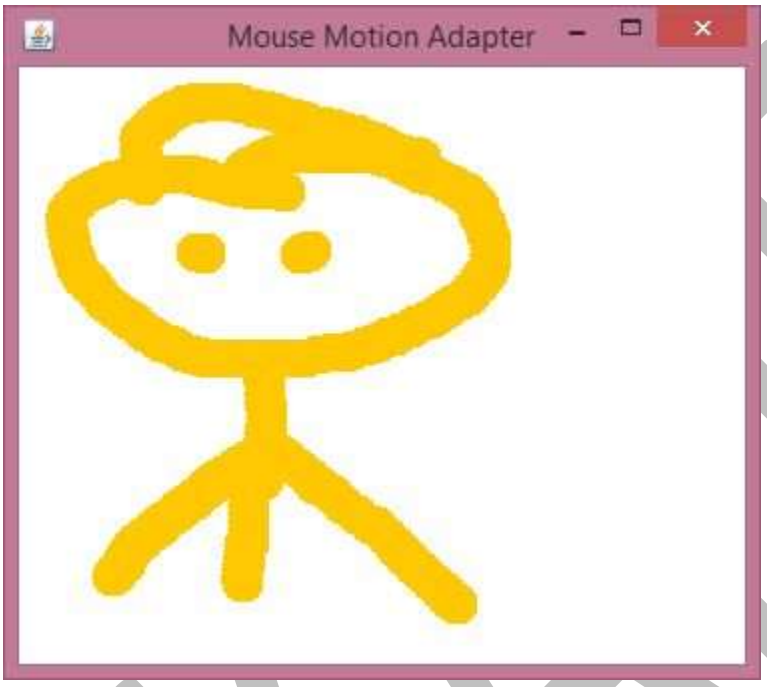
https://www.javatpoint.com/java-adapter-classeshttps://www.javatpoint.com/java-adapter-classeshttps://www.javatpoint.com/java-adapter-classes

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionAdapterExample extends MouseMotionAdapter{
   Frame f;
   MouseMotionAdapterExample(){
      f=new Frame("Mouse Motion Adapter");
      f.addMouseMotionListener(this);

      f.setSize(300,300);
      f.setLayout(null);
      f.setVisible(true);
   }
public void mouseDragged(MouseEvent e) {
   Graphics g=f.getGraphics();
   g.setColor(Color.ORANGE);
   g.fillOval(e.getX(),e.getY(),20,20);
}
public static void main(String[] args) {
```

```
    new MouseMotionAdapterExample();
}
}
```

Output:



Java KeyAdapter Example

https://www.javatpoint.com/java-adapter-classeshttps://www.javatpoint.com/java-adapter-classeshttps://www.javatpoint.com/java-adapter-classes

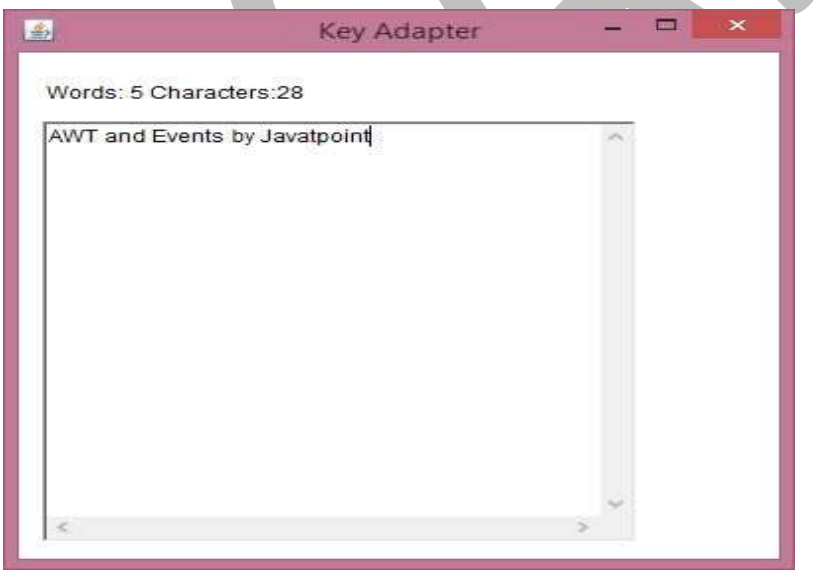
```
import java.awt.*;
import java.awt.event.*;
public class KeyAdapterExample extends KeyAdapter{
    Label l;
    TextArea area;
    Frame f;
    KeyAdapterExample(){
        f=new Frame("Key Adapter");
        l=new Label();
        l.setBounds(20,50,200,20);
```

```
        area=new TextArea();
        area.setBounds(20,80,300, 300);
        area.addKeyListener(this);

        f.add(l);f.add(area);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void keyReleased(KeyEvent e) {
        String text=area.getText();
        String words[]=text.split("\\s");
        l.setText("Words: "+words.length+" Characters:"+text.length());
    }

    public static void main(String[] args) {
        new KeyAdapterExample();
    }
}
```

Output:



Java Inner Classes

Java inner class or nested class is a class which is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

Syntax of Inner class

```
class Java_Outer_class{
 //code
 class Java_Inner_class{
  //code
 }
}
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

3) **Code Optimization**: It requires less code to write.

Difference between nested class and inner class in Java

Inner class is a part of nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

There are two types of nested classes non-static and static nested classes.The non-static nested classes are also known as inner classes.

- Non-static nested class (inner class)
    1. Member inner class
    2. Anonymous inner class
    3. Local inner class
- Static nested class

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interface.

Java Swing

Java Swing is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

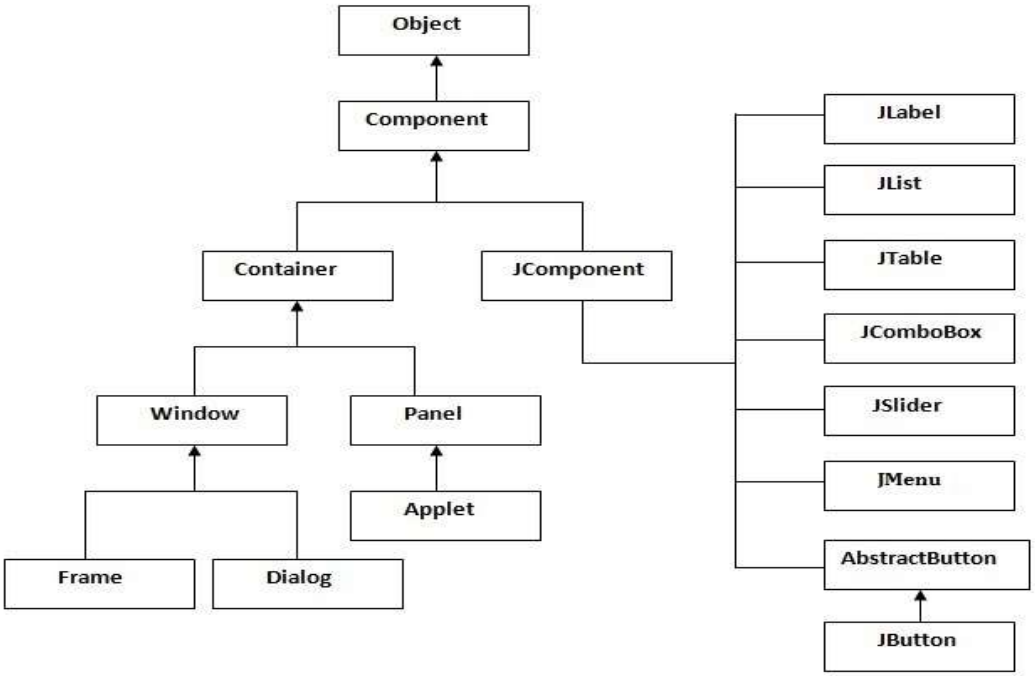
No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWTcomponents are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components**such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager	sets the layout manager for the component.

m)	
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

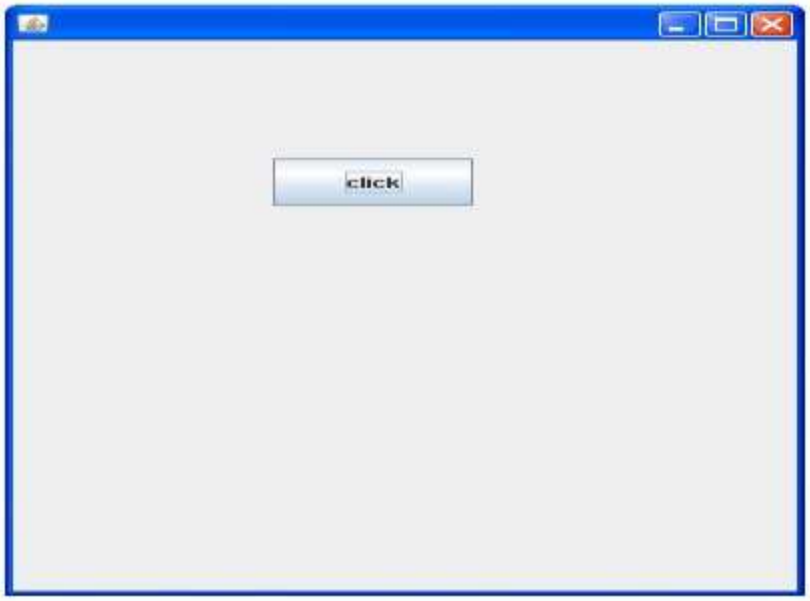
```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
```

```
}
}
```



Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

```
import javax.swing.*;
public class Simple {
JFrame f;
Simple(){
f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);

f.add(b);//adding button in JFrame
```

```
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}

public static void main(String[] args) {
new Simple();
}
}
```

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

File: Simple2.java

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

Java JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

```
public class JButton extends AbstractButton implements Accessible
```

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.

void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JButton Example

```
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:

Java JButton Example with ActionListener

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    final JTextField tf=new JTextField();
    tf.setBounds(50,50, 150,20);
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    b.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
            tf.setText("Welcome to Javatpoint.");
        }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:

Java JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

public class JLabel **extends** JComponent **implements** SwingConstants, Accessible

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the

	title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Commonly used Methods:

Methods	Description
String getText()	t returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.

int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Java JLabel Example

