KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-21

LECTURE PLAN
COMPUTER APPLICATIONS

SUBJECT NAME: ADVANCED JAVA AND SPRINGS SUBJECT CODE: 18CAP303
SEMESTER: III CLASS: II MCA

Scope: The course covers both core and advanced Java concepts like Database connectivity, Threads, Exception Handling, Collections, JSP, Servlets, XMLHandling etc. students will also learn various Java frameworks like Spring.

Course Objective: After the completion of the Advance Java Course, students will be able to:
1. Develop the code with various Java data types, conditions and loops.
2. Implement arrays, functions and string handling techniques.
3. Understand object oriented programming through Java using Classes, Objects and various Java concepts like Abstract, Final etc.
4. Implement multi-threading and exception handling.
5. Write a code in JDBC to communicate with Database.
6. Write code with spring framework components.

UNIT I

Exception Handling, Exception-Handling Fundamentals, Exception Types- checked & unchecked, Uncaught Exceptions, Using try and catch, Multiple catch Clauses, Nested try Statements, throw, throws and finally. Multithreaded Programming: Introduction to Threads, Creating and Running Threads, Volatile Variables, Life Cycle of a Thread, Thread Priorities and Thread Scheduling – Creating and Executing Thread – Thread Synchronization Runnable Interface

UNIT II

Files and Streams: Advanced Input/output Streams, Readers and Writers, Character and Byte Streams, PrintWriter, Reading Text, Scanner Class, Reading and Writing Files, Copying a File, Class File, Creating a Sequential, Access File, Reading Data from a Sequential, Access File, Random-Access Files, Creating/Writing/Reading Random-Access Files, New I/O APIs for the Java Platform.

UNIT III

Framework: Overview, Generics Fundamentals, Autoboxing, The Collection Interfaces-List Interface, Set Interface, SortedSet Interface, NavigableSet Interface, The Collection Classes-ArrayList, LinkedList, HashSet, LinkedHashSet, TreeSet, Accessing a Collection via an Iterator, Enumeration Interface, Vector, HashTable, Properties, StringTokenizer and Date Class. Serialization: Serializable, Externalizable.

UNIT IV

Introducing the Spring Framework, Spring Framework RunTime & architecture, Inversion of Control (IoC), Dependency Injection, Different Forms of Dependency Injection, Dependency Injection variants, DI classes & Parameter in Spring framework, Bean naming, @Autorwired annotation, The Bean Factory, XML Bean Configuration, Managing the Bean Lifecycle, Basics of Aspect-Oriented Programming (AOP), AOP concepts - Join point, Pointcut, Advice, Types of advice, @AspectJ support

UNIT V

DAO Support and JDBC Framework, Operations with JdbcTemplate, JdbcTemplate Convenience Methods, Basic Queries Using the JdbcTemplate, Batch Updates, Transaction and Resource Management, Global transaction vs. local transaction, Declarative transaction management, XML-based, Annotation-based, Object/Relational Mapping, Basic O/R Mapping, Object Query Languages, Data Access Objects, Setup in a Spring Context, Introduction to Spring MVC, DispatchServlet, Context configuration, Identify the design goals and core concepts of Spring MVC, Spring MVC controllers & Views

SUGGESTED READINGS

1. Deitel & Deitel. (2014), Java How to Program, 10th Edition, Pearson Education Asia, New Delhi.
2. Craig Walls (2014), Spring in Action, 4th Edition
3. Herbert Schildt (2014), "Java Complete Reference", 9th edition. Tata McGraw Hill, New Delhi.
4. Balagurusamy.E (2012), "Programming with Java", 3rd edition ,Tata Mc-Graw Hill, New Delhi.
5. ISRD Group (2012), "Introduction to Object Oriented Programming through Java", 1st Edition, Tata Mc- Graw Hill, New Delhi.
6. Aaron walsh, Justin couch & Daniel H.Steinberg. (2000)," Java 2 Programming", IDG Books India (P) Ltd., New Delhi.
7. Rod Johnson, Jurgen Holler & Alef Arendsen. Professional Java Development with the Spring Framework

WEB SITES

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

CIA	Max.Marks(50)
Part A	Objective type questions - 20 x 1 = 20 Marks
Part B	Answer all the questions - 3 x 2 = 6 Marks
Part C	Answer all the questions Either/Or - 3 x 8 = 24 Marks

ESE	Max.Marks(60)
Part A	Objective type questions -20 x 1 = 20 Marks
Part B	Answer all the questions Either/Or -5 x 6 = 30 Marks
Part C	Answer all the questions Compulsory-1 x 10 = 10Marks

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

Coimbatore-21

Department of Computer Applications

Course Code: 18CAP303 **Course Name**: Advanced Java and Springs

LECTURE PLAN

UNIT-I

Sl. No.	Lecture Duration (Hr)	Topics to be Covered	Support Materials
1	1	Exception Handling Fundamentals, Exception Types-Checked &unchecked	R1-P(120-122)
2	1	Uncaught Exceptions using try & catch, Multiple catch classes	J1, R1-P(123-128)
3	1	Nested try Statements , throw, throws and finally	R1-P(128-133)
4	1	Introduction to threads, Creating and Running Threads, Volatile Variables	R1-P(138-146)
5	1	Life Cycle of a Thread , Thread priorities and Thread Scheduling	R1-P(146-149)
6	1	Creating and Executing Thread	R1-P(144-146)
7	1	Thread Synchronization	W1,R1-P(149-152)
8	1	Runnable Interface	W1,R1-P(149-152)
9	1	Recapitulation and discussion on important questions	
Total no. of Hours planned for Unit-I - 9			

TextBooks: R1:ISRD Group(2012),"Introduction to Object Oriented Programming through Java" ,1st Edition,Tata Mc-Graw Hill, New Delhi.

Websites: W1-www.javatpoint.com.

Journals:J1-International Journal of Computer science and information technology and security.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

Coimbatore-21

Department of Computer Applications

Course Code: 18CAP303 **Course Name**: Advanced Java and Springs

LECTURE PLAN

UNIT-II

Sl. No.	Lecture Duration(Hr)	Topics to be covered	Support Materials
1.	1	Advanced Input/output streams, Readers and writers	R1-P(258-269)
2.	1	Character and Byte Streams, Print writer	R2-P(838-840)
3.	1	Reading text, Scanner class, Reading and Writing Files	R1-P(259-261)
4.	1	Copying a file, Class file, Creating a sequential Access Files	R1-P261 R2-P(846-857)
5.	1	Reading Data from a Sequential Access File	R2-P(857-870)
6.	1	Random Access Files	R2-P(857-870)
7.	1	Creating/Writing/Reading Random Access Files	R2-P(870-885),J1
8.	1	New I/O APIs for the Java Platform	R2-P(897-903),W1
9.	1	Recapitulation and discussion on important questions	
Total no. of Hours planned for Unit-II - 9			

Textbooks: R1-ISRD Group(2012),"Introduction to Object Oriented Programming through Java ",1st Edition Tata-Mc Graw Hill,New Delhi.

R2-Deitel and Deitel(2014)Java How to program 10th Edition,Pearson Education Asia New Delhi.

Websites: W1-www.javatpoint.com

Journals:J1-International Journal of Computer science and information technology and security.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

Coimbatore-21

Department of Computer Applications

Course Code: 18CAP303 **Course Name**: Advanced Java and Springs

LECTURE PLAN

UNIT-III

Sl. No.	Lecture Duration(Hr)	Topics to be covered	Support Materials
1.	1	Collections-Overview, Generics Fundamentals, Autoboxing	R2-P(1- 42) W2-protechtraining.com W3-tutorialspoint.com W1-javapoint.com
2.	1	Generics Fundamentals, Autoboxing	R2-P(1- 42) W2-protechtraining.com W3-tutorialspoint.com W1-javapoint.com
3	1	Collection interface :List interface, Set interface,	R1-P(232-233) R1-P(224-225)
4.	1	Sorted set Interface, Navigable Set Interface	R1-P(232-233) R1-P(224-225)
5.	1	Collection classes-Array List, Linked List, Hashset ,Linked Hashset, Tree Set	R1-P(225-228) R1-P(233-234) J1
6.	1	Accessing a Collection-an Iterator , Enumeration interface, Vector	R1-P(223-224) R1-P(228-230)
7.	1	Hashtable, Properties, String Tokenizer and Data class	R1-P(237-240) R1-P(246-247),P242
8.	1	Serializable, Externalizable	W3
9.	1	Recapitulation and discussion on important questions	
Total no. of Hours planned for Unit-III - 9			

Textbooks: R1-ISRD Group(2012),"Introduction to Object Oriented Programming through Java ",1st Edition Tata-Mc Graw Hill,New Delhi.

R2-Deitel and Deitel(2014)Java How to program 10th Edition,Pearson Education Asia New Delhi.

Websites: W2-protechtraining.com, W3-tutorialspoint.com

Journals:J1-International Journal of Computer science and information technology and security.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

Coimbatore-21

Department of Computer Applications

Course Code: 18CAP303 **Course Name**: Advanced Java and Springs

LECTURE PLAN

UNIT-IV

Sl No.	Lecture Duration(Hr)	Topics to be covered	Support Materials
1.	1	Introduction to Spring Framework,	R3-P(1-27)
2.	1	Spring Framework RunTime and architecture	R3-P(1-27)
3.	1	Inversion of Control, Dependency injection, Different Forms of Dependency injection	R3-P(39-43)
4.	1	Dependency injection variants, DI Classes and	R3,J2
5.		Parameter in Spring Framework	W3
6.	1	Bean naming@ Autowired annotation, the Bean Factory, XML Bean Configuration	R3-P(47-53)
7.	1	Managing the Bean Lifecycle, Basics of Aspect-Oriented programming(AOP)	R3-P64,P(118-161)
8.	1	AOP Concepts-Join point, Pointcut ,Advice, Types of advice, @ Aspect J Support	R3-P(119-125) W3
9	1	Recapitulation and discussion on important questions	
Total no. of Hours planned forUunit-IV -9			

Textbooks: R3-Rod Johnson ,Jurgen Holler and Alef Arendsen, ”Professional Java Development with the Spring Framework”.

Websites: W3-tutorialspoint.com

Journals:J2-International Journal of Engineering innovative technology

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

Coimbatore-21

Department of Computer Applications

Course Code: 18CAP303 **Course Name**: Advanced Java and Springs

LECTURE PLAN

UNIT-V

SI No.	Lecture Duration(Hr)	Topics to be covered	Support Materials
1.	1	DAO Support and JDBC Framework operations with JdbcTemplate	R3-P(173) P-184
2.	1	JdbcTemplate Convenience Methods	R3-P(173) P-184
3.	1	Basic Queries Using the Jdbc Template Batch Updates, Transaction and Resource Management	R3-P(185-186) P(217-253)
4.	1	Global transaction vs local transaction ,	W3
5.		Declarative transaction Management	W3
6.	1	XML-Based, Annotation-based, object/Relational Mapping	R3-P(255-302)
7.	1	Basic O/R Mapping, object query languages	R3-P(256-257)
8.	1	Data Access objects, setup in a Spring context Introduction to Spring MVC DispatchServlet Context configuration, Identify the design goals and core concept of Spring MVC, Spring MVC Controls and views	J2,R3-P(270-273) R3-P431 W3
9.	1	Discussion of previous ESE question papers	
10	1	Discussion of previous ESE question papers	
11.	1	Discussion of previous ESE question papers	
12.	1	Recapitulation and discussion on important questions	
Total no. of Hours planned for Unit-V -12			

Textbooks: R3-Rod Johnson ,Jurgen Holler and Alef Arendsen, ”Professional Java Development with the Spring Framework”.

Websites: W3-tutorialspoint.com

Journals:J2-International Journal of Engineering innovative technology.

UNIT: I

SYLLABUS

Exception Handling, Exception-Handling Fundamentals, Exception Types- checked & unchecked, Uncaught Exceptions, Using try and catch, Multiple catch Clauses, Nested try Statements, throw, throws and finally. Multithreaded Programming: Introduction to Threads, Creating and Running Threads, Volatile Variables, Life Cycle of a Thread, Thread Priorities and Thread Scheduling – Creating and Executing Thread – Thread Synchronization Runnable Interface

Exceptional handling:

Exception handling is a mechanism to handle runtime errors, so that normal flow of the program can be maintained.

Exception Hierarchy:

Throwable is the super class.

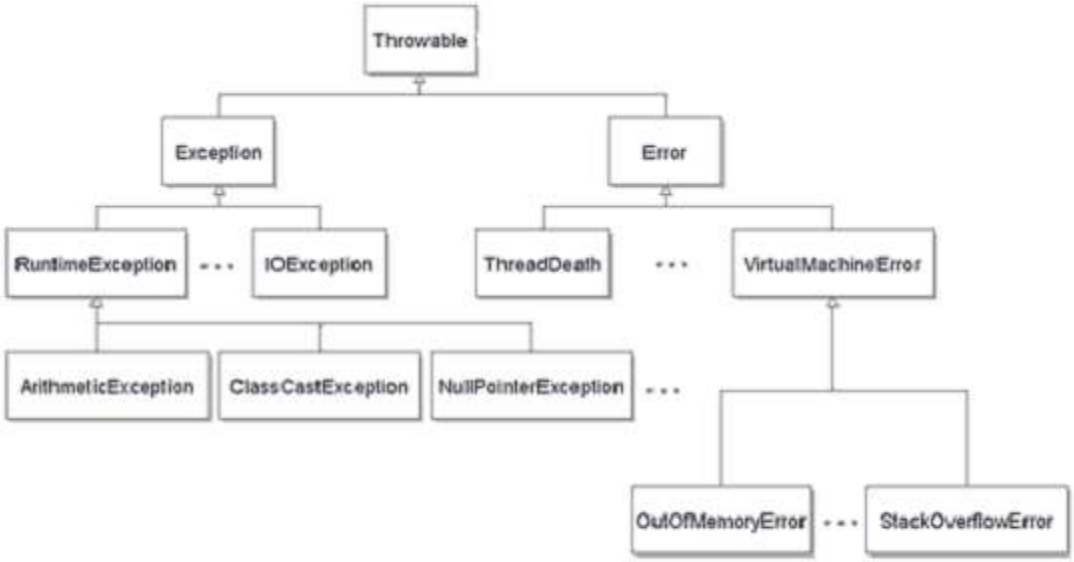


Figure by javawithease

Advantages/Benefits of exceptional handling:

1. Using exceptional handling we can separate the error handling code from normal code.

2. Using exceptional handling we can differentiate the error types.

3. Normal flow of program can be maintained.

Types of Exception:

1. Checked exception.
2. Unchecked exception.
3. Error.

Checked exceptions:

Checked exceptions are those exceptional conditions that are checked by compiler at the compile time. A checked exception forces you to either use try-catch or throws. All exceptions except Error, RuntimeException, and their subclasses are checked exceptions.

e.g. – IOException, SQLException etc.

Unchecked exceptions:

Unchecked exceptions are those exceptional conditions that are not checked by compiler at the compile time. Unchecked exceptions are checked at runtime. An unchecked exception not forces you to either use try-catch or throws. RuntimeException and their subclasses are unchecked exceptions. This Exception can be avoided by programmer.

e.g. – NullPointerException, ArithmeticException etc.

Error:

Errors are those exceptional conditions that are not checked by compiler at the compile time. Errors are checked at runtime. An error not forces you to either use try-catch or throws. Error and their subclasses are represents errors. Error can't be avoided by programmer, it is irrecoverable.

e.g. – OutOfMemoryError etc.