```
import javax.swing.*;
class LabelExample
{
public static void main(String args[])
    {
    JFrame f= new JFrame("Label Example");
    JLabel l1,l2;
    l1=new JLabel("First Label.");
    l1.setBounds(50,50, 100,30);
    l2=new JLabel("Second Label.");
    l2.setBounds(50,100, 100,30);
    f.add(l1); f.add(l2);
    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
    }
    }
```

Output:

Java JLabel Example with ActionListener

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class LabelExample extends Frame implements ActionListener{
   JTextField tf; JLabel l; JButton b;
   LabelExample(){
      tf=new JTextField();
      tf.setBounds(50,50, 150,20);
      l=new JLabel();
      l.setBounds(50,100, 250,20);
      b=new JButton("Find IP");
      b.setBounds(50,150,95,30);
      b.addActionListener(this);
      add(b);add(tf);add(l);
      setSize(400,400);
      setLayout(null);
      setVisible(true);
   }
   public void actionPerformed(ActionEvent e) {
      try{
```

```
    String host=tf.getText();
    String ip=java.net.InetAddress.getByName(host).getHostAddress();
    l.setText("IP of "+host+" is: "+ip);
    }catch(Exception ex){System.out.println(ex);}
  }
  public static void main(String[] args) {
    new LabelExample();
  } }
```

Output:

Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

JTextField class declaration

Let's see the declaration for javax.swing.JTextField class.

```
public class JTextField extends JTextComponent implements SwingConstants
```

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods:

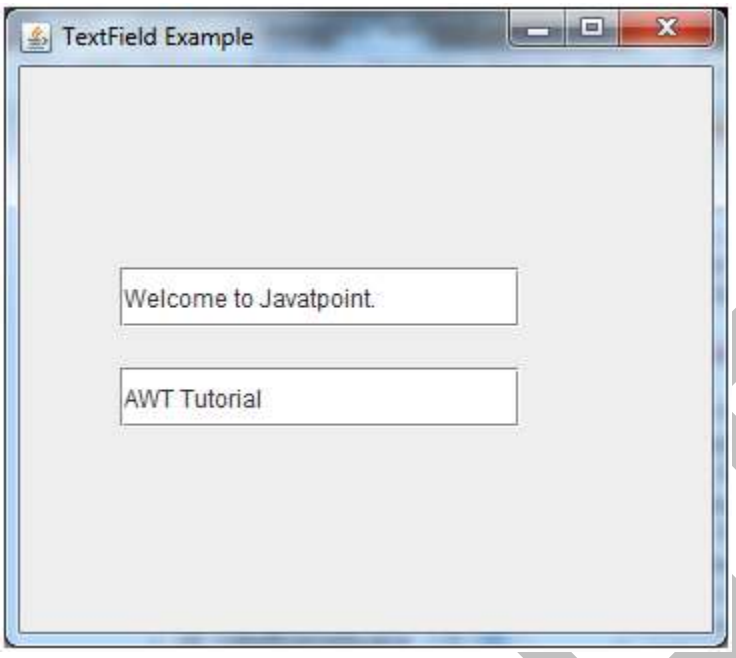
Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.

void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

Java JTextField Example

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
    {
    JFrame f= new JFrame("TextField Example");
    JTextField t1,t2;
    t1=new JTextField("Welcome to Javatpoint.");
    t1.setBounds(50,100, 200,30);
    t2=new JTextField("AWT Tutorial");
    t2.setBounds(50,150, 200,30);
    f.add(t1); f.add(t2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
    }
    }
```

Output:



Java JTextField Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class TextFieldExample implements ActionListener{
   JTextField tf1,tf2,tf3;
   JButton b1,b2;
   TextFieldExample(){
      JFrame f= new JFrame();
      tf1=new JTextField();
      tf1.setBounds(50,50,150,20);
      tf2=new JTextField();
      tf2.setBounds(50,100,150,20);
      tf3=new JTextField();
      tf3.setBounds(50,150,150,20);
      tf3.setEditable(false);
      b1=new JButton("+");
      b1.setBounds(50,200,50,50);
      b2=new JButton("-");
      b2.setBounds(120,200,50,50);
```

```
        b1.addActionListener(this);
        b2.addActionListener(this);
        f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        String s1=tf1.getText();
        String s2=tf2.getText();
        int a=Integer.parseInt(s1);
        int b=Integer.parseInt(s2);
        int c=0;
        if(e.getSource()==b1){
            c=a+b;
        }else if(e.getSource()==b2){
            c=a-b;
        }
        String result=String.valueOf(c);
        tf3.setText(result);
    }
public static void main(String[] args) {
    new TextFieldExample();
} }
```

Output:

Java JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

JMenuBar class declaration

public class JMenuBar **extends** JComponent **implements** MenuElement, Accessible

JMenu class declaration

public class JMenu **extends** JMenuItem **implements** MenuElement, Accessible

JMenuItem class declaration

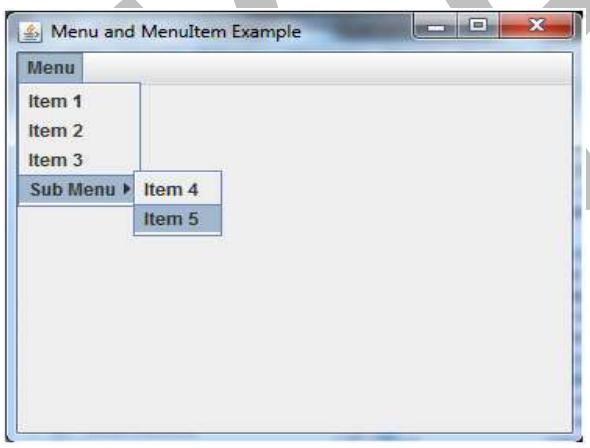
public class JMenuItem **extends** AbstractButton **implements** Accessible, MenuElement

Java JMenuItem and JMenu Example

```
import javax.swing.*;
class MenuExample
{
    JMenu menu, submenu;
    JMenuItem i1, i2, i3, i4, i5;
    MenuExample(){
    JFrame f= new JFrame("Menu and
MenuItem Example");
    JMenuBar mb=new JMenuBar();
    menu=new JMenu("Menu");
    submenu=new JMenu("Sub Menu");

    i1=new JMenuItem("Item 1");
    i2=new JMenuItem("Item 2");
    i3=new JMenuItem("Item 3");
    i4=new JMenuItem("Item 4");
    i5=new JMenuItem("Item 5");
    menu.add(i1); menu.add(i2); menu.a
dd(i3);
    submenu.add(i4); submenu.add(i5);
    menu.add(submenu);
    mb.add(menu);
    f.setJMenuBar(mb);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}}
```

Output:

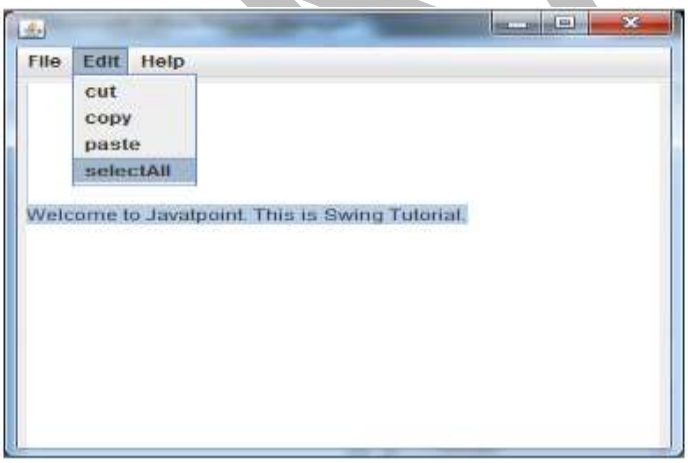


Example of creating Edit menu for Notepad:

```
import javax.swing.*;
import java.awt.event.*;
public class MenuExample implements A
ctionListener{
```

```
JFrame f;
JMenuBar mb;
JMenu file,edit,help;
JMenuItem cut,copy,paste,selectAll;
JTextArea ta;
MenuExample(){
f=new JFrame();
cut=new JMenuItem("cut");
copy=new JMenuItem("copy");
paste=new JMenuItem("paste");
selectAll=new JMenuItem("selectAll");
cut.addActionListener(this);
copy.addActionListener(this);
paste.addActionListener(this);
selectAll.addActionListener(this);
mb=new JMenuBar();
file=new JMenu("File");
edit=new JMenu("Edit");
help=new JMenu("Help");
edit.add(cut);edit.add(copy);
edit.add(paste);
edit.add(selectAll);
mb.add(file);mb.add(edit);mb.add(help);
ta=new JTextArea();
ta.setBounds(5,5,360,320);
f.add(mb);f.add(ta);
f.setJMenuBar(mb);
f.setLayout(null);
f.setSize(400,400);
f.setVisible(true);
}
public void actionPerformed(ActionEvent
e) {
if(e.getSource()==cut)
ta.cut();
if(e.getSource()==paste)
ta.paste();
if(e.getSource()==copy)
ta.copy();
if(e.getSource()==selectAll)
ta.selectAll();
}
public static void main(String[] args) {
   new MenuExample();   }   }
```

Output:



BorderLayout (LayoutManagers)

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

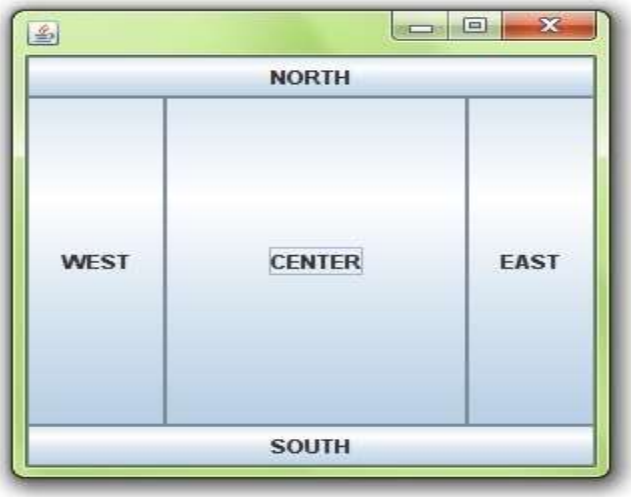
1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.

- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class:




```
import java.awt.*;
import javax.swing.*;

public class Border {
JFrame f;
Border(){
   f=new JFrame();

   JButton b1=new JButton("NORTH");;
   JButton b2=new JButton("SOUTH");;
   JButton b3=new JButton("EAST");;
   JButton b4=new JButton("WEST");;
   JButton b5=new JButton("CENTER");;

   f.add(b1,BorderLayout.NORTH);
   f.add(b2,BorderLayout.SOUTH);
```

```
    f.add(b3,BorderLayout.EAST);
    f.add(b4,BorderLayout.WEST);
    f.add(b5,BorderLayout.CENTER);

    f.setSize(300,300);
    f.setVisible(true);
  }
  public static void main(String[] args) {
    new Border();
  }
  }
```

An Overview of Servlet

Servlet technology is used to create web application (resides at server side and generates dynamic web page).

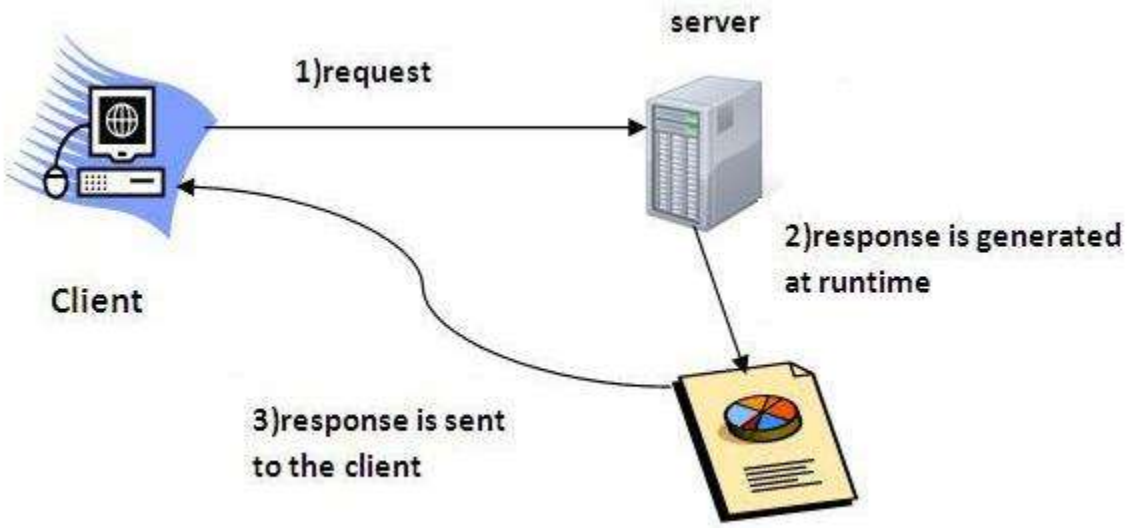
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

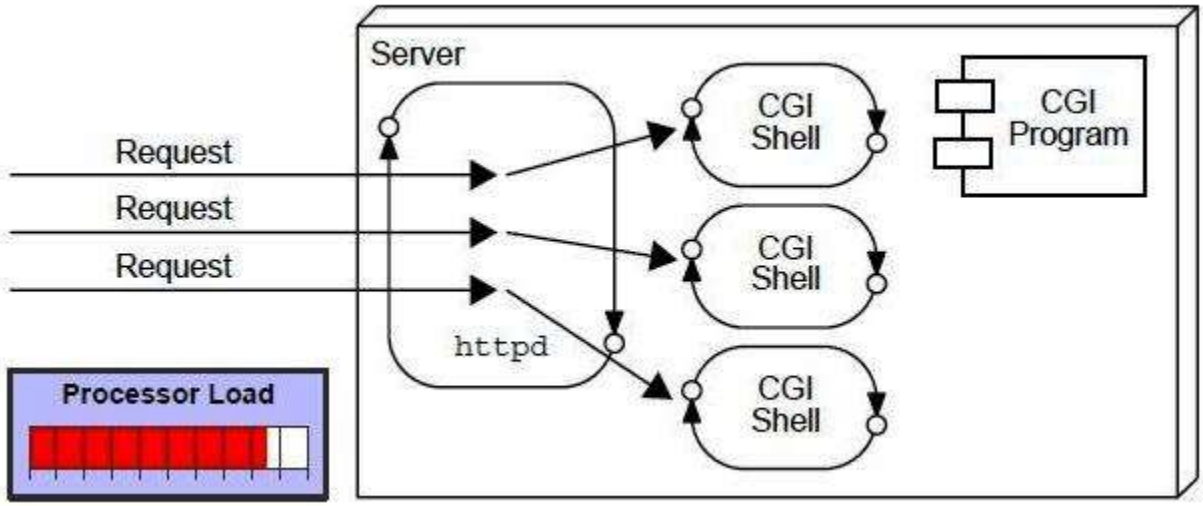


What is web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

CGI(Commmon Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

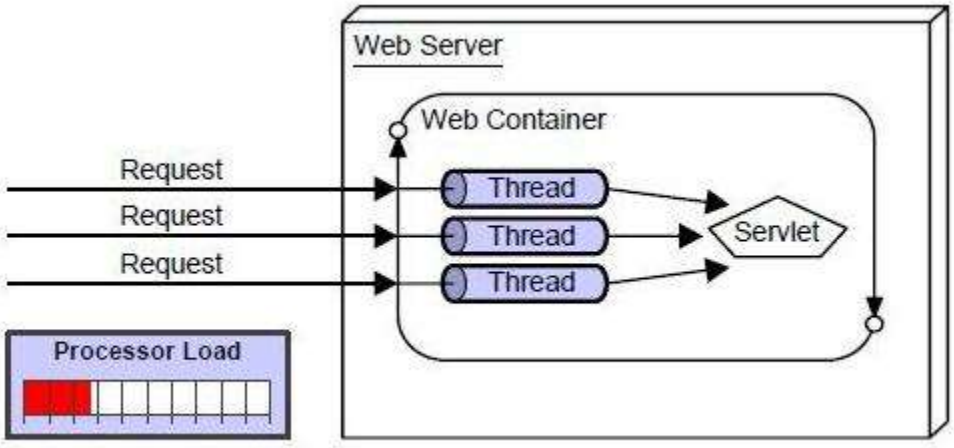


Disadvantages of CGI

There are many problems in CGI technology:

1. If number of clients increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantage of Servlet



There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

1. **better performance:** because it creates a thread for each request not process.
2. **Portability:** because it uses java language.
3. **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
4. **Secure:** because it uses java language.

Event and Listener in Servlet

Events are basically occurrence of something. Changing the state of an object is known as an event.

We can perform some important tasks at the occurrence of these exceptions, such as counting total and current logged-in users, creating tables of the database at time of deploying the project, creating database connection object etc.

There are many Event classes and Listener interfaces in the javax.servlet and javax.servlet.http packages.

Event classes

The event classes are as follows:

1. ServletRequestEvent
2. ServletContextEvent
3. ServletRequestAttributeEvent
4. ServletContextAttributeEvent
5. HttpSessionEvent
6. HttpSessionBindingEvent

Event interfaces

The event interfaces are as follows:

1. ServletRequestListener
2. ServletRequestAttributeListener
3. ServletContextListener
4. ServletContextAttributeListener
5. HttpSessionListener
6. HttpSessionAttributeListener
7. HttpSessionBindingListener
8. HttpSessionActivationListener

POSSIBLE QUESTIONS

2 MARKS

1. What are packages and how it is used?

2. Define and give the syntax for interface.

3. Write the difference between an applet and an application.

4. Draw the life cycle of Java Applets.

5. What are Java Packages?

6. What is Thread prioritization?

7. Write any two AWT Controls.

8. What is servelets?

9. What is web application?

10. Define CGI(Commmon Gateway Interface).

11. Mention the Disadvantages of CGI.

12. What are the Advantages of Servlet?

13. What is Java Swing?

14. Difference between AWT and Swing.

15. What is JFC?

16. Draw the Hierarchy of Java Swing classes.

17. What are the Advantage of Applet?

18. List the Drawback of Applet.

19. Draw the Hierarchy of Applet.

20. How to run an Applet?

6 MARKS

1. What are the different types of AWT components? How are these components added to the container?

2. Write an applet that draws circle, a line, and a polygon inside the applet's visible area.

3. What do you mean by an event? Explain different components of an event.

4. Define applet. Write an applet that draws circle, a line, and a polygon inside the applet's visible area

5. Explain the lifecycle of applet and Write an applet that draws circle, a line, and a polygon inside the applet's visible area

6. Discuss Graphic objects and methods with example.

7. Explain Event Handling Mechanism with an example.

8. Explain Graphic Object for drawing figures with example program.

9. Elucidate about Applets and its Life Cycle and Methods with appropriate examples.

10. Display various shapes in a window using menu as options for those shapes.

11. Explain in detail about working with graphics with example.

12. Discuss in detail about Adapter and inner classes.

13. Enlighten incorporating images and sounds with example.

KAHE

KARPAGAM ACADEMY OF HIGHER EDUCATION

COIMBATORE - 21

DEPARTMENT OF COMPUTER SCIENCE,CA & IT

CLASS : I BCA

BATCH : 2018-2021

Part -A Online Examinations (1 mark questions)

SUBJECT: Programming in Java SUBJECT CODE: 18CAU201

UNIT 5

S. No	Questions	opt1	opt2	opt3	opt4	answer
1	AWT stands for _____	Abstract Window Toolkit	Absolute Window Toolkit	Absolute Windowing Toolkit	Abstract Windowing Toolkit	Abstract Window Toolkit
2	The _____ class is an abstract class which includes large number of methods for positioning and sizing components, repainting, etc.	Container	Component	Swing	Beans	Component
3	The _____ class creates a push button that generates an event when it is pressed	Label	TextField	Button	Checkbox	Button
4	In which class is the method setVisible() first defined	Component	Container	Window	Frame	Component
5	While creating a window using Frame constructor, we can specify	Title	Size	Visibility attribute	All	Title
6	Color class also defines _____ common colors as constants	Canvas	Frame	Dialog	Panel	Dialog
7	The most basic menu in an application consists of _____ main elements	2	4	3	5	3
8	The _____ class implements a scrollable list of text items	Choice	Checkbox	List	Scrollbar	List
9	A TextField is a subclass of the _____ class	TextComponent	Button	Label	TextArea	TextComponent
10	Checkbox consists of _____ states	2	4	3	5	2
11	Checkbox can of _____ types	4	3	5	2	2
12	A _____ list appears like a menu	TextArea	choice	Component	Container	choice

13	_____ are used to select continuous values bewtween a specified minimum and maximum.	List	Scrollbar	TextField	TextArea	Scrollbar
14	_____ is an abstract subclass of the abstract class component	Component	Container	Window	Frame	Container
15	Applet is a subclass of _____ class	List	Scrollbar	Panel	Frame	Panel
16	_____ provides a basic file Open/Save dialog box that enables accessibility to the file system	Dialog	Frame	FileDialog	Container	FileDialog
17	The frame must be closed explicitly by adding _____ object to the frame	WindowListener	FileDialog	Dialog	Applet	WindowListener
18	_____ class provides a compact multiple choice scrolling selection list.	Scroll bar	List	menu bar	Combo Box	List
19	Which of these functions is called to display the output of an applet?	display()	print()	displayApplet()	PrintApplet()	print()
20	____ parameters are passed to drawArc method	4	5	6	3	3
21	____ is the default color for drawing graphics color	white	black	red	green	black
22	how many colors does a GIF image can have?	180	256	3600	4800	256
23	when a portion of a applet window is to be redrawn ____ method is used	paint()	start()	update()	repaint()	update()
24	_____ method is used set the background color	setbackGround()	Setcolor()	setBackGround()	setBackground()	setBackground()
25	_____is the distance from the base line to the top of the character	font size	ascent	descent	baseline	ascent
26	_____is the distance from the base line to the bottom of the character	font size	ascent	descent	baseline	descent
27	_____is an abstract class which encapsulates all the attributes of the visual component	component	container	applet	both b and c	component
28	To get the URL of the applet, you use _____.	getCodeBase()	getDocumentBase()	returnCodeBase()	returnDocumentBase	getCodeBase()