Exception -handling fundamentals :

A java exception is an object that describes an exceptional condition that has occurred in the piece of the code> When an exceptional condition arises an object representing the exception is created and thrown in the method that caused the error. that method may choose to handle the exception itself or pass it on. Either way at some point the exception is caught and processed.

Exception can be generated by the java run time system or they can be manually generated by the code. Exception thrown by the java relate to the fundamental errors that violate the rules of the java languages or the constraints of the java execution environment. Manually generated exception is typically used to report some error conditions to the caller of the method.

Java exception handling is managed via five keywords: try, catch, throw, throws and finally. Briefly here is how it works. Program statements that you want to monitor for the exception are contained within the try blocks. if an exception is occurred within the try block, it is thrown. Your code can catch this exception using the catch and handle it in some rational manner. System generated exception are automatically thrown by the java run time system. To manually throw an exception use the keyword throw. An Exception that is thrown out of a method should must be specified as such by the throw clause. Any code that absolutely must be executed after the try block completes is put in a finally block.

This is the general form of the exception handling block:

```
try{
// block of the code to monitor for the errors
}
catch(Exception Type1 exOb){
Exception handler for the Exception Type1
}
catch (Exception Type2 exOb)
//Exception handeler for the Exception Type2
}
//…
finally{
// block of the code to be executed after the try blocks end
}
```

Multithreading :

Multithreading is a conceptual programming concept where a program (process) is divided into two or more subprograms (process), which can be implemented at the same time in parallel.Amultithreadedprogramcontainstwoormorepartsthatcanrunconcurrently.Each

partofsuchaprogramiscalledathread,andeachthreaddefinesaseparatepathofexecution.

A *process* consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of aprocess.

There are two distinct types of Multitasking i.e. Processor-Based and Thread-Based multitasking.

- Processes are heavyweight tasks where threads arelightweight
- Processes require their own separate address space where threads share the address space
- Interprocess communication is expensive and limited where Interthread communication is inexpensive, and context switching from one thread to the next is lower incost.

Benefits of Multithreading

1. Enables programmers to do multiple things at onetime
2. Programmers can divide a long program into threads and execute them in parallel which eventually increases the speed of the programexecution
3. Improved performance andconcurrency
4. Simultaneous access to multipleapplications

Life Cycle of Thread

A thread can be in any of the five following states

1. Newborn State: When a thread object is created a new thread is born and said to be in Newbornstate.

2. Runnable State: If a thread is in this state it means that the thread is ready for executionandwaitingfortheavailabilityoftheprocessor.Ifallthreadsinqueueareo f same priority then they are given time slots for execution in round robinfashion

3. Running State: It means that the processor has given its time to the thread for execution. A thread keeps running until the following conditionsoccurs

    a. Thread give up its control on its own and it can happen in the following situations

        i. A thread gets suspended using suspend() method which can only be revived with resume()method

        ii. A thread is made to sleep for a specified period of time using sleep(time) method, where time inmilliseconds

        iii. Athreadismadetowaitforsomeeventtooccurusingwait()method. In this case a thread can be scheduled to run again using notify () method.

    b. A thread is pre-empted by a higher prioritythread

4. Blocked State: If a thread is prevented from entering into runnable state and subsequently running state, then a thread is said to be in Blockedstate.

5. DeadState:ArunnablethreadenterstheDeadorterminatedstatewhenitcompletes

its task or otherwiseterminates.

Create Thread by Implementing Runnable

The easiest way to create a thread is to create a class that implements the Runnable interface.

ToimplementRunnable,aclassneedonlyimplementasinglemethodcalledrun(),whichis declared likethis:

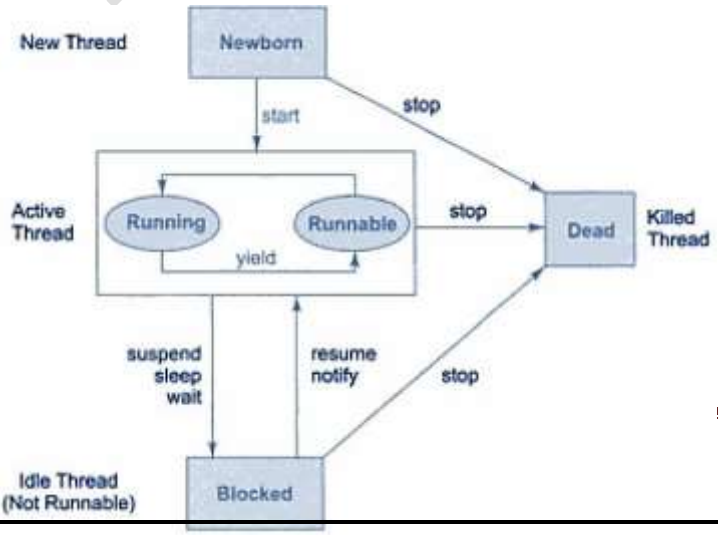
You will define the code that constitutes the new thread inside run() method. It is important

tounderstandthatrun()cancallothermethods,useotherclasses,anddeclarevariables,just like the main threadcan.

After you create a class that implements Runnable, you will instantiate an object of type

Threadfromwithinthatclass.Threaddefinesseveralconstructors.Theonethatwewillusei s shownhere:

```
Thread(Runnable threadOb, String threadName);
```



HerethreadObisaninstanceofaclassthatimplementstheRunnableinterfaceandthename of the new thread is specified by threadName. After the new thread is created, it will not start running until you call its start( ) method, which is declared within Thread. The start( ) method is shownhere:

```
void start( );
```

Example to Create a Thread using Runnable Interface

```
class t1 implements Runnable
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        t1 obj1 = new t1();
        Thread t = new Thread(obj1);
        t.start();
    }
}
```

Output:

```
C:\NIEC Java>javac t1.java

C:\NIEC Java>java t1
Thread is Running

C:\NIEC Java>
```

Create Thread by Extending Thread

The second way to create a thread is to create a new class that extends Thread, and then to create an instance of that class. The extending class must override the run( ) method,

which is the entry point for the new thread. It must also call start( ) to begin execution of the new thread.

Example to Create a Thread by Extending Thread Class

```
class t2 extends Thread
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        t2 obj1 = new t2();
        obj1.start();
    }
}
```

Output:

```
C:\NIEC Java>javac t2.java

C:\NIEC Java>java t2
Thread is Running

C:\NIEC Java>
```

Thread Methods

SN	Methods with Description
1	public void start() Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.

2	public void run() If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	public final void setName(String name) Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	public final void setPriority(int priority) Sets the priority of this Thread object. The possible values are between 1 and 10.
5	public final void setDaemon(boolean on) A parameter of true denotes this Thread as a daemon thread.
6	public final void join(long millisec) The current thread invokes this method on a second thread, causing the current thread

	to block until the second thread terminates or the specified number of milliseconds passes.
7	public void interrupt() Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	public final boolean isAlive() Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

Example:

```
class t2 extends Thread
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        t2 obj1 = new t2();
        obj1.start();
        obj1.start();
    }
}
```

Asyoucanseetwostatementstostart a same thread is written in the code which will not give error during compilation but when you run it you can see an Exception as shown in the OutputScreenshot

Output:

```
C:\NIEC Java>javac t2.java

C:\NIEC Java>java t2
Thread is Running
Exception in thread "main" java.lang.IllegalThreadStateException
        at java.lang.Thread.start(Unknown Source)
        at t2.main(t2.java:11)
```

Use of Yield() Method

Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled

Example

```
class A extends Thread
{
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            if(i==2) yield();
            System.out.println("A:" +i);
        }
        System.out.println("Exit from A");
    }
}
class B extends Thread
{
    public void run()
    {
        for(int j=1; j<=5; j++)
        {
            System.out.println("B:" +j);
        }
        System.out.println("Exit from B");
    }
}
class yieldtest
{
    public static void main(String args[])
    {
        A a = new A();
        B b = new B();
        a.start();
        b.start();
    }
}
```

Condition is checked and when i==2 yield() method is evoked taking control to thread B

As you can see in the output below, thread A gets started and when condition if(i==2) gets satisfied yield() method gets evoked and the control is relinquished from thread AtothreadBwhichruntoitscompletionandonlyafterthatthreadaregainthecontrol back

Output

```
C:\NIEC Java>javac yieldtest.java

C:\NIEC Java>java yieldtest
A:1
B:1
B:2
B:3
B:4
B:5
Exit from B
A:2
A:3
A:4
A:5
Exit from A
```

Use of stop() Method

The stop() method kills the thread on execution

Example

```
class C extends Thread
{
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            if(i==2) stop();
            System.out.println("A:" +i);
        }
        System.out.println("Exit from A");
    }
    public static void main(String args[])
    {
        C c = new C();
        c.start();
    }
}
```

Condition is checked and when i==2 stop() method is evoked causing termination of thread execution

Output

```
C:\NIEC Java>java C
A:1
```

Use of sleep() Method

Causes the currently running thread to block for at least the specified number of milliseconds. You need to handle exception while using sleep() method.Example

```
class C extends Thread
{
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            try
            {
                if(i==2) sleep(1000);
            }
            catch(Exception e)
            {
            }
            System.out.println("A:" +i);
        }
        System.out.println("Exit from A");
    }
    public static void main(String args[])
    {
        C c = new C();
        c.start();
    }
}
```

Condition is checked and when i==2 sleep() method is evoked which halts the execution of the thread for 1000 milliseconds. When you see output there is no change but there is delay in

Output

```
C:\NIEC Java>javac C.java

C:\NIEC Java>java C
A:1
A:2
A:3
A:4
A:5
Exit from A
```

Use of suspend() and resume() method

A suspended thread can be revived by using the resume() method. This approach is useful when we want to suspend a thread for some time due to certain reason but do not want to kill it.

```
class C extends Thread
{
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            System.out.println("C:" +i);
        }
        System.out.println("Exit from C");
    }
}
class A extends Thread
{
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            System.out.println("A:" +i);
        }
        System.out.println("Exit from A");
    }
}
class suspendtest
{
    public static void main(String args[])
    {
        C c = new C();
        A a = new A();
        c.start();
        a.start();
        c.suspend();
        c.resume();
    }
}
```

Although Thread 'C' is started earlier than Thread 'A' but due to suspend method Thread 'A' gets completed ahead of Thread 'C'

Output

```
C:\Achin Jain>java suspendtest
A:1
A:2
A:3
A:4
A:5
Exit from A
C:1
C:2
C:3
C:4
C:5
Exit from C
```

is the example in which two threads C and A are created. Thread C is started ahead

```
class C extends Thread
{
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            System.out.println("C:" +i);
        }
        System.out.println("Exit from C");
    }
}
class A extends Thread
{
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            System.out.println("A:" +i);
        }
        System.out.println("Exit from A");
    }
}
class suspendtest
{
    public static void main(String args[])
    {
        C c = new C();
        A a = new A();
        c.start();
        a.start();
        c.suspend();
        c.resume();
    }
}
```

Although Thread 'C' is started earlier than Thread 'A' but due to suspend method Thread 'A' gets completed ahead of Thread 'C'

Output

```
C:\Achin Jain>java suspendtest
A:1
A:2
A:3
A:4
A:5
Exit from A
C:1
C:2
C:3
C:4
C:5
Exit from C
```

of Thread A, but C is suspended using suspend() method causing hread A to get hold of

the processor allowing it to run and when Thread C is resumed using resume() method it runs to its completion

POSSIBLE QUESTIONS

PART-A (20 Marks)

(Q.No 1 to 20 Online Examination)

PART-B (6 Marks)

1. Discuss Thread Priorities and Thread Scheduling
2. List out the various Exceptions in java.
3. Explain Life Cycle of Thread and Runnable Interface
4. Write a java program to handle the Arithmetic Exception.
5. Explain why exception handling is an effective means for dealing with constructor failure.
6. Name three threads that are created automatically by the java virtual machine and discuss the purpose of each thread
7. Discuss the Exception types and Uncaught Exceptions
8. Explain Thread Synchronization with suitable examples
9. Discuss how to create and run a Thread
10. Explain the following i) try..catch block ii) throw iii) throws iv)finally

PART – C (10 marks)

1.What is synchronization and why is it important

QUESTION	OPT 1	OPT 2	OPT 3	OPT 4	OPT 4	OPT 5	ANSWER
_____keyword is used to specify the exception thrown by method.	catch	throws	finally	throw			throws
_____blocks execute compulsorily whether exception is catch or not.	finally	catch	throw	throws			throws
_____exception is thrown when divide by zero statement.	NumberFormatException	ArithmeticException	NullPointerException	Exception			ArithmeticException
_____exception in java arises in code sequence.	RunTime	CompilationTime	can occur any time	does not occur			RunTime
_____keywords is not a part of exception handling.	try	finally	thrown	catch			thrown
Super class of all exceptional type classes is_____	string	RunTimeException	throwable	catchable			throwable
_____class is related to all the exceptional that can be catch by using catch.	Error	Exception	RunTimeException	CompileTime			Exception
_____keyword is used to manually throw an exception.	try	catch	throw	finally			throw
Operator used to generate an instance of an exception than can be thrown by using throw is_____	new	malloc	alloc	thrown			new
_____handles the exception when no catch is used.	default handler	finally	throw handler	final			default handler
_____JVM runs out of memory which exception will be thrown.	Memory Bound Exception	Out of memory error	Out of range exception	NullReference Exception.			Out of memory error
In java_____ programming environment,the throw keyword is used.	to generate exception programatically	to throw exception object	to catch exception object	to terminate exception			to generate exception programatically
Exception _____is thrown by read() method.	Exception	file not found	ReadException	IOException			IOException
Exception and error are immediate sub classes of a class called _____	object	Throwable	AWT	panel			Throwable
_____does not deal with exception.	throws	throw	finallize	finally			finallize
URL throws an exception called_____	illegalURLException	URLException	malformedhostException	malformedURLException			malformedURLException
_____class is base class for all exception.	string	error	throwable	RunTimeException			throwable
A program can be _____	single threaded and multi threaded	finally	try	catch			single threaded and multi threaded
When the event for _____a thread is blocked.	thread moves to the ready queue	thread blocked	ready	thread completes			thread moves to the ready queue
Termination of the process terminates when_____	first thread of the process	first two threads of the process	all threads within the process	no threads within the process			all threads within the process
_____is not a valid state of a thread.	running	parsing	ready	blocked			parsing
The register context and stacks of a thread are deallocated when the thread is___	terminates	blocks	unblocks	spawns			terminates
_____of predefined class thread is used to check whether current thread being checked is still running.	isAlive()	join()	isRunning()	Alive()			isAlive()
_____are types of multitasking.	Thread based	Process and Thread based	Process based	RunTime			Process and Thread based
_____is not a part of exception handling.	try	catch	finally	thrown			thrown
_____keywords must be used for monitoring the exception.	try	catch	finally	throw			try
_____handles the exception when no catch is used.	default handler	finally	throw handler	java run time			default handler
_____class is related to all the exceptions that cannot be caught.	error	exception	RunTimeException	Exception handling			error
The packages contain all the Java's built in exceptions is_____	java.io	java.util	java.lang	java.net			java.lang
Thread Priority in Java is_____	Integer	Float	Double	Long			Integer
The default priority of a newly created thread is_____	MIN_PRIORITY	MAX_PRIORITY	NORM_PRIORITY	PRIORITY			NORM_PRIORITY
Two threads cannot simultaneously enter into the methods of the same object if the methods are	static	Synchronized	Private	Package			Synchronized
The name of the method which is used to schedule a thread for execution is_____	init()	start()	run()	resume()			run()
The default priority of a thread in Java is_____	10	5	3	1			5
The maximum thread priority in Java is_____	10	12	5	8			10
A thread becomes not runnable when	its sleep method is invoked	program terminates	an event occurs	An event Suspend			its sleep method is invoked
_____ is the static member of thread?	Current Thread()	Join()	getName()	interrupt()			Current Thread()
Select the valid thread state transition.	ready to running	ready to waiting	waiting to running	running to ready			ready to running
The dead thread in Java is_____	Thead waiting	Thread is in sleep	Thread completed its run method	Thread Suspended			Thread completed its run method
_____ are types of multitasking.	Process based	Thread based	Process & Thread based	P-Thread			Process & Thread based
_____packages contain all the Java's built in exceptions.	java.io	java.util	java.lang	java.net			java.lang
The name of the method used to start a thread execution is_____	resume()	run()	start()	init()			start()
The method _____is used to find out that a thread is still running or not.	run()	Alive()	isAlive()	checkRun()			isAlive()
_____ waits for the thread to terminate.	sleep()	join()	isAlive()	stop()			join()
_____is used to explicitly set the priority of a thread.	set()	make()	SetPriority()	run()			SetPriority()
AWT stands for	All Window Tools	Abstract Window Toolkit	All Writing Tools	Absract Writing Toolkit			Abstract Window Toolkit
Thread Priority in Java is_____	Double	long	Integer	Float			Integer
_____will not directly cause a thread to stop?	InputStream	wait()	sleep()	notify()			notify()
_____will directly stop the execution of a thread?	wait()	notifyall()	notify()	sleep()			wait()
_____registers a thread in a thread scheduler.	start()	construct()	register()	run()			start()
_____contain the body of the thread.	stop()	run()	main()	start()			run()
_____class is used to read characters and strings in Java from Console.	Bufferedreader	StringReader	InputStreamReader	BufferedWriter			Bufferedreader
_____ is used to read a string from the input stream.	get()	getline()	read()	readline()			read()
_____ class is used to read from byte array.	InputStream	BufferedInputStream	both a& b	ByteArrayInputStream			ByteArrayInputStream
_____is a type of stream in Java.	Integer stream	Short stream	Byte stream	Long stream			Byte stream
_____ is used to perform all input & output operations in Java.	Streams	Variables	Classes	None			Streams
_____classes are used by character streams for input and output operations.	InputStream	Writer	ReadStream	OutputStream			Writer
_____contain the body of the thread.	stop()	run()	main()	start()			run()