29	To get the image file at a specified URL, you use _____.	getImage(url)	createImage(url)	url.getImage()	url.createImage()	getImage(url)
30	The _____ method of class Graphics draw a line between two points	Line	Putline	drawline	getline	drawline
31	What is the data type for the parameter of the sleep() method?	long	int	byte	double	long
32	What is the mechanisam defind by java for the Resources to be used by only one Thread at a time?	priority	parameters	arguments	Synchronisation	Synchronisation
33	Garbage collector thread belongs to which priority?	high-priority	low-priority	middle-priority	highest-priority	low-priority
34	When a Java program starts up, ____ thread begins running immediately	program	main	function	input	main
35	The ____ method causes the thread from which it is called to suspend execution for the specified period of	wait()	notify()	sleep( )	run()	sleep( )
36	To implement Runnable, a class need only implement a single method called ____	wait()	notify()	sleep( )	run()	run()
37	A ____ is an object that is used as a mutually exclusive lock to achieve synchronization	monitor	thread	process	applet	monitor
38	_____ are small applicationsthat are accessed on an internet server	utilities	networks	applets	bean	applets
39	The compiled applet is tested using _____	word	dos	notepad	applet viewer	applet viewer
40	The _____ tag is used to start an applet from both HTML and JDK applet viewer	Html	JDK	applet	title	applet
41	_____ method gets called first	paint	start	init	update	init
42	Applet basically is a Java class defined in the _____ package of JDK	java.awt	java.lang	java.applet	java.util	java.applet
43	The Applet class which is in the java.applet package inherits the properties of the _____ class which	Container	Componenet	Panel	List	Panel
44	The Panel class inherits the properties of the _____ class in the java.awt package	Container	Componenet	Panel	List	Container
45	The container class inherits the properties of the _____ class	Container	Componenet	Panel	List	Componenet

46	An _____ is a window based event driven program	Html	JDK	applet	title	applet
47	The _____ and _____ method executes only once	stop() and destroy()	start() and stop()	init() and paint()	init() and destroy()	init() and destroy()
48	Immediately after calling init() methodthe browser calls the _____ method	stop()	start()	init()	destroy()	start()
49	The _____ method also called when the user returns to an HTML page that contains the applet	paint()	init()	destroy()	start()	start()
50	The _____ methodis called each time your applet's output is redrawn	stop()	start()	init()	paint()	paint()
51	The _____ method acalled when the user moves from the HTML page that contains an applet	paint()	init()	stop()	destroy()	stop()
52	The _____ method that is used to release additional resource	paint()	init()	destroy()	start()	destroy()
53	There are _____ main methods defined in java.awt.Component	2	4	5	3	3
54	The _____ method is defined by the AWT and is usually called by the applet for screen updating	paint()	init()	stop()	repaint()	repaint()
55	_____ class cannot be created directly by using constructors	Panel	Container	Componen et	Graphics	Grapahics
56	In java color is encapsulated by the _____ class	Container	Componenet	Graphics	Color	Color
57	Color class also defines _____ common colors as constants	10	13	12	14	13
58	Methods of _____ class can also be used in the Graphiocs class methods to set and get the background	Container	Componenet	Panel	List	Compone net
59	There are _____ common terms that are used when describing fonts	2	4	5	3	5
60	The java.applet package defines _____ inetrfaces	2	4	5	3	3
61	The user cannot have their HTML document,applet code,data and web browser at _____ different	2	4	5	3	4
62	The loop() method plays the audio clip automatically while _____ plays it only once	paint()	play()	init()	start()	play()

63	The audio clip can be stopped by calling the _____ method	paint()	init()	stop()	repaint()	stop()
64	The _____ interface provides the inter_communication between an applet and the parent container	AppletContext	AppletStub	getApplet	showDocument	AppletStub
65	The _____ inetface gives the information about the applet's execution environment	AppletStub	getApplet	AppletContext	showDocument	AppletContext
66	The setBackground() is the part of the class _____	Graphics	AppletStub	Component	Container	Component
67	If you want to assign a vlaue 99 to a variable called number, which of the following lines you will use within an	number=99	param = number value=99	param name = number	param number =99	param name = number

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021

INFORMATION TECHNOLOGY/COMPUTER TECHNOLOGY/

COMPUTER SCIENCE/COMPUTER APPLICATIONS

Second Semester
FIRST INTERNAL EXAMINATION - December 2018

PROGRAMMING IN JAVA

Class & Section: I B.Sc (IT), CT, CS & BCA **Duration: 2 hours**
Date & Session : 18.12.18 & FN **Maximum marks: 50 marks**
Subj.Code: 18ITU201/18CTU201/18CSU201/18CAU201

PART- A (20 * 1= 20 Marks)
Answer ALL the Questions

1. Java is a _____ language
   a. structured programming
   c. procedural oriented
   b. object oriented
   d. machine
2. Variables are declared in_____
   a. only in main()
   c. before the main() only
   b. anywhere in the scope
   d. only at the beginning
3. _____refers to permit initialization of the variables at run time
   a. Dynamic initialization
   c. Data binding
   b. Dynamic binding
   d. Dynamic message
4. Keyword _____ indicates that method do not return any value.
   a. Static b. Final **c. void** d. null
5. Java does not have _____ statement
   a. goto b. if c. do d. do while
6. byte belongs to _____ type.
   a. character b. Boolean c. floating **d. integer**
7. In Java an int is _____ bits
   a. 16 b. 64 c. 52 **d. 32**
8. byte is a signed _____ type
   a. 16 bit **b. 8 bit** c. 32 bit d. 64 bit
9. The _____ statement is often used in switch statement
   a. break b. end c. do d. none
10. _____ translates the Java source code to bytecode files that the interpreter can understand
    a. javac b. java c. javap d. jdk
11. In java the functions are called as _____
    a. fields **b. method** c. variables d. none
12. Java does not have _____
    a. struct b. header files c. union **d. all**
13. _____ is a access specifier
    a. static b. void main **c. public** d. none

14. Data type Short occupies _____ bytes.
    a. 1 **b. 2** c. 4 d. 8
15. A built_in class which encapsulates the data structure of a string is _____
    a. java io **b. String** c. Character d. None
16. The instances of the class String is created using _____
    a. new b. free c. object d. None
17. To extract a single character from a string , the _____ method is used.
    a. charAt b. Stringto c. charOne d. None
18. To get the substring from a string _____ method is used.
    a. getchars b. substr c. extract d. substring
19. The _____ method compares the characters inside the string.
    a. = = b. equivalent **c. equals** d. None
20. The search for a certain character or substring is done using _____ & _____.
    a. index & indexof **c. indexof & lastindexof**
    b. index & lastindex d. None

PART- B (3 * 2= 6 Marks)
Answer ALL the Questions

21. What is Java Tokens?

In a Java program, all characters are grouped into symbols called **tokens**. Larger language features are built from the first five categories of tokens (the sixth kind of token is recognized, but is then discarded by the Java compiler from further processing). We must learn how to identify all six kinds of tokens that can appear in Java programs. In EBNF we write one simple rule that captures this structure:

token = identifier | keyword | separator | operator | literal | comment

The different types of Tokens are:

1. Identifiers: names the programmer chooses
2. Keywords: names already in the programming language
3. Separators (also known as punctuators): punctuation characters and paired-delimiters
4. Operators: symbols that operate on arguments and produce results
5. Literals (specified by their **type**)
    - Numeric: **int** and **double**
    - Logical: **boolean**
    - Textual: **char** and **String**

- Reference: **null**

6. Comments
    - Line

22. Give any 4 differences between C++ and Java.

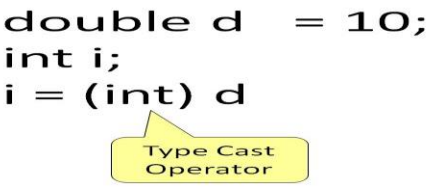
S.NO	Comparison Index	C++	Java
1	Platform-independent	C++ is platform-dependent.	Java is platform-independent.
2	Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
3	Goto	C++ supports goto statement.	Java doesn't support goto statement.
4	Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
5	Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.

23. Define Type Conversion with example.

A variable of one type can receive the value of another type. Here there are 2 cases - **case 1)** Variable of smaller capacity is be assigned to another variable of bigger capacity.

```
double d ;
int i = 10;
d = i;
```

This process is Automatic, and non-explicit is known as ***Conversion***
case 2) Variable of larger capacity is be assigned to another variable of smaller capacity



In such cases you have to explicitly specify the **type cast operator. This process is known as *Type Casting.***

PART C (3 * 8 = 24 Marks)
Answer ALL the Questions

24. a. Discuss operators in java with example.

Operators in Java

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-

1. Arithmetic Operators
2. Logical Operators
3. Bitwise Operators

Let's take a look at them in detail.

Arithmetic Operators: They are used to perform simple arithmetic operations on primitive data types.

1. **:** Multiplication
2. **/ :** Division
3. **% :** Modulo
4. **+ :** Addition
5. **:** Subtraction

```
// Java program to illustrate
// arithmetic operators
publicclassoperators
{
   publicstaticvoidmain(String[] args)
   {
      inta = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
      String x = "Thank", y = "You";

      // + and - operator
      System.out.println("a + b = "+(a + b));
      System.out.println("a - b = "+(a - b));

      // + operator if used with strings
      // concatenates the given strings.
      System.out.println("x + y = "+x + y);
```

```
        // * and / operator
        System.out.println("a * b = "+(a * b));
        System.out.println("a / b = "+(a / b));

        // modulo operator gives remainder
        // on dividing first operand with second
        System.out.println("a % b = "+(a % b));

        // if denominator is 0 in division
        // then Arithmetic exception is thrown.
        // uncommenting below line would throw
        // an exception
        // System.out.println(a/c);
    }
}
```

Output:

a+b = 30

a-b = 10

x+y = ThankYou

a*b = 200

a/b = 2

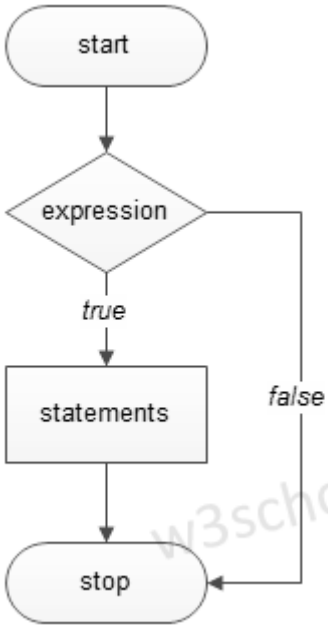
a%b = 0

Decision-making statements

A Java decision-making statement allows you to make decision, based upon the result of a condition.