UNIT: II

SYLLABUS

Files and Streams: Advanced Input/output Streams, Readers and Writers, Character and Byte Streams, PrintWriter, Reading Text, Scanner Class, Reading and Writing Files, Copying a File, Class File, Creating a Sequential, Access File, Reading Data from a Sequential, Access File, Random-Access Files, Creating/Writing/Reading Random-Access Files, New I/O APIs for the Java Platform

Java files and streams

The java.io package contains nearly every class you might ever need to perform input and output *I*/*O* in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, Object, localized characters, etc.

Stream

A stream can be defined as a sequence of data. There are two kinds of Streams InPutStream: The InputStream is used to read data from a source. OutPutStream: the OutputStream is used for writing data to a destination. Java provides strong but flexible support for I/O related to Files.

Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are , FileInputStream and FileOutputStream. Following is an example which makes use of these two classes to copy an input file into an output file:

```
im port java.io.* ;
public class CopyFile{
public static void m ain(String args[]) throws IOException
{
FileInputStreamin = null;
FileOutputStreamout = null;
try{
in= new FileInputStream("input.txt");
out= new FileOutputStream("output.txt");
intc;
while((c = in.read()) != -1) {
out.write(c);
}
}finally{
if(in != null) {
in.close();
}
if(out != null) {
out.close();
}
}
}
}
```

Now let's have a file input.txt with the following content:

This is test for copy file.

As a next step, compile above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put above code in CopyFile.java file and do

the following:

$ javacCopyFile.java

$ javaCopyFile

Character Streams

Java Byte streams are used to perform input and output of 8-bit bytes, where as Java Character streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are , FileReader and FileWriter.. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

We can re-write above example which makes use of these two classes to copy an input file *havingunicodecharacters*into an output file:

```
im port java.io.* ;
public class CopyFile{
public static void m ain(String args[]) throws IOException
{
FileReaderin = null;
FileWriterout = null;
try{
in= new FileReader("input.txt");
out= new FileWriter("output.txt");
intc;
```

```
while((c = in.read()) != -1) {
out.write(c);
}
}finally{
if(in != null) {
in.close();
}
if(out != null) {
out.close();
}
}
}
```

Now let's have a file input.txt with the following content:

This is test for copy file.

As a next step, compile above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put above code in CopyFile.java file and do

the following:

$ javacCopyFile.java

$ javaCopyFile

Standard Streams

All the programming languages provide support for standard I/O where user's program can take input from a keyboard and then produce output on the computer screen. If you are aware if C or C++ programming languages, then you must be aware of three standard devices STDIN, STDOUT and STDERR. Similar way Java provides following three standard streams Standard Input: This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as System.in.

Standard Output:

This is used to output the data produced by the user's program and usually a computer screen is used to standard output stream and represented as System.out.

Standard Error:

This is used to output the error data produced by the user's program and usually a computer screen is used to standard error stream and represented as System.err. Following is a simple program which creates InputStreamReaderto read standard input stream
until the user types a "q":

```
im port java.io.* ;
public class ReadConsole{
public static void m ain(String args[]) throws IOException
{
InputStream Reader cin= null;
try{
cin= new InputStream Reader(System .in);
System .out.println("Enter characters, 'q' to quit.");
charc;
do{
c = (char) cin.read();
```

```
System .out.print(c);
} while(c != 'q');
}finally{
if(cin!= null) {
cin.close();
}
}
}
}
```

Let's keep above code in ReadConsole.java file and try to compile and execute it as below. This program continues reading and outputting same character until we press 'q':

```
$ javacReadConsole.java
$ javaReadConsole
Enter characters, 'q' to quit.
1
1
e
e
q
q
```

Reading and Writing Files:

As described earlier, A stream can be defined as a sequence of data. The InputStreamis used to read data from a source and the OutputStreamis used for writing data to a destination. Here is a hierarchy of classes to deal with Input and Output streams. The two important streams are FileInputStreamand FileOutputStream.

FileInputStream

This stream is used for reading data from the files. Objects can be created using the keyword new and there are several types of constructors available. Following constructor takes a file name as a string to create an input stream object to read the file.:

```
InputStreamf = new FileInputStream("C:/java/hello");
```

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File method as follows:

```
File f = new File("C:/java/hello");
InputStreamf = new FileInputStream(f);
```

Once you have *InputStream*object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

SN Methods with Description

1 public void close throws IOException{}

This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.

2 protected void finalizethrowsIOException {}

This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.

3 public intread*intr*throwsIOException{}

This method reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and -1 will be returned if it's end of file.

4 public intread*byte*[]*r* throws IOException{}

This method reads r.length bytes from the input stream into an array. Returns the total number of bytes read. If end of file -1 will be returned.

5 public int available throws IOException{}

Gives the number of bytes that can be read from this file input stream.Returns an int.

There are other important input streams available, for more detail you can refer to the following links:

ByteArrayInputStream

DataInputStream

FileOutputStream:

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output. Here are two constructors which can be used to create a FileOutputStream object. Following constructor takes a file name as a string to create an input stream object to write the file:

```
OutputStreamf = new FileOutputStream("C:/java/hello")
```

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File method as follows:

```
File f = new File("C:/java/hello");
OutputStreamf = new FileOutputStream(f);
```

Once you have *OutputStream*object in hand, then there is a list of helper methods, which can be used to write to stream or to do other operations on the stream.

SN Methods with Description

1 public void close throws IOException{}

This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException

2 protected void finalizethrowsIOException {}

This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.

3 public void write*intw*throwsIOException{} This methods writes the specified byte to the output stream.

4 public void write*byte*[]*w*

Writes w.length bytes from the mentioned byte array to the OutputStream.

There are other important output streams available, for more detail you can refer to the following links:

ByteArrayOutputStream

DataOutputStream

Example:

Following is the example to demonstrate InputStream and OutputStream:

```
im port java.io.* ;
public class fileStream Test{
public static void m ain(String args[]){
try{
byte bWrite[] = {11,21,3,40,5};
OutputStream os= new FileOutputStream("test.txt");
for(int x=0; x <bWrite.length; x++){
os.write(bWrite[x] ); // writes the bytes
}
os.close();
InputStream is = new FileInputStream("test.txt");
int size = is.available();
```

```
for(inti=0; i<size; i++){
System .out.print((char)is.read() + " ");
}
is.close();
}catch(IOExceptione){
System .out.print("Exception");
}
}
}
```

The above code would create file test.txt and would write given numbers in binary format. Same would be output on the stdout screen.

File Navigation and I/O:

There are several other classes that we would be going through to get to know the basics of File Navigation and I/O.

- File Class
- FileReader Class
- FileWriter Class

Directories in Java:

A directory is a File which can contains a list of other files and directories. You use File object to create directories, to list down files available in a directory. For complete detail check a list of all the methods which you can call on File object and what are related to directories.

Creating Directories:

There are two useful File utility methods, which can be used to create directories: The mkdirmethod creates a directory, returning true on success and false on failure. Failure indicates

that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet.

Following example creates "/tmp/user/java/bin" directory:

```
im port java.io.File;
public class CreateDir{
public static void m ain(String args[]) {
String dirnam e = "/tm p/user/java/bin";
File d = new File(dirnam e);
// Create directory now.
d.mkdirs();
}
}
```

Compile and execute above code to create "/tmp/user/java/bin".

Note: Java automatically takes care of path separators on UNIX and Windows as per conventions.

If you use a forward slash / on a Windows version of Java, the path will still resolve correctly.

Listing Directories:

You can use list method provided by File object to list down all the files and directories available in a directory as follows:

```
im port java.io.File;
public class ReadDir{
public static void m ain(String[] args) {
File file= null;
String[] paths;
try{
// create new file object
file= new File("/tm p");
```

```
// array of files and directory
paths= file.list();
// for each nam e in the path array
for(String path:paths)
{
// prints filenam e and directory nam e
System .out.println(path);
}
}catch(Exception e){
// if any error occurs
e.printStackTrace();
}
}
}
```

This would produce following result based on the directories and files available in your /tmp directory:

```
test1.txt
test2.txt
ReadDir.java
ReadDir.class
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

POSSIBLE QUESTIONS

PART-A (20 Marks)

(Q.No 1 to 20 Online Examination)

PART-B (6 Marks)

1.Develop a program in File Class to obtain file and directory information

2. Explain how to read Data from Sequential Access File

3. Discuss how to create / read / write Random Access Files in java.

4. Explain Character and Byte Streams

5. Discuss the methods of Reader and Writer Class

6. How to read data from the console without using InputStreamReader Class.

7. Develop a java program to read and write data from console using Scanner Class

8. Discuss New I/O APIs

9. Develop a java program for PrintWriter Class

10. Discuss how to write data randomly to a Random-Access File

PART-C(10 marks)

1.Develop a java program to read text from the console

KARPAGAM ACADEMY OF HIGHER EDUCATION
COIMBATORE - 21
DEPARTMENT OF COMPUTER APPLICATIONS

CLASS : II MCA **BATCH : 2017-2020**

SUBJECT: ADVANCED JAVA AND SPRING

S.NO	QUESTIONS (UNIT 2)	OPTION A	OPTION B	OPTION C	OPTION D	ANSWER
1	_____ contains the classes can work on character stream	Input Stream	Output stream	character stream	Buffered input Stream	character stream
2	_____ class is used to read characters in a file.	FileReader	FileWriter	FileInputStream	InputStreamReader	FileReader
3	_____method of FileReader class is used to read characters from a file.	read()	scanf()	get()	getInteger()	read()
4	_____class can be used to implement input stream that uses a character array as the source	BufferedReader	FileReader	charArrayReader	FileArrayReader	charArrayReader
5	_____is a method to clear all the data present in output buffers	clear()	flush()	fflush()	close()	flush()
6	_____classes can return more than one character to be returned to input stream	BufferedReader	BufferedWriter	PushbachReader	CharArrayReader	PushbachReader
7	_____class contains the methods used to write in a file	FileStream	FileInputStream	BufferedOutputStream	FileBufferStream	FileInputStream
8	_____exception is thrown in cases when the file specified for writing it not found	IOException	FileException	FileNotFoundException	FileInputException	FileNotFoundException
9	_____methods are used to read in from file	get()	read()	scan()	readFileInput()	read()
10	_____values is returned by read() method is end of file(EOF) is encountered	0	1	-1	NULL	-1
11	_____exception is thrown by close() and read() methods	IOException	FileException	FileNotFoundException	FileInputOutputException	IOException
12	_____ methods is used to write() into a file	put()	putFile()	write()	writeFile()	write()
13	_____ is the value of "d" after this line of code has been executed. Double d=Math.round (2.5+	2	3	4	2.5	3
14	_____is used to compile the program without error	int a=Math.abs(-5);	int b=Math.abs(5.0);	int c=Math.abs(5.5F);	int d=Math.abs(5L);	int a=Math.abs(-5);
15	_____ are valid calls to Math.max 1) Math.max(1,4) 2) Math.max(2.3,5) 3) Math.max(	1,2 and 4	2,3 and 4	1,2 and 3	3 and 4	1,2 and 4
16	_____is superclass of every class in java	String class	Object class	Abstract class	ArrayList class	Object class
17	_____method of Object class can clone an object	Objectcopy()	copy()	Object clone()	clone()	Object clone()
18	_____ method of Object class is used to obtain class of an object at run time	get()	void getclass()	Class getclass()	getInteger()	Class getclass()
19	_____ keywords can be used to prevent inheritance of a class	super	constant	Class	final	final
20	_____keywords cannot be used for a class which has been declared final	abstract	extends	abstract and extends	Object	abstract
21	_____ relies upon its subclasses for complete implementation of its methods	Object class	abstract class	ArrayList class	File class	abstract class
22	_____keywords is used to define packages in java	pkg	Pkg	package	Package	package
23	_____is a mechanism for naming and visibilty control of a class and its content	Object	Packages	Interfaces	class	Packages
24	_____ access specifiers can be used for a class so that it's members can be accessed by a diff	Public	Protected	Private	No Modifier	Public
25	_____is correct way of importing an entire package 'pkg'	import pkg.	Import pkg.	import pkg.*	Import pkg.*	import pkg.*
26	_____package stores all the standard java classes	lang	java	util	java.packages	java
27	_____ is used to access the datbase server at time of executing the program and get the dat	Embedded SQL	Dynamic SQL	SQL declarations	SQL data analysis	Dynamic SQL
28	_____ header must be included in java program to establish database connectivity using JD	Import java.sql.*;	Import java.sql.odbc.jdbc.*;	Import java.jdbc.*;	Import java.spl.jdbc.*;	Import java.sql.*;
29	_____function is used to find the column count of the particular resultset	getMetaData()	Metadata()	getColumn()	getCount()	getMetaData()
30	_____ statement is a prepared statements	Insert into department values(?	Insert into department values(x,x	SQLSetConnectOption(conn,	SQLTransact(conn, SQL RO	Insert into department values(?,?,?)
31	_____ is used as the embedded SQL in COBOL	EXEC SQL;	EXEC SQL END-EXEC	EXEC SQL	EXEC SQL END EXEC;	EXEC SQL END-EXEC
32	_____ is used to distngush the variables in SQL from the host language variables	.	-	:	,	-
33	_____ is used to perform all input & output operations in java	streams	Variables	classes	Methods	streams
34	AWT stands for_____	All window Tools	All Writing Tools	Abstract Window Toolkit	Abstract Writing Toolkit	Abstract Window Toolkit
35	_____ is a type of stream in java	Integer stream	Short stream	Byte stream	Long stream	Byte stream
36	_____ classes are used by character streams for input and output operations	InputStream	Writer	ReadStream	InputOutputStream	Writer
37	_____ classes are used by byte streams for input and output operation	InputStream	InputOutputStream	Reader	outputStream	InputStream
38	In object class parent class reference variable can refer the child class object, known as	Upcasting	Impilicit casting	Expilicit casting	Boolean casting	Upcasting
39	_____ method compares the given object to this object	public boolean equals(Object o	public final void notifyAll()	public final void notify()	public final ClassgetClass()	public boolean equals(Object obj)
40	The clone() method is defined in_____	Abstract class	Object class	ArrayList class	Fileclass	Object class
41	_____method of object class can clone an object	copy()	Objectcopy()	Objectclone()	Clone()	Objectclone()
42	_____ is an acronym for, it physically exists; it contains JRE+ development tools	JRE	JVM	JDK	JDBC	JDK
43	In _____ standard collection classes implements a linked list data structure	LinkedList	AbstractList	HashSet	ArrayList	LinkedList
44	_____ extends the AbstractList class and implements List and Deque interfaces	AbstractList	LinkedList	HashSet	ArrayList	LinkedList
45	Generally string is a sequence of characters, But in java, string is an_____	Object	Class	Package	long stream	Object
46	In _____ string class function returns the number of characters in a string	length()	replace()	charAt()	equalIgnoreCase()	length()
47	String class in encapsulated under _____ package	java.lang	java.util	java.io	java.awt	java.lang
48	Java defines a peer class of String, called_____	StringBuffer	StringBuilder	StringReader	String Literal	StringBuffer
49	_____ concept is used to make Java more memory efficient (because no new objects ar	String literal	By new keyword	StringBuffer	StringBuilder	String literal
50	A pool of strings, initially empty, is maintained privately by the class String is_____	intern() method	length() method	trim() method	charAt() method	intern() method
51	_____ is thread-safe i.e.multiple threads cannot access it simultaneously, So it is safe a	StringBuffer class	StringBuilder class	StringReader class	StringLiteral class	StringBuffer class
52	_____ constructor creates an empty string buffer with the specified capacity as length	StringBuffer()	StringBuffer(String str)	StringBuffer(int capacity)	StringBuffer(int)	StringBuffer(int capacity)
53	_____constructors are defined in StringTokenizer class	2	3	4	5	3
54	_____method of class string is used to obtain length of string object	get()	sizeof()	length()	length	length
55	_____method of class string is used to extract a single character from a string object	CHARAT()	charat()	charAt()	CharAt()	charAt()
56	_____constructer is used to create an empty string object	String	String(void)	String(0)	String(int)	String

UNIT: III

SYLLABUS

Framework: Overview, Generics Fundamentals, Autoboxing, The Collection Interfaces-List Interface, Set Interface, SortedSet Interface, NavigableSet Interface, The Collection Classes-ArrayList, LinkedList, HashSet, LinkedHashSet, TreeSet, Accessing a Collection via an Iterator, Enumeration Interface, Vector, HashTable, Properties, StringTokenizer and Date Class. Serialization: Serializable, Externalizable.

Collections in Java

1. Java Collection Framework
2. Hierarchy of Collection Framework
3. Collection interface
4. Iterator interface

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSetetc).

What is Collection in java

Collection represents a single unit of objects i.e. a group.

What is framework in java

- provides readymade architecture.
- represents set of classes and interface.
- is optional.

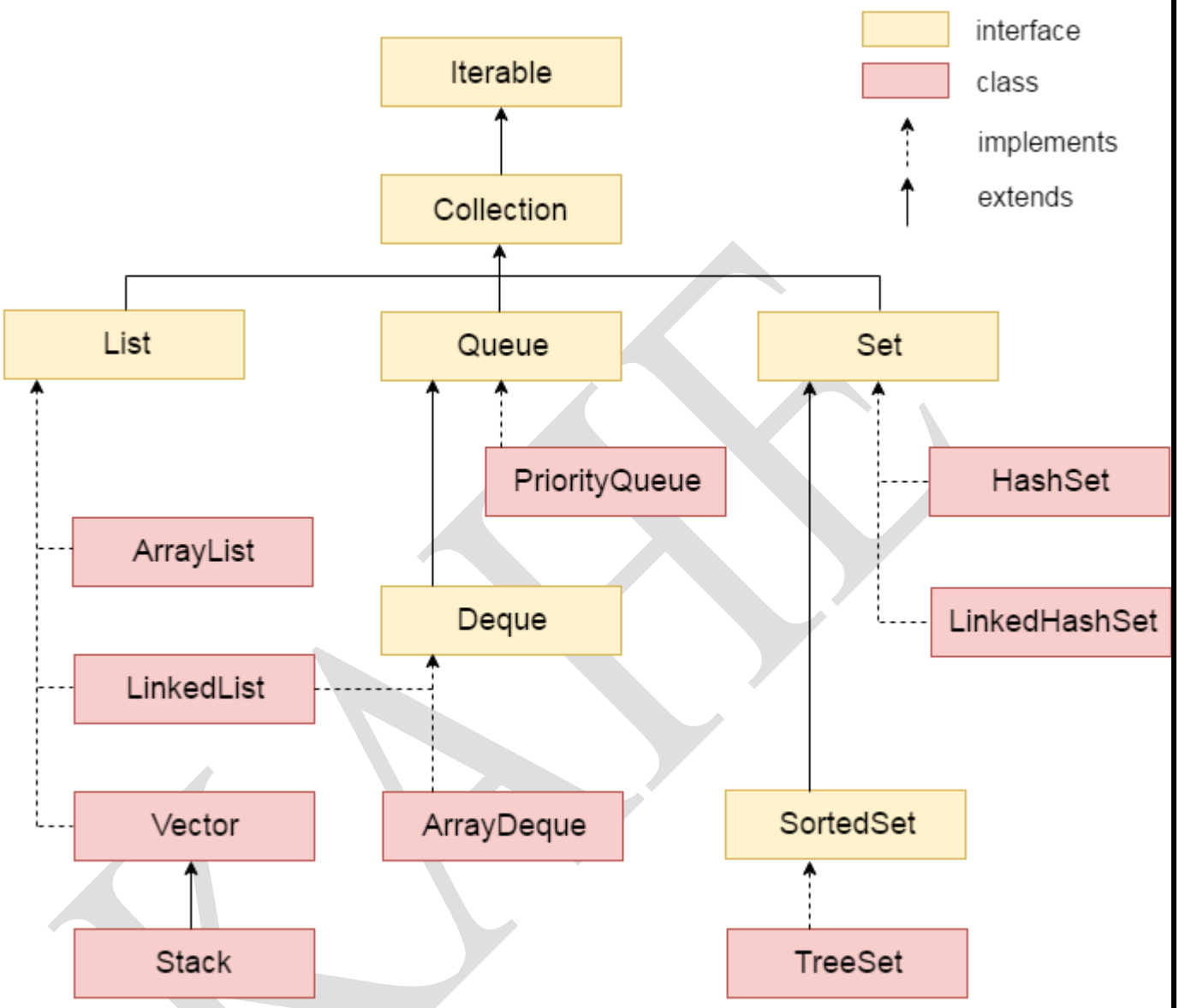
What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm

Hierarchy of Collection Framework

Let us see the hierarchy of collection framework.The java.util package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description

1	public boolean add(Object element)	is used to insert an element in this collection.
2	public booleanaddAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public booleanremoveAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public booleanretainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public booleancontainsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.

12	public booleanisEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public inthashCode()	returns the hashcode number for collection.

Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

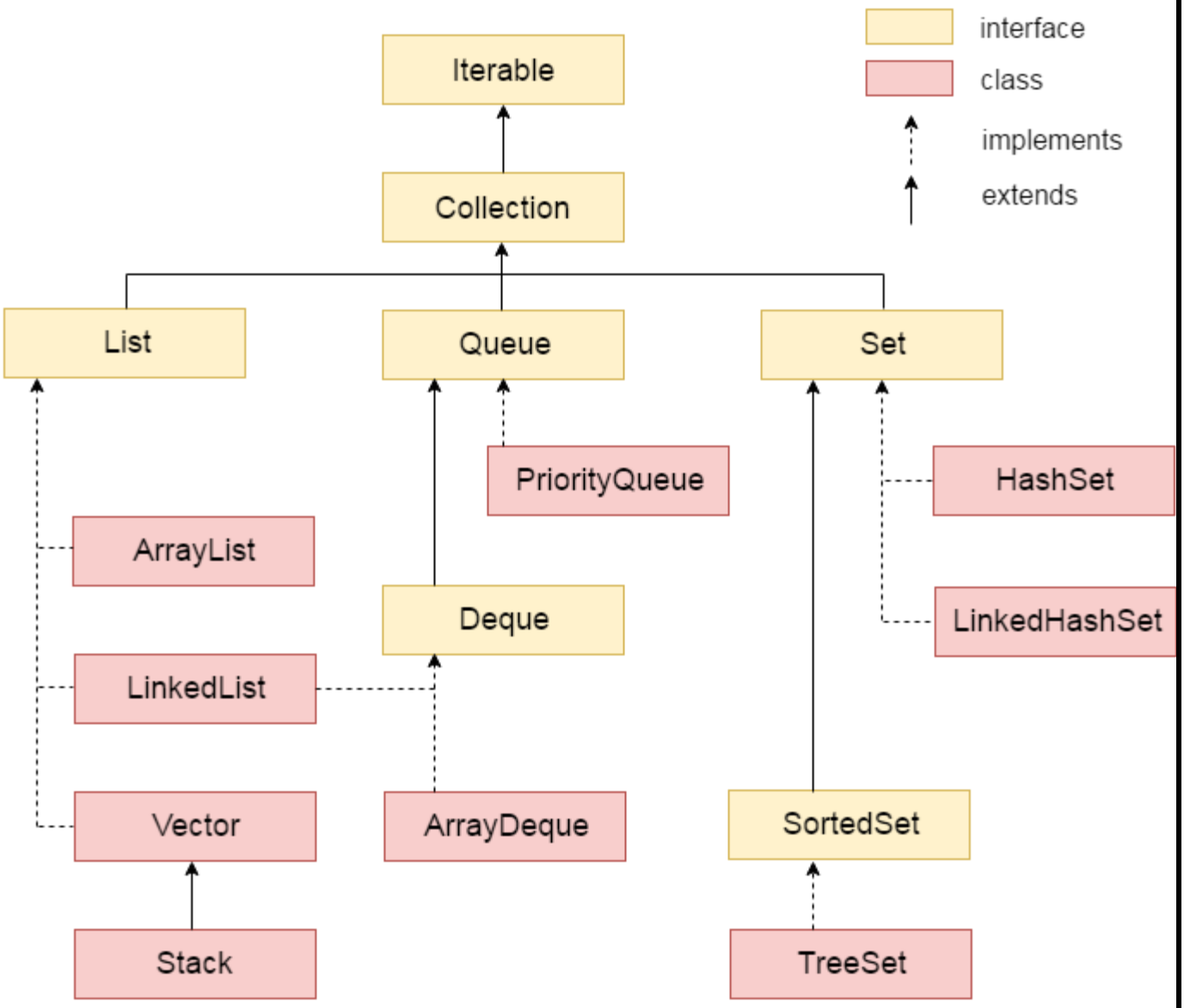
Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

No.	Method	Description
1	public booleanhasNext()	It returns true if iterator has more elements.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is rarely used.

Hierarchy of Collection Framework

Let us see the hierarchy of collection framework.The java.util package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.
2	public booleanaddAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public booleanremoveAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public booleanretainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public	is used to search the specified collection in this

	booleancontainsAll(Collection c)	collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public booleanisEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public inthashCode()	returns the hashcode number for collection.

Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

No.	Method	Description
1	public booleanhasNext()	It returns true if iterator has more elements.
2	public Object next()	It returns the element and moves the cursor pointer to the next

		element.
3	public void remove()	It removes the last elements returned by the iterator. It is rarely used.

Java List Interface

List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface.

List Interface declaration

1. public interface List<E> extends Collection<E>

Methods of Java List Interface

Method	Description
void add(intindex,Object element)	It is used to insert element into the invoking list at the index passed in the index.
booleanaddAll(intindex,Collection c)	It is used to insert all elements of c into the invoking list at the index passed in the index.
object get(int index)	It is used to return the object stored at the specified index within the invoking collection.
object set(intindex,Object element)	It is used to assign element to the location specified by index within the invoking list.

object remove(int index)	It is used to remove the element at position index from the invoking list and return the deleted element.
ListIteratorlistIterator()	It is used to return an iterator to the start of the invoking list.
ListIteratorlistIterator(int index)	It is used to return an iterator to the invoking list that begins at the specified index.

Java List Example

```
1. import java.util.*;
2. public class ListExample{
3. public static void main(String args[]){
4. ArrayList<String> al=new ArrayList<String>();
5. al.add("Amit");
6. al.add("Vijay");
7. al.add("Kumar");
8. al.add(1,"Sachin");
9. System.out.println("Element at 2nd position: "+al.get(2));
10. for(String s:al){
11.  System.out.println(s);
12. }
13. }
14. }
```

Output:

```
Element at 2nd position: Vijay
Amit
Sachin
```

Vijay
Kumar

Java List Interface

List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface.

List Interface declaration

1. public interface List<E> extends Collection<E>

Methods of Java List Interface

Method	Description
void add(intindex,Object element)	It is used to insert element into the invoking list at the index passed in the index.
booleanaddAll(intindex,Collection c)	It is used to insert all elements of c into the invoking list at the index passed in the index.
object get(int index)	It is used to return the object stored at the specified index within the invoking collection.
object set(intindex,Object element)	It is used to assign element to the location specified by index within the invoking list.
object remove(int index)	It is used to remove the element at position index from the invoking list and return the deleted element.

ListIteratorlistIterator()	It is used to return an iterator to the start of the invoking list.
ListIteratorlistIterator(int index)	It is used to return an iterator to the invoking list that begins at the specified index.

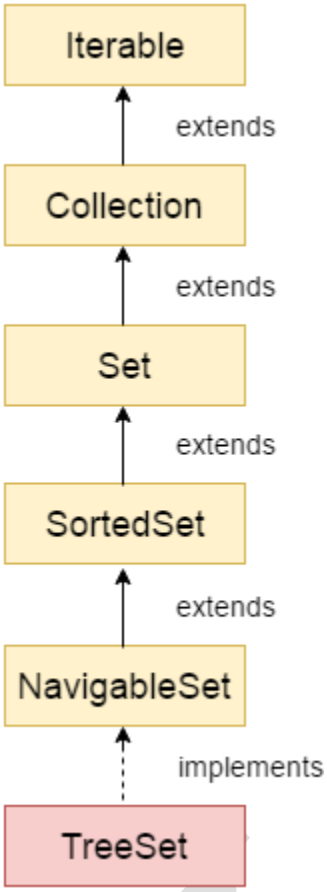
Java List Example