All the programs in Java have set of statements, which are executed sequentially in the order in which they appear. This happens when jumping of statements or repetition of certain calculations is not necessary. However there may arise some situations where programmers have to change the order of execution of statements based on certain conditions which involves kind of decision-making statements. In this chapter you will learn about how the control flow statements works.

The flowchart of Decision making technique in Java can be expressed as:



Java has such decision making capabilities within its program by the use of following decision making statements:

Decision Making Statements in Java

- if Statement
  - if statement
  - if-else statement
  - else-if statement
- Conditional Operator
- switch statement

Java if Statements

If a statement in Java is used to control the program flow based on some condition, it's used to execute some statement code block if expression is evaluated to true, otherwise it will get skipped. This is an simplest way to modify the control flow of the program.

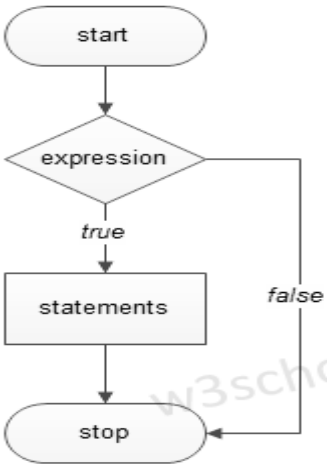
The basic format of if statement is:

Syntax:

```
if(test_expression)
{
   statement 1;
   statement 2;
   ...
}
```

'Statement n' can be a statement or a set of statements and if the test expression is evaluated to true, the statement block will get executed or it will get skipped.

Figure – Flowchart of if Statement:



Example of a Java Program to Demonstrate If statements

Example:

```
public class Sample{
  public static void main(String args[]){
    int a=20, b=30;
    if(b>a)
      System.out.println("b is greater");
    }}
```

Program Output:

```
run:
b is greater
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java if-else Statement

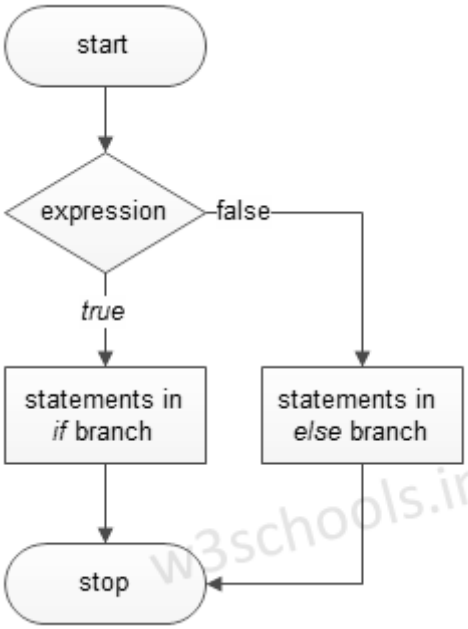
If else a statement in Java is also used to control the program flow based on some condition, only the difference is: it's used to execute some statement code block if expression is evaluated to true, otherwise executes else statement code block.

The basic format of if else statement is:

Syntax:

```
if(test_expression)
{
  //execute your code
}
else
{
  //execute your code
}
```

Figure – Flowchart of if else Statement:



Example of a Java Program to Demonstrate If else statements

Example:

```
public class Sample {
 public static void main(String args[]) {
  int a = 80, b = 30;
  if (b & gt; a) {
   System.out.println("b is greater");
  } else {
   System.out.println("a is greater");
  } }}
```

Program Output:

```
run:
a is greater
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java else-if Statements

else if statements in Java is like another if condition, it's used in program when if statement having multiple decisions.

The basic format of else if statement is:

Syntax:

```
if(test_expression)
{
  //execute your code
}
else if(test_expression n)
{
```

```
  //execute your code
}
else
{
  //execute your code
}
```

Example of a Java Program to Demonstrate else If statements

Example:

```
public class Sample {
   public static void main(String args[]) {
      int a = 30, b = 30;
      if (b > a) {
         System.out.println("b is greater");
      }
      else if(a > b){
         System.out.println("a is greater");
      }
      else {
         System.out.println("Both are equal");
      }   }}
```

Program Output:

```
run:
Both are equal
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java switch Statements

Java switch statement is used when you have multiple possibilities for the if statement.

The basic format of switch statement is:

Syntax:

```
switch(variable)
{
case 1:
  //execute your code
break;

case n:
  //execute your code
break;

default:
  //execute your code
break;
}
```

After the end of each block it is necessary to insert a break statement because if the programmers do not use the break statement, all consecutive blocks of codes will get executed from each and every case onwards after matching the case block.

Example of a Java Program to Demonstrate Switch Statement

Example:

```
public class Sample {
 public static void main(String args[]) {
  int a = 5;
  switch (a) {
   case 1:
    System.out.println("You chose One");
    break;
   case 2:
    System.out.println("You chose Two");
    break;
   case 3:
    System.out.println("You chose Three");
    break;
   case 4:
    System.out.println("You chose Four");
    break;
   case 5:
    System.out.println("You chose Five");
    break;
   default:
    System.out.println("Invalid Choice. Enter a no between 1 and 5");
    break;
  } }}
```

Program Output:

```
run:
You chose Five
BUILD SUCCESSFUL (total time: 0 seconds)
```

When none of the case is evaluated to true, then default case will be executed, and break statement is not required for default statement.

(OR)

b. Explain in detail about the features and architecture of JAVA.

Java Architecture:

1. Compilation and interpretation in Java

Java combines both the approaches of compilation and interpretation. First, java compiler compiles the source code into byte code. At the run time, Java Virtual Machine (JVM) interprets this byte code and generates machine code which will be directly executed by the machine in which java program runs. So java is both compiled and interpreted language.

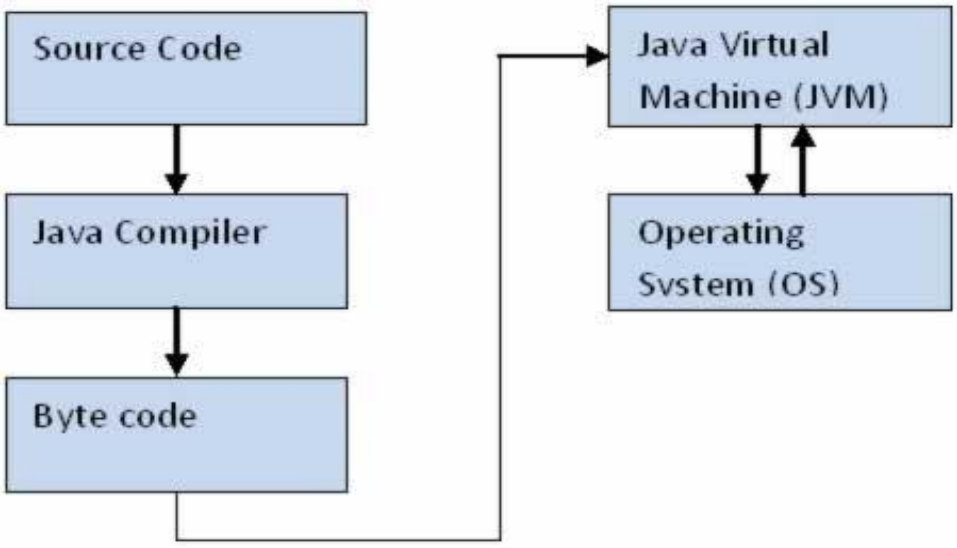


Figure 1.1: Java Architecture

2. Java Virtual Machine (JVM)

JVM is a component which provides an environment for running Java programs. JVM interprets the byte code into machine code which will be executed the machine in which the Java program runs. Java was developed with the concept of **WORA (*Write Once Run Anywhere)*** which runs on a **VM**. **The compiler** will be compiling the java file into a java **.class** file. The **.class** file is input to JVM which Loads and executes the class file. Below goes the Architecture of JVM.

Java Environment

The Java Virtual Machine

At the heart of Java's network-orientation is the Java virtual machine, which supports all three prongs of Java's network-oriented architecture: platform independence, security, and network-mobility.

The Java virtual machine is an abstract computer. Its specification defines certain features every Java virtual machine must have, but leaves many choices to the designers of each implementation. For example, although all Java virtual machines must be able to execute Java byte codes, they may use any technique to execute them. Also, the specification is flexible enough to allow a Java virtual machine to be implemented either completely in software or to varying degrees in hardware. The flexible nature of the Java virtual machine's specification enables it to be implemented on a wide variety of computers and devices.

A Java virtual machine's main job is to load class files and execute the byte codes they contain. As you can see in Figure 1-3, the Java virtual machine contains a *class loader*, which loads class files from both the program and the Java API. Only those class files from the Java API that are actually needed by a running program are loaded into the virtual machine. The byte codes are executed in *an execution engine.*

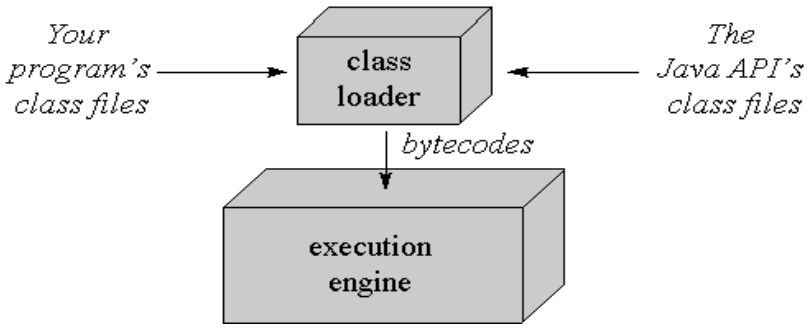


Figure 1-2. A basic block diagram of the Java virtual machine.

The execution engine is one part of the virtual machine that can vary in different implementations. On a Java virtual machine implemented in software, the simplest kind of execution engine just interprets the byte codes one at a time. Another kind of execution engine, one that is faster but requires more memory, is a *just-in-time compiler*. In this scheme, the byte codes of a method are compiled to native machine code the first time the method is invoked.

Java architecture

Java's architecture arises out of four distinct but interrelated technologies:

- the Java programming language

- the Java class file format
- the Java Application Programming Interface
- the Java virtual machine

When you write and run a Java program, you are tapping the power of these four technologies. You express the program in source files written in the Java programming language, compile the source to Java class files, and run the class files on a Java virtual machine. When you write your program, you access system resources (such as I/O, for example) by calling methods in the classes that implement the Java Application Programming Interface, or Java API. As your program runs, it fulfills your program's Java API calls by invoking methods in class files that implement the Java API. You can see the relationship between these four parts in Figure 1-1.

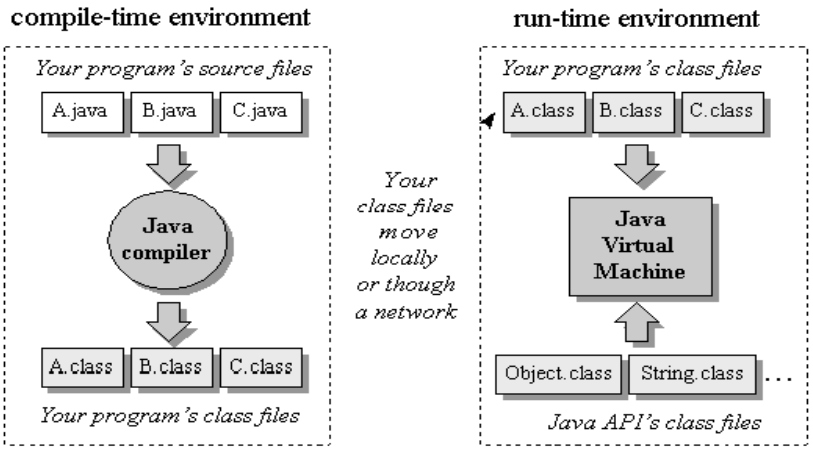


Figure 1-3. The Java programming environment.

Together, the Java virtual machine and Java API form a "platform" for which all Java programs are compiled. In addition to being called the *Java runtime system*, the combination of the Java virtual machine and Java API is called the *Java Platform* (or, starting with version 1.2, the *Java 2 Platform*). Java programs can run on many different kinds of computers because the Java Platform can itself be implemented in software. As you can see in Figure 1- 2, a Java program can run anywhere the Java Platform is present.

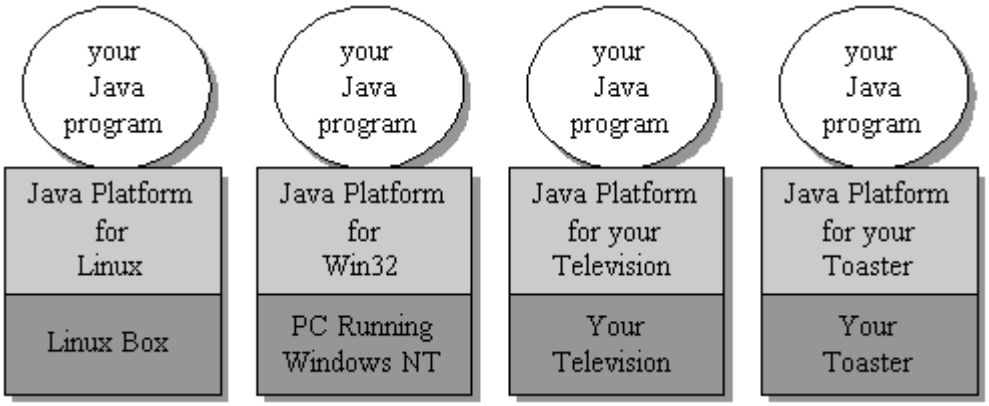


Figure 1-4. Java programs run on top of the Java Platform.

1.3.4 Java development kit

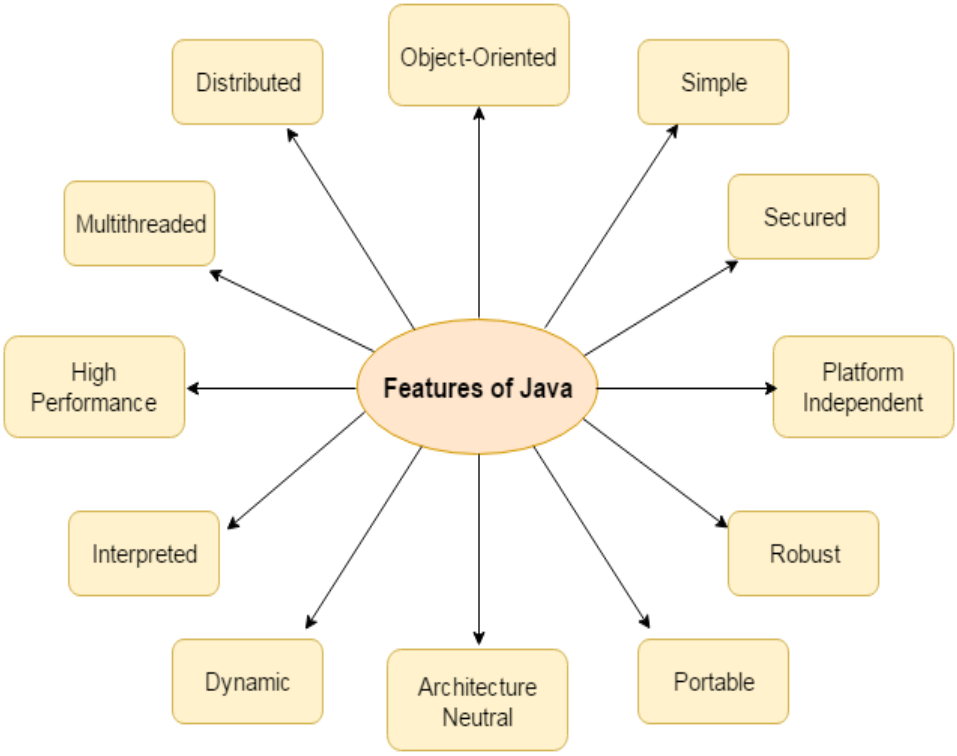
The **Java Development Kit** (**JDK**) is a Sun Microsystemsproduct aimed at Java developers. Since the introduction of Java, it has been by far the most widely used Java SDK. On 17 November 2006, Sun announced that it would be released under the GNU General Public License (GPL), thus making it free software. This happened in large part on 8 May 2007[3]; Sun contributed the source code to the OpenJDK.

The JDK has as its primary components a collection of programming tools, including:

- java – the loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment. The old deployment launcher, jre, no longer comes with Sun JDK.
- javac – the compiler, which converts source code into Java byte code
- applet viewer – this tool can be used to run and debug Java applets without a web browser

Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.



1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

1. Simple

- According to Sun, Java language is simple because:
- Syntax is based on C++ (so easier for programmers to learn it after C++).
- Removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.
- No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

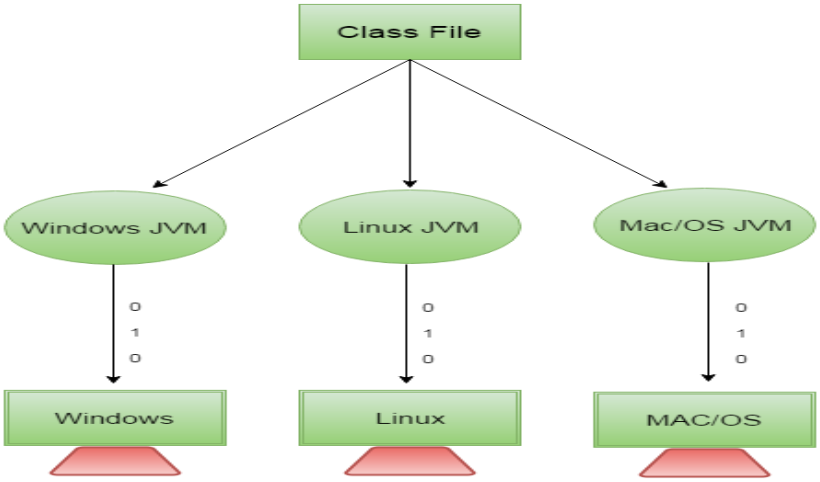
2. Object-oriented

- Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Object-oriented programming (OOPs) is methodologies that simplify software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

3. Platform Independent



A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

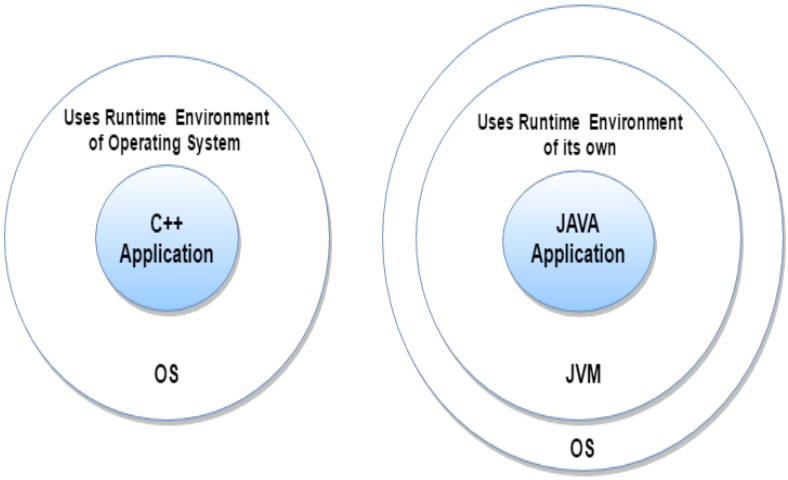
1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac / OS etc. Java code is compiled by the compiler and converted into byte code. This byte code is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere (WORA).

4. Secured

Java is secured because:

- **No explicit pointer**
- **Java Programs run inside virtual machine sandbox**



- **Classloader:** adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** determines what resources a class can access such as reading and writing to the local disk.

These securities are provided by java language. Some security can also be provided by application developer through SSL, JAAS, and Cryptography etc.

5. Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

6. Architecture-neutral

There are no implementation dependent features e.g. size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

7. Portable

We may carry the java bytecode to any platform.

8. High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

9. Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

10. Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

25. a. Illustrate the working of control statements in JAVA with appropriate examples.

Java Loops

Sometimes it is necessary in the program to execute the statement several times, and Java loops execute a block of commands a specified number of times, until a condition is met. In this chapter you will learn about all the looping statements of Java along with their use.

What is Loop?

A computer is the most suitable machine to perform repetitive tasks and can tirelessly do a task tens of thousands of times. Every programming language has the feature to instruct to do such repetitive tasks with the help of certain form of statements. The process of repeatedly executing a collection of statement is called looping. The statements gets executed many number of times based on the condition. But if the condition is given in such a logic that the repetition continues any number of times with no fixed condition to stop looping those statements, then this type of looping is called infinite looping.

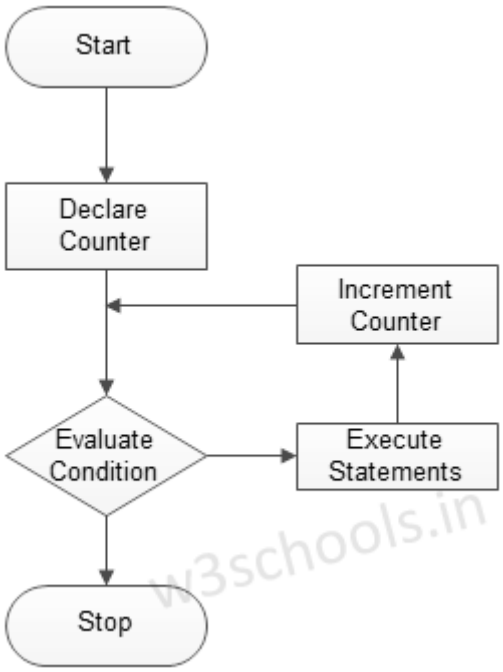
Java supports many looping features which enable programmers to develop concise Java programs with repetitive processes.

Java supports following types of loops:

- while loops
- do while loops
- for loops

All are slightly different and provides loops for different situations.

Figure – Flowchart of Looping:



Java Loop Control Statements

A Loop control statement is used to change normal sequence of execution of loop.

Statement	Syntax	Description
break statement	break;	Is used to terminate loop or switch statements.
continue statement	continue;	Is used to suspend the execution of current loop iteration and transfer control to the loop for the next iteration.
goto statement	goto labelName;labelName: statement;	It's transfer current program execution sequence to some other part of the program.

Java while loops

Java while loops statement allows to repeatedly running the same block of code, until a condition is met.

While loop is most basic loop in Java. It has one control condition, and executes as long the condition is true. The condition of the loop is tested before the body of the loop is executed, hence it is called an entry-controlled loop.

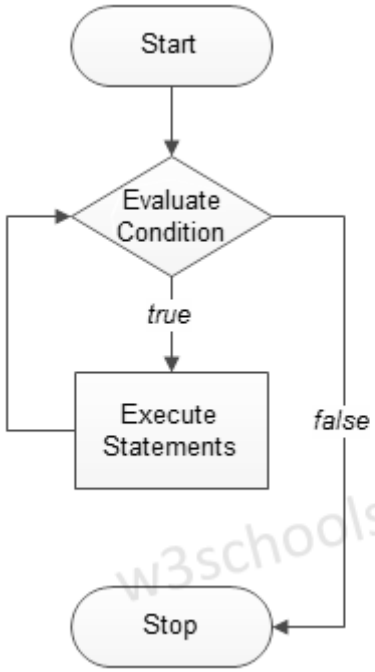
The basic format of while loop statement is:

Syntax:

```
While (condition)
{
  statement(s);
incrementation;
```

}

Figure – Flowchart of while loop:



Example of a Java Program to Demonstrate while loop

Example:

```
public class Sample {

   public static void main(String args[]) {
      /* local variable Initialization */
      int n = 1, times = 5;
      /* while loops execution */
      while (n <= times) {
         System.out.println("Java while loops:" + n);
         n++;
      }   }}
```

Program Output:

```
run:
Java while loops:1
Java while loops:2
Java while loops:3
Java while loops:4
Java while loops:5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Java do while loops

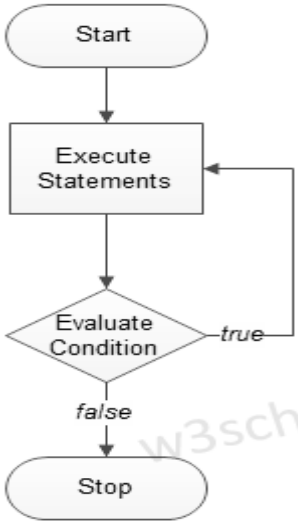
Java do while a loop is very similar to the while loops, but it always executes the code block at least once and further more as long as the condition remains true. This is exit-controlled loop.

The basic format of do while loop statement is:

Syntax:

```
do
{   statement(s);
}while( condition );
```

Figure – Flowchart of do while loop:



Example of a Java Program to Demonstrate do while loop

Example:

```
public class Sample {
   public static void main(String args[]) {
      /* local variable Initialization */
      int n = 1, times = 0;
      /* do-while loops execution */
      do {
         System.out.println("Java do while loops:" + n);
         n++;
      } while (n <= times);    }}
```

Program Output:

```
run:
Java do while loops:1
BUILD SUCCESSFUL (total time: 1 second)
```

Java for loops

Java for loops is very similar to Java while loops in that it continues to process a block of code until a statement becomes false, and everything is defined in a single line.

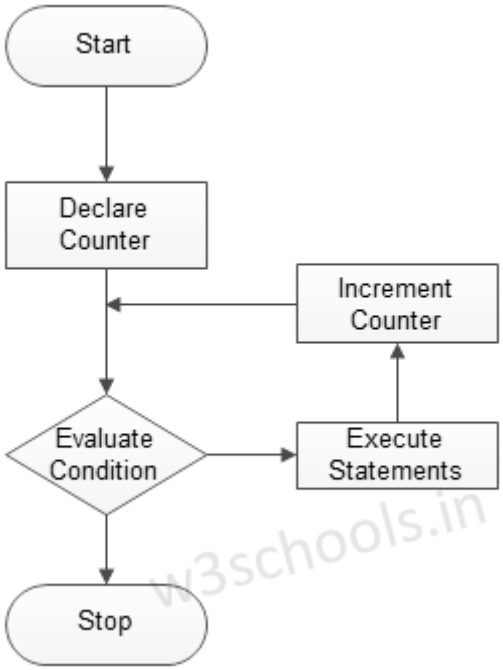
The basic format of for loop statement is:

Syntax:

```
for ( init; condition; increment )
```

```
{
  statement(s);
}
```

Figure – Flowchart of for loop:



Example of a Java Program to Demonstrate for loop

Example:

```
public class Sample {
   public static void main(String args[]) {
      /* local variable Initialization */
      int n = 1, times = 5;
      /* for loops execution */
      for (n = 1; n <= times; n = n + 1) {
         System.out.println("Java for loops:" + n);       }    }}
```

Program Output:

```
run:
Java for loops:1
Java for loops:2
Java for loops:3
Java for loops:4
Java for loops:5
BUILD SUCCESSFUL (total time: 0 seconds)
```

(OR)

b. Write a java program to find the factorial of a given number.

Program:

```
import java.io.*;
class Factorial
{
public static void main(String args[])
{
int i,fact=1;
int number=5;
for(i=1;i<=number;i++)
{
fact=fact*i;
}
System.out.println("Factorial of "+number+" is: "+fact);
}
}
```

26. a. How will you declare and initialize an array? Write a program for single dimensional array by defining the array dynamically.

Program:

```
class array1
{
public static void main(String[] args) throws IOException
{
int num[]=new int [20];
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int i,j,temp;
int x=Integer.parseInt(br.readLine());
System.out.println("\n Enter number:");
for(i=0;i<x;i++)
{
num[i] = Integer.parseInt(br.readLine());
}
for(i=0;i<x;i++)
{
for(j=0;j<i+1;j++)
{
if (num[i] < num[j])
{
temp = num[i];
num[i] = num[j];
num[j]= temp;
}
}
```

```
}
System.out.println(“Sorted array is”);
for(i=0;i<x;i++)
{
System.out.println(“ “+num[i] );
}
}
}
```

(OR)

b. How will you create strings? Explain any four string operations with example.

String Class

String is a sequence of characters, for e.g. “Hello” is a string of 5 characters. In java, string is an immutable object which means it is constant and can cannot be changed once it has been created. In this tutorial we will learn about String class and String methods in detail along with many other Java String tutorials.

Creating a String

There are two ways to create a String in Java

- String literal
- Using new keyword

String literal

In java, Strings can be created like this: Assigning a String literal to a String instance:String str1 = "Welcome";

String str2 = "Welcome";

Using New Keyword

As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this:

String str1 = new String("Welcome");

String str2 = new String("Welcome");

In this case compiler would create two different objects in memory having the same string.

A Simple Java String Example

```
public class Example{
  public static void main(String args[]){
      //creating a string by java string literal
      String str = "Beginnersbook";
      char arrch[]={'h','e','l','l','o'};
      //converting char array arrch[] to string str2
      String str2 = new String(arrch);
      //creating another java string str3 by using new keyword
      String str3 = new String("Java String Example");
      //Displaying all the three strings
      System.out.println(str);
      System.out.println(str2);
      System.out.println(str3);     }}
```

Output:

```
Beginnersbook
hello
```

Creating and using String Objects

String class

It is a predefined class in java.lang package can be used to handle the String. String class is immutable that means whose content cannot be changed at the time of execution of program.

String class object is immutable that means when we create an object of String class it never changes in the existing object.

Example:

```
classStringHandling{
publicstaticvoid main(String arg[]){
```

Output:

```
java
```

```
String s=newString("java");
s.concat("software");
System.out.println(s);}}
```

Explanation: Here we cannot change the object of String class so output is only java not java software.

Manipulating String

1. length()

length(): This method is used to get the number of character of any string.

Example

```
classStringHandling        {                                    Output
publicstaticvoid main(String arg[])      {                      Length: 4
int l;
String s=newString("Java");
l=s.length();
System.out.println("Length: "+l);}}
```

2. charAt(index)

charAt(): This method is used to get the character at a given index value.

Example

```
classStringHandling{                                            Output
publicstaticvoid main(String arg[]){                            Character: v
char c;
String s=newString("Java");
c=s.charAt(2);
System.out.println("Character: "+c);}}
```

3. toUpperCase()

toUpperCase(): This method is use to convert lower case string into upper case.

Example ***Output***

```
classStringHandling{                                            String: JAVA
publicstaticvoid main(String arg[]){
String s="Java";
System.out.println("String: "+s.toUpperCase());}}
```

4. toLowerCase()

toLowerCase(): This method is used to convert lower case string into upper case.

Example ***Output***

```
classStringHandling{
publicstaticvoid main(String arg[]){
String s="JAVA";
System.out.println("String: "+s.toLowerCase());}}
```

String: java

5. concat()

concat(): This method is used to combined two string.