```
1. import java.util.*;
2. public class ListExample{
3. public static void main(String args[]){
4. ArrayList<String> al=new ArrayList<String>();
5. al.add("Amit");
6. al.add("Vijay");
7. al.add("Kumar");
8. al.add(1,"Sachin");
9. System.out.println("Element at 2nd position: "+al.get(2));
10. for(String s:al){
11.  System.out.println(s);
12. }
13. }
14. }
```

Output:

```
Element at 2nd position: Vijay
Amit
Sachin
Vijay
Kumar
```

Java TreeSet class



Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

- Contains unique elements only like HashSet.
- Access and retrieval times are quiet fast.
- Maintains ascending order.

Hierarchy of TreeSet class

As shown in above diagram, Java TreeSet class implements NavigableSet interface. The NavigableSet interface extends SortedSet, Set, Collection and Iterable interfaces in hierarchical order.

TreeSet class declaration

Let's see the declaration for java.util.TreeSet class.

1. public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, Cloneable, Serializable

Constructors of Java TreeSet class

Constructor	Description
TreeSet()	It is used to construct an empty tree set that will be sorted in an ascending order according to the natural order of the tree set.
TreeSet(Collection c)	It is used to build a new tree set that contains the elements of the collection c.
TreeSet(Comparator comp)	It is used to construct an empty tree set that will be sorted according to given comparator.
TreeSet(SortedSetss)	It is used to build a TreeSet that contains the elements of the given SortedSet.

Methods of Java TreeSet class

Method	Description
booleanaddAll(Collection c)	It is used to add all of the elements in the specified collection to this set.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
booleanisEmpty()	It is used to return true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
void add(Object o)	It is used to add the specified element to this set if it is not already present.
void clear()	It is used to remove all of the elements from this set.
Object clone()	It is used to return a shallow copy of this TreeSet instance.
Object first()	It is used to return the first (lowest) element currently in this sorted set.
Object last()	It is used to return the last (highest) element currently in this sorted set.
int size()	It is used to return the number of elements in this set.

Java TreeSet Example

```
1. import java.util.*;
2. class TestCollection11{
```

```
3.   public static void main(String args[]){
4.    //Creating and adding elements
5.    TreeSet<String> al=new TreeSet<String>();
6.    al.add("Ravi");
7.    al.add("Varun Vijay");
8.    al.add("Ravi");
9.    al.add("Ajay");
10.   //Traversing elements
11.   Iterator<String> itr=al.iterator();
12.   while(itr.hasNext()){
13.    System.out.println(itr.next());
14.   }
15.  }
16. }
```

Test it Now

Output:

```
Ajay
Ravi
Varun Vijay
```

Serialization in Java

1. Serialization
2. Serializable Interface
3. Example of Serialization
4. Deserialization
5. Example of Deserialization
6. Serialization with Inheritance
7. Externalizable interface

8. Serialization and static datamember

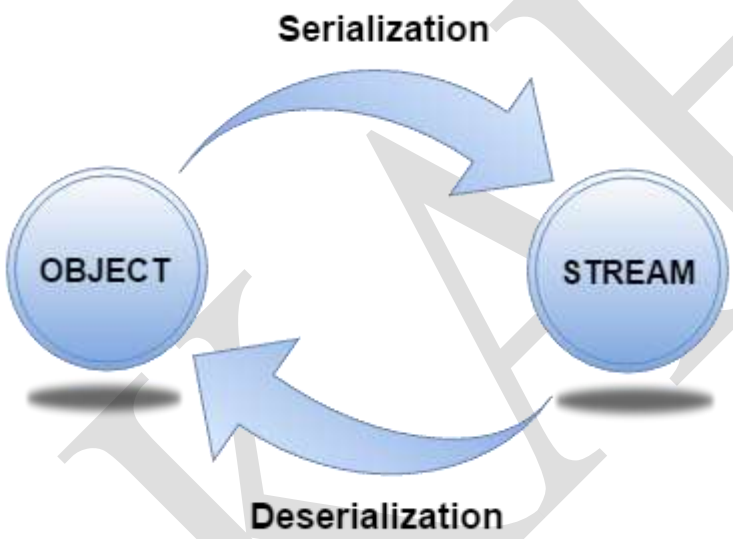
Serialization in java is a mechanism of *writing the state of an object into a byte stream*.

It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

The reverse operation of serialization is called *deserialization*.

Advantage of Java Serialization

It is mainly used to travel object's state on the network (known as marshaling).



java.io.Serializable interface

Serializable is a marker interface (has no data member and method). It is used to "mark" java classes so that objects of these classes may get certain capability. The Cloneable and Remote are also marker interfaces.

It must be implemented by the class whose object you want to persist.

The String class and all the wrapper classes implements *java.io.Serializable* interface by default.

Let's see the example given below:

```
1. import java.io.Serializable;
2. public class Student implements Serializable{
3.  int id;
4.  String name;
5.  public Student(int id, String name) {
6.   this.id = id;
7.   this.name = name;
8.  }
9. }
```

POSSIBLE QUESTIONS

PART-A (20 Marks)

(Q.No 1 to 20 Online Examination)

PART-B (6 Marks)

1. **Describe the methods of List Interface**
2. **Differentiate between Enumeration and Iterator interface**
3. **What is Java Collections Framework?. List out some benefits of Collections framework**
4. **Mention the methods to add, remove and locate elements in a Vector class with suitable example**
5. **Explain SortedSet Interface**
6. **Discuss the methods of HashTable**
7. **Explain the operation of each of the following methods of class Hashtable.**

   i) put ii) get iii) isEmpty iv) containsKey v)keyset

8. **Discuss Serializable and Externalizable with suitable examples**
9. **Explain the constructors and methods of Set Interface**
10. **List out the difference between Serializable and Externalizable**

PART-C(10 MARKS)

1. Use an ArrayList to demonstrate several Collection Interface capabilities and use an Iterator to remove the strings from the ArrayList Collection

QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	OPT 5	OPT 6	ANSWER
The_____ packages contain all the collection classes.	java.lang	java.util	java.net	java.awt			java util
A _____ class is not a part of java collection framework	Maps	Array	Stack	Queue			Maps
_____Interface is not a part java collection framework	List	Set	SortedMap	SortedList			SortedList
A_____ methods is used to deletes all the element invoking collection.	clear()	reset()	delete()	refresh()			clear()
A group _____ is a collection in java.	Objects	Classes	interface	Abstract classes			Objects
The _____ class is a superclass of String and StringBuffered class	java.lang	java.util	ArrayList	Object			java.lang
The _____ operator is used to concatenate two or more String Objects.	+	=	&	||			+
A_____ method is used to extract a single character from a string Object	get()	sizeof()	lengthof()	length()			length()
_____ constructors is used to create an empty string Object	String()	String(void)	String(0)	String(char)			String()
A stream Process of extracting/removing the state of an object is called_____.	Serialization	Externalization	File Filtering	Deserialization			Deserialization
A_____ process occur automatically by java run time system.	Serialization	ObjectStream	ObjectOutput	ObjectInput			serialization
_____ types cannot be used to initiate a generic type	Interger class	Float class	Primitive types	collection			Primitive types
A_____ type of interface is used to extends DataOutput interfaces	Serialization	Externalization	Objectoutput	ObjectInput			ObjectOutput
A_____class is used to extend InputStream class.	ObjectStream	ObjectStreamInput	Objectoutput	ObjectInput			ObjectStreamInput
_____ is a method of ObjectInput interface used to deserializa an object from a stream	int read()	void class	object read object	object write object			Object read Object
The_____ collection classes implements a LinkedList data structure.	AbstractList	LinkedList	HashSet	AbstractSet			LinkedList
_____ class implements set interface.	AbstractList	LinkedList	HashSet	Dynamic List			HashSet
A_____ method is used to add an elements to start a LinkedList object.	add()	First()	addFirst()	AddFirst()			addFirst()
_____ method is used to add the elements in HASHSET class.	add()	Add()	addFirst()	insert()			add()
which of the following classes directly implement set interface?	Vector	HashSet	LinkedList	HashTable			HashSet
A group of ---------- is a collection in java.	Objects	Classes	interface	String			Objects
A _____ method is used to delete the last element in a LinkedList Object	remove()	delete()	removeLast()	deleteLast()			removeLast()
_____ method of class string is used to obtain length of string object.	get()	sizeof()	lengthof()	length()			length()
A_____keywords is used to define interface in java.	interface	Interface	intf	Intf			interface
_____ can be used to fully abstract a class from its implementation.	Object	Packages	interface	String			interface
A_____ package store all the standard java classes.	lang	java	util	.net			java
A_____ class must be extends by all event classes.	java.util.EventListner	java.util.EventObject	java.awt.AWTEvent	java.awt.Event.InputEvent			java..util.EventObject
_____ access specifies can be used to for an interface.	public	private	protected	none			public
A _____package class belongs to the math.	java.math	java.lang	java.util	java.net			java.math
A_____instance cannot be created	interger instance	generic class instance	generic type instance	collection instance			generic type instance
_____ keywords is used as a class interface to defined previously.	import	Import	implements	Implements			implements
Treemap class is used to implement _____ collection interface.	Set	SortedSet	SortedMap	List			SortedSet
_____is the name of the collection interface used to represent elements in a sequence.	Collection	Set	List	Map			List
A_____ classes directly implement set interface.	Vector	HashSet	LinkedList	HashTable			HashSet
Treemap class is used to implement _____ collection interface.	Set	SortedSet	SortedMap	List			SortedSet
The first statement in java source file _____.	import statement	package statement	main statement	try{} and catch{}			package statement
A package is a collection of_____.	classes	interface	editing file	classes and interface			classes and interface
For which purpose packages are used in java?	categorizes data	Organizing java classes into namespace	for faster compilation	Terminate the process			Organizing java classes into namespaces
A_____ parameter is used for a generic methods to return &acceptvany type of object.	K	N	T	V			T
A_____ type parameter is used for a generic methods to return &acceptvany type of a number.	K	N	T	V			N
_____ allows us to call generate methods as a normal method.	Type interface	Interface	Inner class	Outer class			Type interface
A_____ iteration can be used only with List.	Setiteration	ListIterator	Literator	iterator()			Listiterator
A_____ methods can be used to move to next elements in a collection.	next	move	shuffle()	hasNext()			next
_____ return type of hasNext() method of an iterator.	Interger	Double	Boolean	Collection Object			Boolean
A_____ methods is used to obtain an iterator to the start ofb collection	start()	begin()	iterator()Set	iterator()			iterator()
_____ exception is thrown by remove method.	IoException	SystemException	ObjectNotFoundException	IllegalStateException			IllegalStateException
Interface which declares core method that all collection is _____.	Set	EventLister	Comparator	Collection			Collection
A_____ interface handle sequences	Set	List	Collection	Comparator			List
_____ interface must contains a unique element.	List	Set	Array	Collection			Set
_____ is basic interafce that all other interface inherits.	Set	Array	Collection	list			Collection
A_____ interface is implemented by timer task class	Runnable	Thread	Observer	Threadcount			Runnable
_____ package provides the ability to read and write in zip format.	java.lang	java.io	java.util.zip	java.util.zar			java.util.zip
_____ type of data canot be type parameterized.	Array	List	Map	Set			Array
A_____instance cannot be created	interger instance	generic class instance	generic type instance	collection instance			generic type instance
A_____ is static variable which is defined in collection.	EMPTY_SET	EMPTY_LIST	EMPTY_MAP	All of these			all of these
A_____ types cannot be used to initiate a generic type	Interger class	Float class	Primitive types	collection			Primitive types
_____ class is used to extend InputStream class.	ObjectStream	ObjectStreamInput	Objectoutput	ObjectInput			ObjectStreamInput
_____ interface for control overs serialization and deserialization.	Serialization	Externalization	File Filtering	ObjectInput			Externalization
A group _____ is a collection in java.	Objects	Classes	interface	String			Objects
A_____ class directly implement set interface.	Vector	HashSet	LinkedList	HashTable			HashSet

UNIT IV

SYLLABUS

Introducing the Spring Framework, Spring Framework RunTime & architecture, Inversion of Control (IoC), Dependency Injection, Different Forms of Dependency Injection, Dependency Injection variants, DI classes & Parameter in Spring framework, Bean naming, @Autorwired annotation, The Bean Factory, XML Bean Configuration, Managing the Bean Lifecycle, Basics of Aspect-Oriented Programming (AOP), AOP concepts - Join point, Pointcut, Advice, Types of advice, @AspectJ support

1. Introduction to Spring Framework

Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.

Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.

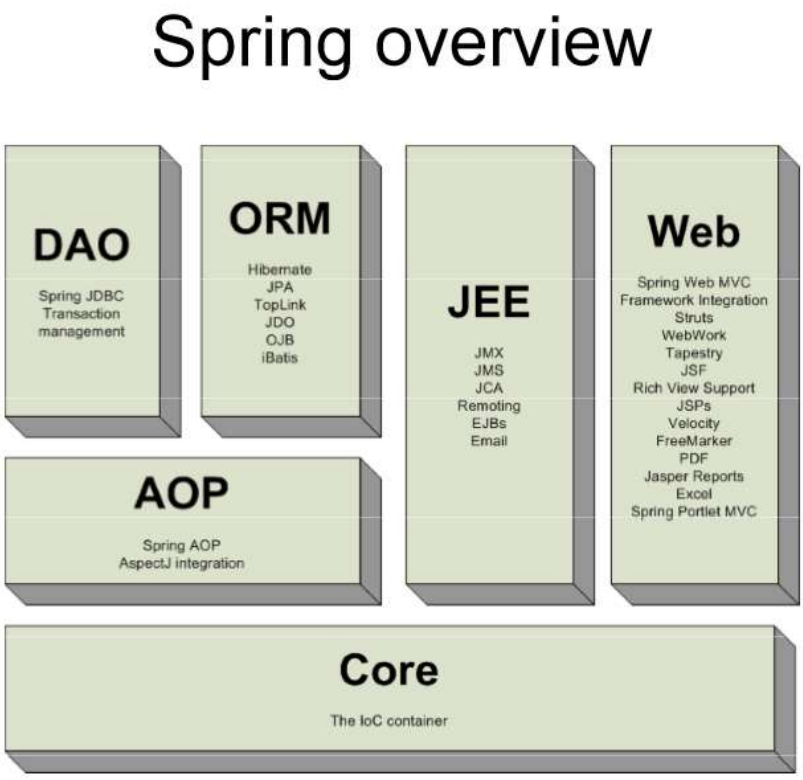
Examples of how you, as an application developer, can use the Spring platform advantage:

Make a Java method execute in a database transaction without having to deal with transaction APIs.

Make a local Java method a remote procedure without having to deal with remote APIs.

Make a local Java method a management operation without having to deal with JMX APIs.

Make a local Java method a message handler without having to deal with JMS APIs.

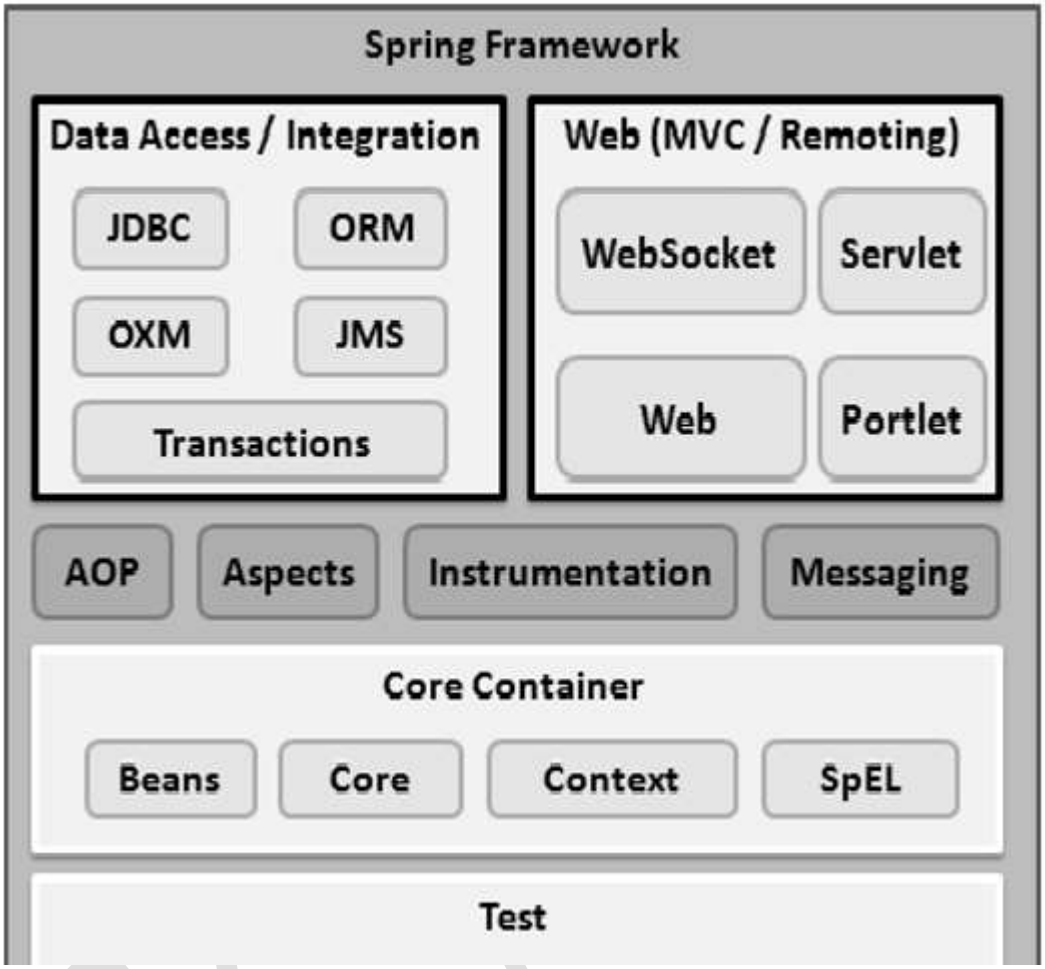


Spring architecture

Spring could potentially be a one-stop shop for all your enterprise applications. However, Spring is modular, allowing you to pick and choose which modules are applicable to you, without having to bring in the rest. The following section provides details about all the modules available in Spring Framework.

The Spring Framework provides about 20 modules which can be used based on an

application requirement.



Core Container

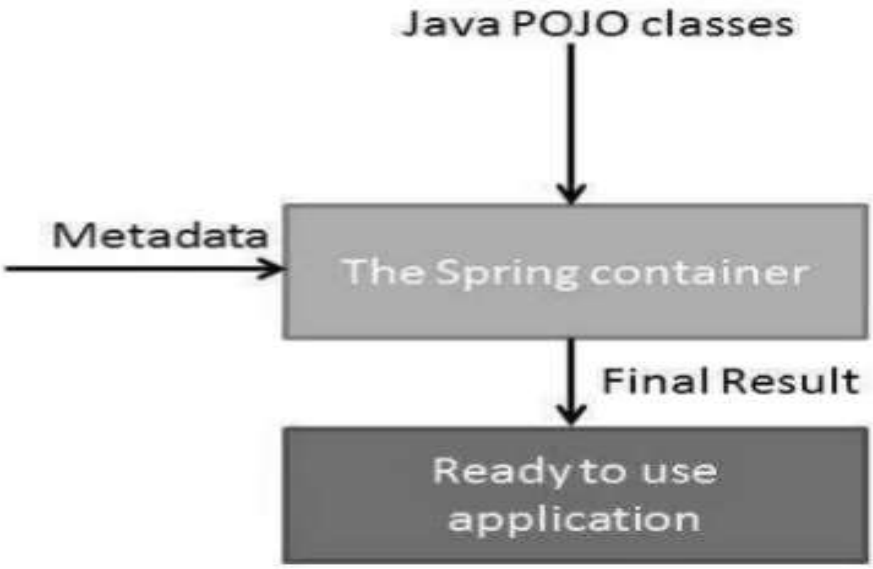
The Core Container consists of the Core, Beans, Context, and Expression Language modules the details of which are as follows:

 The Core module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.

 The Bean module provides BeanFactory, which is a sophisticated implementati3on.

Spring IOC Container

The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses DI to manage the components that make up an application. These objects are called Spring Beans, which we will discuss in the next chapter. The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code. The following diagram represents a high-level view of how Spring works. The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.



Spring provides the following two distinct types of containers.

S.No	Container & Description
1	Spring BeanFactory Container This is the simplest container providing the basic support for DI and is defined by the org.springframework.beans.factory.BeanFactoryinterface. The BeanFactory and related interfaces, such as BeanFactoryAware, InitializingBean, DisposableBean, are still present in Spring for the purpose of backward compatibility with a large number of third-party frameworks that integrate with Spring. 5. Spring ─ IoC Containers Spring Framework 18
2	Spring ApplicationContext Container This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by theorg.springframework.context.ApplicationContext interface.

Dependency injection

Every Java-based application has a few objects that work together to present what the end-user sees as a working application. When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing. Dependency

Injection (or sometime called wiring) helps in gluing these classes together and at the same time keeping them independent.

Consider you have an application which has a text editor component and you want to provide a spell check. Your standard code would look something like this −

```
public class TextEditor {

privateSpellCheckerspellChecker;

publicTextEditor() {

spellChecker = new SpellChecker();

  }

}
```

What we've done here is, create a dependency between the TextEditor and the SpellChecker. In an inversion of control scenario, we would instead do something like this −

```
public class TextEditor {

privateSpellCheckerspellChecker;

publicTextEditor(SpellCheckerspellChecker) {

this.spellChecker = spellChecker;   }  }
```

Here, the TextEditor should not worry about SpellChecker implementation. The SpellChecker will be implemented independently and will be provided to the TextEditor at the time of TextEditor instantiation. This entire procedure is controlled by the Spring Framework.

Here, we have removed total control from the TextEditor and kept it somewhere else (i.e. XML configuration file) and the dependency (i.e. class SpellChecker) is being injected into the class TextEditor through a Class Constructor. Thus the flow of control has been "inverted" by Dependency Injection (DI) because you have effectively delegated dependances to some external system.

The second method of injecting dependency is through Setter Methods of the TextEditor class where we will create a SpellChecker instance. This instance will be used to call setter methods to initialize TextEditor's properties.

Thus, DI exists in two major variants and the following two sub-chapters will cover both of them with examples −

S.No	Dependency Injection Type & Description
1	Constructor-based dependency injection Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on the other class.
2	Setter-based dependency injection Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean....

Types of dependency Injection in Spring

There are two main types of dependency injections in spring and one other less used type.

1.Constructor based injection

In spring Constructor injection the dependent objects are injected into the client using the client's constructor. this tutorial looks into more details into the constructor injection.

2.Setter based injection

In this type of injection the dependencies are injected after the client object has been created. The dependencies are injected using the setter method of the client. This tutorials looks into details of Setter injection.

Wiring with annotations

Since Spring 2.5, one of the most interesting ways of wiring beans in Spring has been to use annotations to automatically wire bean properties. Autowiring with annotations isn't much different than using the autowire attribute in XML. But it does allow for more fine-grained autowiring, where you can selectively annotate certain properties for autowiring. Annotation wiring isn't turned on in the Spring container by default. So, before we can use annotation-based autowiring, we'll need to enable it in our Springconfigura- tion. The simplest way to do that is with the <context:annotation-config> element from Spring's context configuration namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">
<context:annotation-config />
<!-- bean declarations go here -->
</beans><context:annotation-config>
```

tells Spring that you intend to use annotation-based wiring in Spring. Once it's in place you can start annotating your code to indicate that Spring should automatically wire values into properties, methods, and constructors. Spring 3 supports a few different annotations for autowiring: Spring's own @Autowired annotation The @Inject annotation from JSR-330 The @Resource annotation from JSR-250 We'll look at how to use Spring's @Autowired first. Then we'll try out standards-based dependency injection with JSR-330's @Inject and JSR-250's @Resource.

Wiring with annotations

3.2.1 Using @Autowired Suppose that you want to use @Autowired to have Springautowire the instrument property of the Instrumentalist bean. You could annotate the setInstrument() method like this:

@Autowired public void setInstrument(Instrument instrument) { this.instrument = instrument; }

Now you can get rid of the <property> element that wires the Instrumentalist with an instrument. When Spring sees that you've annotated setInstrument() with @Autowired it'll try to perform byTypeautowiring on the method. What's especially interesting about @Autowired is that you don't have to use it with a setter method. You can use it on any method to automatically wire in bean references:

@Autowired public void heresYourInstrument(Instrument instrument) { this.instrument = instrument; } The @Autowired annotation can even be used on constructors:

@Autowired public Instrumentalist(Instrument instrument) { this.instrument = instrument; }

When used with constructors, @Autowired indicates that the constructor should be autowired when creating the bean, even if no <constructor-arg> elements are used to configure the bean in XML. What's more, you can directly annotate properties and do away with the setter methods altogether:

@Autowired private Instrument instrument; As you can see, @Autowired won't even be thwarted by the private keyword. Even though the instrument property is private, it'll still be autowired. Is there no limit to @Autowired's reach? Actually, there are a couple of circumstances that could keep @Autowired from get- ting its job done. Specifically, there must be exactly one bean that's applicable for wir- ing into the @Autowired property or parameter. If

there are no applicable beans or if multiple beans could be autowired, then @Autowired will run into some trouble. Fortunately, there's a way that we can help @Autowired out in those circumstances. First, let's look at how to keep @Autowired from failing when there isn't a matching bean.

Bean- LifeCycle

The life cycle of a Spring bean is easy to understand. When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required.

Though, there are lists of the activities that take place behind the scene between the time of bean Instantiation and its destruction, this chapter will discuss only two important bean life cycle callback methods, which are required at the time of bean initialization and its destruction.

To define setup and teardown for a bean, we simply declare the <bean> with init- method and/or destroy-method parameters. The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation. Similarly, destroy- method specifies a method that is called just before a bean is removed from the container.

Initialization CallbacksTheorg.springframework.beans.factory.InitializingBean interface specifies a single method:

voidafterPropertiesSet() throws Exception;

Thus, you can simply implement the above interface and initialization work can be done inside afterPropertiesSet() method as follows:

```
public class ExampleBean implements InitializingBean
 {
public void afterPropertiesSet() {
    // do some initialization work
  }
}
```

In the case of XML-based configuration metadata, you can use the init-method attribute to specify the name of the method that has a void no-argument signature. For example:

<bean id="exampleBean"

class="examples.ExampleBean" init-method="init"/>

8. Spring – Bean Life Cycle

Spring Framework

34

Following is the class definition:

```
public class ExampleBean {
public void init() {
    // do some initialization work
  }
}
```

Destruction CallbacksTheorg.springframework.beans.factory.DisposableBean interface specifies a single method:

```
void destroy() throws Exception;
```

Thus, you can simply implement the above interface and finalization work can be done inside destroy() method as follows:

```
public class ExampleBean implements DisposableBean {
public void destroy() {
    // do some destruction work
  }
}
```

In the case of XML-based configuration metadata, you can use the destroy- method attribute to specify the name of the method that has a void no-argument signature. For example:

<bean id="exampleBean"

class="examples.ExampleBean" destroy-method="destroy"/>

Following is the class definition:

```
public class ExampleBean {
public void destroy() {
    // do some destruction work
```

```
  }
}
```

If you are using Spring'sIoC container in a non-web application environment; for example, in a rich client desktop environment, you register a shutdown hook with the JVM. Doing so ensures a graceful shutdown and calls the relevant destroy methods on your singleton beans so that all resources are released.

Spring Framework

35

It is recommended that you do not use the InitializingBean or DisposableBeancallbacks, because XML configuration gives much flexibility in terms of naming your method.

Example Let us have a working Eclipse IDE in place and take the following steps to create a Spring application:

Steps Description

1

Create a project with a name SpringExample and create a packagecom.tutorialspoint under the src folder in the created project.

2

Add required Spring libraries using Add External JARs option as explained in the Spring Hello World Example chapter.

3

Create Java classes HelloWorld and MainApp under the com.tutorialspointpackage.

4 Create Beans configuration file Beans.xml under the src folder.

5

The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Here is the content of HelloWorld.java file:

```
packagecom.tutorialspoint;
public class HelloWorld {
private String message;
```

```
public void setMessage(String message){
this.message  = message;
  }
public void getMessage(){
System.out.println("Your Message : " + message);
  }
public void init(){
System.out.println("Bean is going through init.");
  }
public void destroy(){
System.out.println("Bean will destroy now.");
  }
}
```

AOP WITH SPRING FRAMEWORK

One of the key components of Spring Framework is the Aspect oriented programming (AOP) framework. Aspect-Oriented Programming entails breaking down program logic into distinct parts called so-called concerns. The functions that span multiple points of an application are called cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects like logging, auditing, declarative transactions, security, caching, etc.

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Dependency Injection helps you decouple your application objects from each other and AOP helps you decouple cross-cutting concerns from the objects that they affect. AOP is like triggers in programming languages such as Perl, .NET, Java, and others.

Spring AOP module provides interceptors to intercept an application. For example, when a method is executed, you can add extra functionality before or after the method execution..

Types of Advice

Spring aspects can work with five kinds of advice mentioned as follows −

Sr.No	Advice & Description
1	before Run advice before the a method execution.
2	after Run advice after the method execution, regardless of its outcome.
3	after-returning Run advice after the a method execution only if method completes successfully.
4	after-throwing Run advice after the a method execution only if method exits by throwing an exception.
5	Around Run advice before and after the advised method is invoked.