Example ***Output***

```
classStringHandling{
publicstaticvoid main(String arg[]){
String s1="Hitesh";
String s2="Raddy";
System.out.println("Combined String: "+s1.concat(s2));}}
```

Combined String:
HiteshRaddy

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021

INFORMATION TECHNOLOGY/COMPUTER TECHNOLOGY/

COMPUTER SCIENCE/COMPUTER APPLICATIONS

Second Semester
INTERNAL EXAMINATION – February 2019

PROGRAMMING IN JAVA

Class & Section: I B.Sc (IT), CT, CS & BCA **Duration: 2 hours**
Date & Session : 05.02.2019 & FN **Maximum marks: 50 marks**
Subject Code: 18ITU201/18CTU201/18CSU201/18CAU201

PART- A (20 * 1= 20 Marks)
Answer ALL the Questions

1. The file produced by java compiler ends with _____ file extension.
   a. Java c. html
   b. class d. applet
2. The _____ method terminates the program.
   a. System.terminate(0); c. System.halt(0);
   b. **System.exit(0);** d. System.stop(0);
3. _____ is used inside of any method to refer to the current object.
   a. new c. dot
   b. this d. super
4. The method provided by the Writer class is _____.
   a. read() **b. flush()** c. reset() d. skip()
5. The method provided by the Reader class is _____
   a. skip b. write c. flush d. writeX
6. The _____ class defines byte output streams that are connected to files.
   a. InputStream c. OutputStream
   b. **FileInputStream** d.FileOutputStream
7. _____ class in java does not specify how information is retrieved from or stored in files.
   a. Stream **b.file** c. String d. array
8. _____ support 16_bit Unicode character input and output.
   a. ByteStreams b. InputStream
   b. c. OutputStream **d. Character streams**
9. _____ is an explicit specification of a set of methods.
   a. **Interface** b. package c. Statement d. None
10. _____ are containers for classes that are used to keep the class namespace compartmentalized.
    a. Interface **b. package** c. Statement d. None
11. _____ are the means of encapsulation and containing the namespace and scope of variables and methods.
    a. Class b. Package **c.Classes and Packages** d. None

12. _____ is used to extend a class by creating a new class.
    a. Constructors b. Method Overloading **c. Inheritance** d.None
13. The _____ operator creates single instances of a named class and returns a reference to that object.
    a. Dot **b.new** c.super d.this
14. Methods are called on an instance of a class using the _____ operator.
    a. Dot **b.new** c.super d.this
15. _____ initializes an object.
    a. Overloading **b. Constructors** c. Overriding d. none
16. A _____ statement causes control to be transferred directly to the conditional expression that controls the loop.
    a. **Continue** b. return c. jump d. goto
17. Static methods will not refer the _____.
    a. **This** b. dot c. new d.public
18. _____ is used to allocate memory in the constructor.
    a. Delete b. binding c. free **d. new**
19. Subclass constructors can call superclass constructors via the _____ keyword.
    a. Final b. protected c. inherit **d. super**
20. Constructors are invoked automatically when the _____ are created.
    a. Data b. classes **c. objects** d.none of the above

PART- B (3 * 2= 6 Marks)
Answer ALL the Questions

21. What are the Access Modifiers in Java?

Java provides a number of access modifiers to set access levels for classes, variables, methods, and constructors. The four access levels are −

- Visible to the package, the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

22. Define Enumerations.

Enumerations serve the purpose of representing a group of named constants in a programming language. ... The main objective of enum is to define our own data types(Enumerated Data Types). Declaration of enum in java : Enum declaration can be done outside a Class or inside a Class but not inside a Method.

23. Differentiate Method overloading and Method Overriding.

No.	Method Overloading	Method Overriding
1)	Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2)	Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3)	In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
4)	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
5)	In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

PART C (3 * 8 = 24 Marks)
Answer ALL the Questions

24. a. Illustrate Method Overloading with suitable program.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading increases the readability of the program.

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Output:

22

33

(OR)

b. Describe Class constructors with an example.

A constructor is a special method that is used to initialize a newly created object and is called just after the memory is allocated for the object. It can be used to initialize the objects to desired values or default values at the time of object creation. It is not mandatory for the coder to write a constructor for a class.

If no user-defined constructor is provided for a class, compiler initializes member variables to its default values.

Rules for creating a Java Constructor

It has the same name as the class

It should not return a value not even void

Example 1: Create your First Constructor Java

Step 1) Type following code in your editor.

```
class Demo{
     int  value1;
     int  value2;
     Demo(){
        value1 = 10;
        value2 = 20;
        System.out.println("Inside Constructor");
     }

    public void display(){
       System.out.println("Value1 === "+value1);
       System.out.println("Value2 === "+value2);
    }

  public static void main(String args[]){
      Demo d1 = new Demo();
     d1.display();
  }
}
```

Step 2) Save , Run & Compile the code. Observe the output.

Output:

```
Inside Constructor
Value1 === 10
```

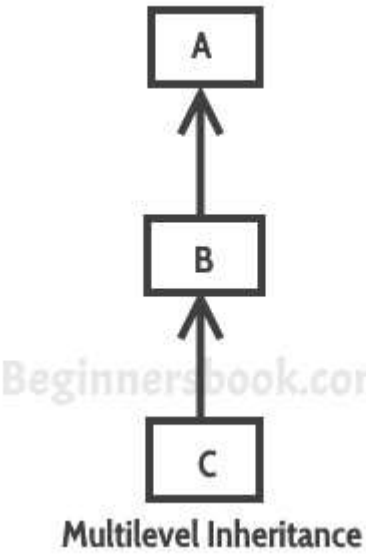
Value2 === 20

- numeric data types are set to 0
- char data types are set to null character('\0')

25. a. Explain Multilevel Inheritance with an example program.

When a class extends a class, which extends anther class then this is called **multilevel inheritance**. For example class C extends class B and class B extends class A then this type of inheritance is known as multilevel inheritance.

Lets see this in a diagram:



It's pretty clear with the diagram that in Multilevel inheritance there is a concept of grand parent class. If we take the example of this diagram, then class C inherits class B and class B inherits class A which means B is a parent class of C and A is a parent class of B. So in this case class C is implicitly inheriting the properties and methods of class A along with class B that's what is called multilevel inheritance.

```
class Car{
  public Car()
  {
    System.out.println("Class Car");
  }
  public void vehicleType()
  {
```

```
    System.out.println("Vehicle Type: Car");
  }
}
class Maruti extends Car{
  public Maruti()
  {
    System.out.println("Class Maruti");
  }
  public void brand()
  {
    System.out.println("Brand: Maruti");
  }
  public void speed()
  {
    System.out.println("Max: 90Kmph");
  }
}
public class Maruti800 extends Maruti{

  public Maruti800()
  {
    System.out.println("Maruti Model: 800");
  }
  public void speed()
  {
    System.out.println("Max: 80Kmph");
  }
  public static void main(String args[])
  {
     Maruti800 obj=new Maruti800();
     obj.vehicleType();
     obj.brand();
```

```
        obj.speed();
    }
}
```

Output:

Class Car

Class Maruti

Maruti Model: 800

Vehicle Type: Car

Brand: Maruti

Max: 80Kmph

(OR)

b. How will you declare a package and import it, Explain?

A package as the name suggests is a pack(group) of classes, interfaces and other packages. In java we use packages to organize our classes and interfaces. We have two types of packages in Java: built-in packages and the packages we can create (also known as user defined package). In this guide we will learn what are packages, what are user-defined packages in java and how to use them.

In java we have several built-in packages, for example when we need user input, we import a package like this:

```
import java.util.Scanner
```

Here:

→ java is a top level package

→ util is a sub package

→ and Scanner is a class which is present in the sub package util.

Types of packages in Java

As mentioned in the beginning of this guide that we have two types of packages in java.

1) User defined package: The package we create is called user-defined package.

2) Built-in package: The already defined package like java.io.*, java.lang.* etc are known as built-in packages.

We have already discussed built-in packages, lets discuss user-defined packages with the help of examples.

Example 1: Java packages
I have created a class Calculator inside a package name letmecalculate. To create a class inside a package, declare the package name in the first statement in your program. A class can have only one package declaration.
Calculator.java file created inside a package letmecalculate

```
package letmecalculate;

public class Calculator {
  public int add(int a, int b){
        return a+b;
  }
  public static void main(String args[]){
        Calculator obj = new Calculator();
        System.out.println(obj.add(10, 20));
  }
}
```

Now lets see how to use this package in another program.

```
import letmecalculate.Calculator;
public class Demo{
  public static void main(String args[]){
        Calculator obj = new Calculator();
        System.out.println(obj.add(100, 200));
  }
}
```

To use the class Calculator, I have imported the package letmecalculate. In the above program I have imported the package as letmecalculate.Calculator, this only imports the Calculator class. However if you have several classes inside package letmecalculate then you can import the package like this, to use all the classes of this package.

import letmecalculate.*;

26. a. Define interface. How will you implement an interface and explain it with an example?

Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).

Interfaces specify what a class must do and not how. It is the blueprint of the class. An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.

If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.

A Java library example is, Comparator Interface. If a class implements this interface, then it can be used to sort a collection. interface <interface_name> {

```
   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

To declare an interface, use interface keyword. It is used to provide total abstraction. That means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default. A class that implement interface must implement all the methods declared in the interface. To implement interface use implements keyword.

Why do we use interface ?

It is used to achieve total abstraction.
Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .
It is also used to achieve loose coupling.
Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes? // Java program to demonstrate working of
// interface.

```
import java.io.*;

// A simple interface
interface in1
{
    // public, static and final
    final int a = 10;

    // public and abstract
    void display();
}

// A class that implements interface.
class testClass implements in1
{
    // Implementing the capabilities of
    // interface.
    public void display()
    {
            System.out.println("Geek");
    }

    // Driver Code
```

```
        public static void main (String[] args)
        {
                testClass t = new testClass();
                t.display();
                System.out.println(a);
        }
    }
```

(OR)

b. Write a java program to demonstrate the concept of boxing and unboxing.

```
// Java program to illustrate the concept
// of Autoboxing and Unboxing
import java.io.*;

class GFG
{
        public static void main (String[] args)
        {
                // creating an Integer Object
                // with value 10.
                Integer i = new Integer(10);

                // unboxing the Object
                int i1 = i;

                System.out.println("Value of i: " + i);
                System.out.println("Value of i1: " + i1);

                //Autoboxing of char
                Character gfg = 'a';

                // Auto-unboxing of Character
                char ch = gfg;
                System.out.println("Value of ch: " + ch);
                System.out.println("Value of gfg: " + gfg);

        }
}

/* Java program to illustrate autoboxing */
import java.io.*;
import java.util.*;

class GFG
{
```

```
        public static void main (String[] args)
        {
                /* Here we are creating a list
                of elements of Integer type.
                adding the int primitives type values */
                List<Integer> list = new ArrayList<Integer>();
                for (int i = 0; i < 10; i++)
                        list.add(i);
        }
}
```

Primitive type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double