POSSIBLE QUESTIONS

PART-A (20 Marks)

(Q.No 1 to 20 Online Examination)

PART-B (6 Marks)

1. Discuss the Spring Framework Architecture
2. Write the steps to create the Spring Application
3. What are the different types of Inversion of control (IoC)?
4. Discuss BeanFactory implementation with suitable example
5. What are the benefits of Spring Framework?
6. Explain the Aspect-Oriented Programming (AOP) Module.
7. Which DI would you suggest Constructor-based or setter-based DI?Justify
8. Explain Bean lifecycle in Spring Framework
9. Explain Bean naming and @Autowired Annotation
10. Explain annotation-based (@AspectJ based) aspect implementation

PART-C(10 Marks)

1. What is the concept of AOP? Which problem does it solve?.

KARPAGAM ACADEMY OF HIGHER EDUCATION
COIMBATORE - 21
DEPARTMENT OF COMPUTER APPLICATIONS

CLASS : II MCA **BATCH : 2017-2020**

SUBJECT: ADVANCED JAVA AND SPRING

S.NO	QUESTION	OPTION A	OPTION B	OPTION C	OPTION D	ANSWER
1	Beans can be created by _____ property	Scope	Property	Class	own constructor	own constructor
2	_____ attribute is used to specify classname of the bean	name	id	class	constructor-args	class
3	_____ method can be used to instantiate a method	static factory method	default-init method	destroy method	lazy-init method	static factory method
4	_____ attribute is used to specify static factory-method	factory-method	default-init method	destroy method	lazy-init method	factory-method
5	Static Factory Method is used to _____	to create an object	Initialize bean	Initialize class	to create a class	to create an object
6	Exception thrown by factory method is _____	IllegalArgumentException	IndexOutofBoundException	ClassPathNotFoundException	BeanCreationException	BeanCreationException
7	_____ factory class can also hold more than one factory method	TRUE	FALSE	FALSE and TRUE	TRUE/FALSE	TRUE
8	_____ Attribute is used to specify the bean declared	factory-bean	scope	getBean	declareBean	factory-bean
9	_____Tag used to enable AspectJ annotation	Introduction	aop:aspectj-autowire	aop:aspectj-autoproxy	AfterSpecial	aop:aspectj-autoproxy
10	Spring AOP configurations is defined by _____ tag	aop:config	aop:configregister	aop:configbeans	aop:bean	aop:config
11	_____ way to declare an advice	pointcut-ref attribute	pointcut attribute	jointpoint	advice	all of the mentioned
12	_____ aspects is used to your target objects	AspectJ Annotation	Weaving	AspectJ	AspectJ Support	Weaving
13	Special compiler used during weaving is _____	jvm	gcc	ajc	cc	ajc
14	Target classes are loaded into JVM by_____ process	load-time weaving	process-time weaving	load-process weaving	process-delivery weaving	load-time weaving
15	_____is used to weave your classes using argument while compiling	javaagent:CLASSPATH	javaagent:PackgePath	javaweave:CLASSPATH	javaweave:PackagePath	javaagent:CLASSPATH
16	_____XML Element is used to include the load-time weaver	aop:config	aop:auto-wire	context:load-time-weaver	aop:load-time-weaver	context:load-time-weaver
17	_____Library is used to AspectJ weaver	spring-instrument.jar	spring-introduction.jar	spring-aop.jar	spring-weave.jar	spring-instrument.jar
18	_____Object is created by outside the container	Domain Objects	User Objects	SpringVisitor Objects	domain	Domain Objects
19	To inject the Spring bean into _____domain objects	AOP	XML	AspectJ	Java Based	AOP
20	The scope does @Configurable instantiated class look for _____	Singleton	Prototype	scope	spring	Prototype
21	_____ class acts as IoC Container	ServletContext	DispatcherServlet	ApplicationContext	Servlet	ApplicationContext
22	_____ class is used to map a database to row a java object in spring	ResultSet	RowMapper	RowSetMapper	ResultSetMapper	RowMapper
23	_____is used to BeanPostProcessor	concrete class	interface	abstract class	class	interface
24	The _____helps in gluing the class togeather at the same time keeping them independent	Dependency Injection	Annotation	Aspect	AspectJ	Dependency Injection
25	The dependency is being injected into the class through a _____	Setter Method	Getter Method	Class Constructor	Setter/Getter Method	Class Constructor
26	_____ is accomplished when the container invokes a class constructor with a number of arguments of another class	Setter Based DI	Constructor Based DI	Getter Based DI	Getter/Setter DI	Constructor Based DI
27	_____are bean that are defined within the scope of another bean	spellchecker	bean	inner bean	XmlBean	Inner bean
28	_____helps in wiring that is injecting a list of values	<map>	<list>	<set>	<props>	<list>
29	DI is a process where by the objects defines by_____	dependencies	injection	IOC	DI	dendencies
30	_____is an application framework and IOC container of the java platform	SpringVisitor Objects	Java Based	XML	Spring Framework	Spring Framework
31	The Spring Framework consists of features organized by_____modules	5	20	15	30	20
32	The order of bean intialization is the same,as it's defined in the_____file	Spring Bean Configuration	XML	class	jar	Spring Bean Configuration
33	In a case of XML based configuration meta data,_____method is used	factory	destroy	init	bean factory	destroy
34	The most commonly used BeanFactory implementation is the _____class	thread	exception	XML BeanFactory	bean	XML BeanFactory
35	_____is the simplest container providing the basic suppport of DI	SpringBeanFactoryContainer	SpringApplicationContextContair	Spring	ApplicationContext	SpringBeanFactoryContainer
36	_____attribute is mandatory and specifies the bean class to be used to create a bean	properties	class	name	scope	class
37	The execution of the method or handling of an exception is_____	SpringFramework	DI	Spring AOP	Aspect	Spring AOP
38	All modules are built on _____ of its core container	Top	Bottom	Center	Down	Top
39	The Spring Framework is_____	Freeware	open source	Software	Hardware	open source
40	_____control is getting freedom, more flexibility,and less dependency	Inversion of control	bean	Java Based	AOP	Inversion of control
41	_____is a container	SpringFramework	bean factory	XmlBean	SQL	bean factory
42	_____is a method that set the name of the bean in bean factory	setMessageSource()	setBeanClassLoader()	setResourceLoader()	setBeanName()	setBeanName()
43	_____is a specification in the corresponding main program the aspect code should be executed	Join Point	Point cut	Advice	AOP	Join point
44	One of the key components of spring is_____	Spring Framework	AOP Framework	Spring AOP	DI	Spring Framework
45	_____is a (AOP) extension created at PARC of the java programming language	AspectJ	Aspect	AspectJ Annotation	DI	AspectJ
46	After instance creation _____will be injected	thread	Dependency	DI	IOC	Dependency
47	Beans are instantiated & managed by_____	Spring IOC Container	Java code	XML	Bean Class	Spring IOC Container
48	DI is a design pattern is used to _____	SpringFramework	IOC	Software	Object Dependencies	Object Dependencies
49	Design pattern is used to_____ dependency of the Programming code	Create	Remove	Delete	Destroy	Remove
50	_____accompolish the constructor invokes a class constructor with a no. of arguments	Getter based DI	Setter based DI	Constructor based DI		Constructor based DI
51	_____is an action taken by an aspect at the particular join point	Advice	Join point	Point cut	IOC	Advice
52	_____allows introduce the new interface and a corresponding implementation to any advised ob	Spring IOC Container	Spring AOP	SpringFramework	Spring Bean Configuration	Spring AOP
53	DI is a also a software pattern that's implements for IOC for Dependencies	resolving	creating	manipulating	invokes	resolving
54	Spring 2.0 interprets the same annotations as_____	AspectJ5	AspectJ	jar	init	AspectJ5
55	Domain Objects created outside the	Domain Objects	User Objects	Container	object	Container
56	Exception thrown by _____ method	Factory method	bean method	run method	compile	Factory method

UNIT – V

SYLLABUS

DAO Support and JDBC Framework, Operations with JdbcTemplate, JdbcTemplate Convenience Methods, Basic Queries Using the JdbcTemplate, Batch Updates, Transaction and Resource Management, Global transaction vs. local transaction, Declarative transaction management, XML-based, Annotation-based, Object/Relational Mapping, Basic O/R Mapping, Object Query Languages, Data Access Objects, Setup in a Spring Context, Introduction to Spring MVC, DispatchServlet, Context configuration, Identify the design goals and core concepts of Spring MVC, Spring MVC controllers & Views

SPRING - JDBC FRAMEWORK

While working with database using plain old JDBC, it becomes cumbersome to write unnecessary code to handle exceptions, opening and closing database connections etc. But Spring JDBC Framework takes care of all the low-level details starting from opening the connection, prepare and execute the SQL statement, process exceptions, handle transactions and finally close the connection.So what you have do is just define connection parameters and specify the SQL statement to be executed and do the required work for each iteration while fetching data from the database.

Spring JDBC provides several approaches and correspondingly different classes to interface with the database. I'm going to take classic and the most popular approach which makes use of JdbcTemplate class of the framework. This is the central framework class that manages all the database communication and exception handling.

JDBCTEMPLATE CLASS

The JdbcTemplate class executes SQL queries, update statements and stored procedure calls, performs iteration over ResultSets and extraction of returned parameter values. It also catches JDBC exceptions and translates them to the generic, more informative, exception hierarchy defined in the org.springframework.dao package.

Instances of the *JdbcTemplate* class are *threadsafe* once configured. So you can configure a single instance of a *JdbcTemplate* and then safely inject this shared reference into multiple DAOs. A common practice when using the JdbcTemplate class is to configure a *DataSource* in your Spring configuration file, and then dependency-inject that shared DataSource bean into your DAO classes, and the JdbcTemplate is created in the setter for the DataSource.

CONFIGURING DATA SOURCE

Let us create a database table Student in our database TEST. I assume you are working with MySQL database, if you work with any other database then you can change your DDL and SQL queries accordingly.

```
CREATE TABLE Student(
ID INT NOT NULL AUTO_INCREMENT,
NAME VARCHAR(20) NOT NULL,
AGE INT NOT NULL,
PRIMARY KEY (ID)
);
```

Now we need to supply a DataSource to the JdbcTemplate so it can configure itself to get database access. You can configure the DataSource in the XML file with a piece of code as shown below:

```
<bean
>
<property nam e="driverClassNam e" value="com .m ysql.jdbc.Driver"/>
<property nam e="url" value="jdbc:m ysql://localhost:3306/TEST"/>
<property nam e="usernam e" value="root"/>
<property nam e="password" value="password"/>
</bean>
```

DATA ACCESS OBJECT *DAO*

DAO stands for data access object which is commonly used for database interaction. DAOs exist to provide a means to read and write data to the database and they should expose this functionality through an interface by which the rest of the application will access them.

The Data Access Object *DAO* support in Spring makes it easy to work with data access technologies like JDBC, Hibernate, JPA or JDO in a consistent way.

EXECUTING SQL STATEMENTS

Let us see how we can perform CRUD *Create*, *Read*, *UpdateandDelete* operation on database tables using SQL and jdbcTemplate object.

Querying for an integer:

```
String SQL = "select count(* ) from Student";
int rowCount = jdbcTem plateObject.queryForInt( SQL );
```

Querying for a long:

```
String SQL = "select count(* ) from Student";
long rowCount = jdbcTem plateObject.queryForLong( SQL );
```

A simple query using a bind variable:

```
String SQL = "select age from Student where id = ?";
int age = jdbcTem plateObject.queryForInt(SQL, new Object[]{10});
```

Querying for a String:

```
String SQL = "select nam e from Student where id = ?";
String nam e = jdbcTem plateObject.queryForObject(SQL, new Object[]{10}, String.class);
```

Querying and returning an object:

```
String SQL = "select * from Student where id = ?";
Student student = jdbcTem plateObject.queryForObject(SQL,
new Object[]{10}, new StudentMapper());
public class StudentMapper im plem ents RowMapper<Student> {
public Student m apRow(ResultSet rs, int rowNum ) throws SQLException {
Student student = new Student();
student.setID(rs.getInt("id"));
student.setNam e(rs.getString("nam e"));
student.setAge(rs.getInt("age"));
return student;
}
}
++
```

Querying and returning multiple objects:

```
String SQL = "select * from Student";
List<Student>students = jdbcTem plateObject.query(SQL,
new StudentMapper());
public class StudentMapper im plem ents RowMapper<Student> {
```

```
public Student m apRow(ResultSet rs, int rowNum ) throws SQLException {
Student student = new Student();
student.setID(rs.getInt("id"));
student.setNam e(rs.getString("nam e"));
student.setAge(rs.getInt("age"));
return student;
}
}
```

Inserting a row into the table:

```
String SQL = "insert into Student (nam e, age) values (?, ?)";
jdbcTem plateObject.update( SQL, new Object[]{"Zara", 11} );
```

Updating a row into the table:

```
String SQL = "update Student set nam e = ? where id = ?";
jdbcTem plateObject.update( SQL, new Object[]{"Zara", 10} );
```

Deletng a row from the table:

```
String SQL = "delete Student where id = ?";
jdbcTem plateObject.update( SQL, new Object[]{20} );
```

Executing DDL Statements

You can use the execute. . method from *jdbcTemplate* to execute any SQL statements or DDL statements. Following is an example to use CREATE statement to create a table:

```
String SQL = "CREATE TABLE Student( " +
"ID INT NOT NULL AUTO_INCREMENT, " +
"NAME VARCHAR(20) NOT NULL, " +
"AGE INT NOT NULL, " +
"PRIMARY KEY (ID));"
```

jdbcTem plateObject.execute(SQL);

Spring JDBC Framework Examples:

Based on the above concepts, let us check few important examples which will help you in understanding usage of JDBC framework in Spring:

S.N. Example & Description

1 Spring JDBC Example

This example will explain how to write a simple a JDBC based Spring application.

2 SQL Stored Procedure in Spring

Learn how to call SQL stored procedure while using JDBC in Spring.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

Spring JDBC - JdbcTemplate Class

The org.springframework.jdbc.core.JdbcTemplate class is the central class in the JDBC core package. It simplifies the use of JDBC and helps to avoid common errors. It executes core JDBC workflow, leaving the application code to provide SQL and extract results. This class executes SQL queries or updates, initiating iteration over ResultSets and catching JDBC exceptions and translating them to the generic, more informative exception hierarchy defined in the org.springframework.dao package.

Class Declaration

Following is the declaration for org.springframework.jdbc.core.JdbcTemplate class −

```
public class JdbcTemplate
  extends JdbcAccessor
    implements JdbcOperations
```

Usage

Step 1 − Create a JdbcTemplate object using a configured datasource.

Step 2 − Use JdbcTemplate object methods to make database operations.

Example

Following example will demonstrate how to read a query using JdbcTemplate class. We'll read the available records in Student Table.

Syntax

```
String selectQuery = "select * from Student";

List <Student> students = jdbcTemplateObject.query(selectQuery, new StudentMapper());
```

Where,

- selectQuery − Select query to read students.
- jdbcTemplateObject − StudentJDBCTemplate object to read student object from the database.
- StudentMapper − StudentMapper is a RowMapper object to map each fetched record to the student object.

To understand the above-mentioned concepts related to Spring JDBC, let us write an example which will select a query. To write our example, let us have a working Eclipse IDE in place and use the following steps to create a Spring application.

Step	Description
1	Update the project *Student* created under chapter *Spring JDBC - First Application*.
2	Update the bean configuration and run the application as explained below.

Following is the content of the Data Access Object interface file StudentDAO.java.

```
package com.tutorialspoint;
```

```
import java.util.List;
import javax.sql.DataSource;


public interface StudentDAO {
  /**
     * This is the method to be used to initialize
     * database resources ie. connection.
  */
  public void setDataSource(DataSource ds);


  /**
     * This is the method to be used to list down
     * all the records from the Student table.
  */
  public List<Student> listStudents();
}
```

JDBC - Batch Processing

Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database.

When you send several SQL statements to the database at once, you reduce the amount of communication overhead, thereby improving performance.

- JDBC drivers are not required to support this feature. You should use the *DatabaseMetaData.supportsBatchUpdates()* method to determine if the target database supports batch update processing. The method returns true if your JDBC driver supports this feature.
- The addBatch() method of *Statement, PreparedStatement,* and *CallableStatement* is used to add individual statements to the batch. The executeBatch() is used to start the execution of all the statements grouped together.
- The executeBatch() returns an array of integers, and each element of the array represents the update count for the respective update statement.
- Just as you can add statements to a batch for processing, you can remove them with the clearBatch() method. This method removes all the statements you added with the addBatch() method. However, you cannot selectively choose which statement to remove.

Batching with Statement Object

Here is a typical sequence of steps to use Batch Processing with Statement Object −

- Create a Statement object using either *createStatement()* methods.
- Set auto-commit to false using *setAutoCommit().*
- Add as many as SQL statements you like into batch using *addBatch()* method on created statement object.
- Execute all the SQL statements using *executeBatch()* method on created statement object.
- Finally, commit all the changes using *commit()* method.

Example

The following code snippet provides an example of a batch update using Statement object −

```
// Create statement object
Statement stmt = conn.createStatement();

// Set auto-commit to false
conn.setAutoCommit(false);

// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(200,'Zia', 'Ali', 30)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create one more SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(201,'Raj', 'Kumar', 35)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create one more SQL statement
String SQL = "UPDATE Employees SET age = 35 " +
```

```
            "WHERE id = 100";

// Add above SQL statement in the batch.

stmt.addBatch(SQL);



// Create an int[] to hold returned values

int[] count = stmt.executeBatch();



//Explicitly commit statements to apply changes

conn.commit();
```

For a better understanding, let us study the Batching - Example Code

JDBC - TRANSACTIONS

If your JDBC Connection is in *auto-commit* mode, which it is by default, then every SQL statement is committed to the database upon its completion.

That may be fine for simple applications, but there are three reasons why you may want to turn off the auto-commit and manage your own transactions −

- To increase performance.
- To maintain the integrity of business processes.
- To use distributed transactions.

Transactions enable you to control if, and when, changes are applied to the database. It treats a single SQL statement or a group of SQL statements as one logical unit, and if any statement fails, the whole transaction fails.

To enable manual- transaction support instead of the *auto-commit* mode that the JDBC driver uses by default, use the Connection object's setAutoCommit() method. If you pass a boolean false to setAutoCommit( ), you turn off auto-commit. You can pass a boolean true to turn it back on again.

For example, if you have a Connection object named conn, code the following to turn off auto-commit −

```
conn.setAutoCommit(false);
```

Commit & Rollback

Once you are done with your changes and you want to commit the changes then call commit() method on connection object as follows −

```
conn.commit( );
```

Otherwise, to roll back updates to the database made using the Connection named conn, use the following code −

```
conn.rollback( );
```

The following example illustrates the use of a commit and rollback object −

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   String SQL = "INSERT INTO Employees  " +
```

```
            "VALUES (106, 20, 'Rita', 'Tez')";
  stmt.executeUpdate(SQL);
  //Submit a malformed SQL statement that breaks
  String SQL = "INSERTED IN Employees  " +
            "VALUES (107, 22, 'Sita', 'Singh')";
  stmt.executeUpdate(SQL);
  // If there is no error.
  conn.commit();
}catch(SQLException se){
  // If there is any error.
  conn.rollback();
}
```

In this case, none of the above INSERT statement would success and everything would be rolled back.

For a better understanding, let us study the Commit - Example Code.

Using Savepoints

The new JDBC 3.0 Savepoint interface gives you the additional transactional control. Most modern DBMS, support savepoints within their environments such as Oracle's PL/SQL.

When you set a savepoint you define a logical rollback point within a transaction. If an error occurs past a savepoint, you can use the rollback method to undo either all the changes or only the changes made after the savepoint.

The Connection object has two new methods that help you manage savepoints −

- setSavepoint(String savepointName): Defines a new savepoint. It also returns a Savepoint object.
- releaseSavepoint(Savepoint savepointName): Deletes a savepoint. Notice that it requires a Savepoint object as a parameter. This object is usually a savepoint generated by the setSavepoint() method.

There is one rollback (String savepointName) method, which rolls back work to the specified savepoint.

The following example illustrates the use of a Savepoint object −

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();


   //set a Savepoint
   Savepoint savepoint1 = conn.setSavepoint("Savepoint1");
   String SQL = "INSERT INTO Employees " +
               "VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees " +
               "VALUES (107, 22, 'Sita', 'Tez')";
   stmt.executeUpdate(SQL);
   // If there is no error, commit the changes.
```

```
  conn.commit();


}catch(SQLException se){
  // If there is any error.
  conn.rollback(savepoint1);
}
```

In this case, none of the above INSERT statement would success and everything would be rolled back.

Local vs. Global Transactions

Local transactions are specific to a single transactional resource like a JDBC connection, whereas global transactions can span multiple transactional resources like transaction in a distributed system.

Local transaction management can be useful in a centralized computing environment where application components and resources are located at a single site, and transaction management only involves a local data manager running on a single machine. Local transactions are easier to be implemented.

Global transaction management is required in a distributed computing environment where all the resources are distributed across multiple systems. In such a case, transaction management needs to be done both at local and global levels. A distributed or a global transaction is executed across multiple systems, and its execution requires coordination between the global transaction management system and all the local data managers of all the involved systems.

Spring Declarative Transaction Management

Declarative transaction management approach allows you to manage the transaction with the help of configuration instead of hard coding in your source code. This means that you can separate transaction management from the business code. You only use annotations or XML-based configuration to manage the transactions. The bean configuration will specify the methods to be transactional. Here are the steps associated with declarative transaction −

- We use <tx:advice /> tag, which creates a transaction-handling advice and at the same time we define a pointcut that matches all methods we wish to make transaction and reference the transactional advice.
- If a method name has been included in the transactional configuration, then the created advice will begin the transaction before calling the method.
- Target method will be executed in a *try / catch* block.
- If the method finishes normally, the AOP advice commits the transaction successfully otherwise it performs a rollback.

Let us see how the above-mentioned steps work but before we begin, it is important to have at least two database tables on which we can perform various CRUD operations with the help of transactions. Let us take a Student table, which can be created in MySQL TEST database with the following DDL −

```
CREATE TABLE Student(
  ID   INT NOT NULL AUTO_INCREMENT,
  NAME VARCHAR(20) NOT NULL,
  AGE  INT NOT NULL,
  PRIMARY KEY (ID)
);
```

Second table is Marks in which we will maintain marks for the students based on years. Here SID is the foreign key for the Student table.

```
CREATE TABLE Marks(
  SID INT NOT NULL,
  MARKS  INT NOT NULL,
  YEAR   INT NOT NULL
);
```

POSSIBLE QUESTIONS

PART-A (20 Marks)

(Q.No 1 to 20 Online Examination)

PART-B (6 Marks)

1. Describe JdbcTemplate Convenience Methods
2. Discuss Spring MVC Controllers.
3. Discuss Basic O/R Mapping
4. Explain the design goals and core concepts of SpringMVC
5. Explain the JDBC abstraction and DAO module
6. Discuss Declarative transaction Management
7. Explain Operations with JdbcTemplate
8. Discuss the following i) DispatchServlet ii) context configuration
9. Discuss the types of Transaction management supported by Spring with suitable example
10. Explain Spring MVC controllers and Views.

PART-C(10 Marks)

1. Develop a java program to Execute select Query using JDBC Connection

KARPAGAM ACADEMY OF HIGHER EDUCATION
COIMBATORE - 21
DEPARTMENT OF COMPUTER APPLICATIONS

CLASS : II MCA **BATCH : 2017-2020**

SUBJECT: ADVANCED JAVA AND SPRING

SNO	UNIT-V QUESTION	OPTION_A	OPTION_B	OPTION_C	OPTION_D	ANSWER
1	_____Jdbc Template class offers template method for batch update operations.	batchUpdate()	update()	batch()	batchUpdate()&update()	batchUpdate()&update()
2	_____ implement this interface to override the statement creation task.	PreparedStatement	PreparedStatementCreator	PreparedCreator	PreparedStatement	PreparedStatementCreator
3	_____primary interface that allows you to process the current row of the result set.	PreparedStatementSetter	PreparedStatementCreator	RowCallbackHandler	Callback	RowCallbackHandler
4	RowMapper implementation _____ can automatically map a row to a new instance of the sp	BeanPropertyRowMapper	BeanPropertyRow	BeanFactory	BeanProperty	BeanPropertyRowMapper
5	_____Method provides list of maps.	queryForList()	update	query()	batch()	queryForList()
6	_____Method to retrieve the JDBC template.	setJdbcTemplate()	getTemplate()	getJdbc()	getJdbcTemplate()	getJdbcTemplate()
7	SimpleJdbcTemplate offers a convenient batch update method in the form of _____	Vector	Set	Map	List	List
8	An unchecked conversion from List to List_____Method has a warning from the Java com	findAll()	query	update	batchupdate	findAll()
9	The direct parent class of DataAccessException is_____	RuntimeException	NestedRuntimeException	Exception	Throwable	NestedRuntimeException
10	The Spring JDBC frameworkthe state of 23505 mapped to DuplicateKeyException is _____	error code	SQL state	error code&Sql State	error	error code&Sql State
11	_____Class represents a real-world entity and its instances will be persisted to a database.	entity	persistent	entity & persistent	db	entity & persistent
12	primitive wrapper type to allow the identifier to be null is _____	java.lang.Integer	java.lang.Long	java.lang.Integer&java.lang.L	java.io	java.lang.Integer&java.lang.Long
13	A set of persistent annotations _____ define mapping metadata.	JPA	JSR	XML	SQL	JPA
14	Core Programming Elements for Different Data Access Strategie is _____	Resource	Resource Factory	Session	All of the above	all of the above
15	Interface_____ instance can be obtained from a SessionFactory instance.	Session	Session Factory	interface	factory	Session
16	Interface_____ instances can be obtained from an EntityManagerFactory instance.	Entity	EntityManager	entity&entitymanager	transaction	EntityManager
17	A session factory via dependency injection is _____	HibernateCourseDao	HibernateCourse	HibernateDao	hibernate	HibernateCourseDao
18	Property of the factory bean to load _____ Hibernate configuration file.	config.xml	config	configLocation	configuration	configLocation
19	FactoryBean to create an _____entity manager factory in the IoC container.	LocalEntityManagerFactoryBe	LocalEntityManagerFactory	LocalEntityManager	LocalContainer	LocalEntityManagerFactoryBean
20	To override some of the configurations in the JPA configuration file is _____	LocalEntityManagerFactoryBe	LocalContainerEntityManagerFa	LocalEntity	LocalContainer	LocalContainerEntityManagerFactoryBean
21	Method to retrieve the JDBC template is _____	setJdbcTemplate()	getTemplate()	getJdbc()	getJdbcTemplate()	getJdbcTemplate()
22	Transaction key property is _____	Atomicity	Consistency	Isolation	All of the above	all of the above
23	To access a database running on the Derby server to add is _____	Derby client library	Tomcat client library	Derby server library	tomcat server library	Derby client library
24	Spring's core transaction management abstraction is based on the _____interface	PlatformTransaction	PlatformTransactionManager	TransactionManager	PlatformManager	PlatformTransactionManager
25	A transaction manager is declared in the Spring IoC container as a normal bean.	TRUE	FALSE	TRUE&FALSE	NULL	TRUE
26	_____Method that allows to start a new transaction	getTransaction	commit	rollback	Transaction	getTransaction
27	_____Method to start a new transaction with that definition	getTransaction	commit	rollback	TransactionTemplate	getTransaction
28	_____ control the overall transaction management process & transaction exception handling	SpringTransactionTemplate	TransactionTemplate	Transaction	Template	TransactionTemplate
29	A transaction callback object that implements_____	TransactionCallback	TransactionCallbackWithoutResu	SpringTransactionTemplate	TransactionCallback &Trans	TransactionCallback &TransactionCallbackWithoutResult class
30	Spring offers a transaction advice that can be easily configured via the_____	rx:advice	bx:advice	tx:advice	txrx:advice	tx:advice
31	_____ method need to surround the code with a try/catch block nor declare throwing an exce	DAO	AOP	AOI	MVC	DAO
32	Named parameters are supported only in SimpleJdbcTemplate	TRUE	FALSE	TRUE&FALSE	TRUE/FALSE	TRUE
33	The return type of the _____ method will be determined by the class argument	query()	queryForObject()	update()	findAll()	queryForObject()
34	JdbcTemplate require statement _____ to be passed as an object array.	arguments	parameters	array	function	parameters
35	Named parameters are supported only in _____	MultipleJdbcTemplate	SingleJdbcTemplate	SimpleJdbcTemplate.	inhertance	SimpleJdbcTemplate.
36	The org.springframework.jdbc.core.support.JdbcDaoSupport class has a _____and_____	setDataSource()	setJdbcTemplate()	setDataSource()& setJdbcTer	setDateTemplate()	setDataSource()& setJdbcTemplate()
37	Named SQL parameters are specified by _____rather than by position.	name	class	file	array	name
38	_____framework offers a consistent data access exception-handling mechanism for its data access	Spring	bean	JDBC	Thread	Spring
39	PlatformTransactionManager is an abstract unit for _____	resource management	transaction management.	process	platform management	transaction management.
40	Session Interface _____ can be obtained from a SessionFactory instance.	Session	Instance	Factory	Independent	Instance
41	The_____ interface that allows to process the current row of the result set.	Primary	Secondary	Local	paramater	Primary
42	DI makes the_____to test the application	intermediate	easy	hard	normal	easy
43	Spring framework provides_____implementation	POJO	JMS	JDBC	JTA	POJO
44	_____provoides the JavaEE Specification	Spring framework	POJO	JDBC	MVC web	Spring framework
45	_____provides the support for caching validation, transaction and formatting	Development support	Declarative suuport	package support	validation support	Declarative suuport
46	POJO is abbrivated as	Program oriented java object	Platform oriented java object	Plain old Java Object	Process old java object	Plain old Java Object
47	MVC frame work is a	Light weight process	resource process	method process	Process	Light weight process
48	MVC abbrivated as	Monitor view controller	Model view controller	Model vehicle commision	mountain view controller	Model view controller
49	The job of the Dispatcher servlet is to take an incoming	URI	AOI	MVC	JSP	URI
50	Dispatcher servlet is based on ____configuration	Java bean	spring	Thread	JDBC	JavaBeans
51	getTransaction Method that allows you to start a	New transaction	Old transaction	rollback	Transaction	New transaction
52	In_____ methods, you neither need to surround the code with a try/catch block nor declare throv	DOA	AOI	IOC	AOP	DOA
53	The direct parent class of DataAccessException is:-	RuntimeException	NestedRuntimeException	Exception	Throwable	NestedRuntimeException
54	Session Interface whose_____ can be obtained from a SessionFactory instance.	Session	Instance	Interface	Independent	Instance

Reg. No.....

[16CAP303]

KARPAGAM UNIVERSITY

Karpagam Academy of Higher Education
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021
(For the candidates admitted from 2016 onwards)

MCA DEGREE EXAMINATION, NOVEMBER 2017

Third Semester

COMPUTER APPLICATIONS

ADVANCED JAVA AND SPRINGS

Time: 3 hours Maximum : 60 marks

PART – A (20 x 1 = 20 Marks) (30 Minutes)
(Question Nos. 1 to 20 Online Examinations)

(Part - B & C 2 ½ Hours)

PART B (5 x 6 = 30 Marks)
Answer ALL the Questions

21. a. Explain the fundamentals of Exception handling.

Or

b. Describe the concept of Thread and the method of creating and running Threads.

22. a. Discuss on the method of reading Text and reading, writing files.

Or

b. Explain the method of creating, accessing and reading from a Sequential access file.

23. a. Describe the SortedSet and NavigableSet Interfaces in Collection Interfaces.

Or

b. Discuss on the features of String Tokenizer and Date Classes.

24. a. Explain the process of Managing the Bean Lifecycle.

Or

b. Describe the concept of Aspect oriented Programming, Pointcuts, Advice and Advisors.

25. a. Discuss on Convenience methods, basic Queries and Updating with JdbcTemplate.

Or

b. Explain the importance Dispatcher Servlet and Spring MVC Controllers.

PART C (1 x 10 = 10 Marks)
CASE STUDY (Compulsory)

26. Write a suitable Java Program to demonstrate the concept of Thread Priorities, Scheduling and Synchronization and explain.
