MC5304 PROGRAMMING WITH JAVA L T P C
3 0 0 3

OBJECTIVES:

- To provide an overview of working principles of internet, web related functionalities
- To understand and apply the fundamentals core java, packages, database connectivity for computing
- To enhance the knowledge to server side programming.
- To Understand the OOPS concept & how to apply in programming.

UNIT I JAVA FUNDAMENTALS 9

Java features – Java Platform – Java Fundamentals – Expressions, Operators, and Control Structures – Classes, Methods – Inheritance - Packages and Interfaces – Boxing, Unboxing – Variable-Length Arguments (Varargs), Exception Handling.

UNIT II COLLECTIONS AND ADVANCE FEATURES 9

Utility Packages- Introduction to collection –Hierarchy of Collection framework – Generics, Array list, LL, HashSet, Treeset, HashMap – Comparators – Java annotations – Premain method.

UNIT III ADVANCED JAVAPROGRAMMING 9

Input Output Packages – Inner Classes – Java Database Connectivity - Introduction JDBC Drivers - JDBC connectivity with MySQL/Oracle -Prepared Statement & Result Set – JDBC Stored procedures invocation - Servlets - RMI – Swing Fundamentals - Swing Classes.

UNIT IV OVERVIEW OF DATA RETRIEVAL & ENTERPRISE APPLICATION DEVELOPMENT 9

Tiered Application development - Java Servers, containers –Web Container – Creating Web Application using JSP/Servlets – Web Frameworks Introduction to Spring/ Play Framework – ORM Layer – Introduction to Hibernate.

UNIT V JAVA INTERNALS AND NETWORKING 9

Java jar Files-Introspection – Garbage collection – Architecture and design – GC Cleanup process, Invoking GC, Generation in GC - Networking Basics Java and the Net – InetAddress – TCP/IP Client Sockets – URL –URL Connection – TCP/IP Server Sockets – A Caching Proxy HTTP Server – Datagrams.

TOTAL : 45 PERIODS

REFERENCES:

1. Amritendu De, “Spring 4 and Hibernate 4: Agile Java Design and Development”, McGraw-Hill Education, 2015
2. Herbert Schildt, The Complete Reference – Java 2, Ninth Edition, Tata McGraw Hill, 2014
3. Joyce Farrell, “Java Programming”, Cengage Learning, Seventh Edition, 2014 35
4. John Dean, Raymond Dean, “Introduction to Programming with JAVA – A Problem Solving Approach”, Tata Mc Graw Hill, 2014.
5. Mahesh P. Matha, “Core Java A Comprehensive Study”, Prentice Hall of India, 2011
6. R. Nageswara Rao,“Core Java: An Integrated Approach”, DreamTech Press, 2016

PART A
UNIT I

1. Define Byte Code.

Byte code is a highly optimizes set of instruction designed to be executed by the JAVA run-time system, which is called to be Java Virtual Machine(JVM). Java bytecode is the form of instructions that the Java virtual machine executes. Each bytecodeopcode is one byte in length, although some require parameters, resulting in some multi-byte instructions.

2. Define JVM.

JVM is an interpreter for bytecode. A Java program is interpreter by the JVM helps to solve the major problems associated with downloading programs over the Internet. A Java virtual machine (JVM) is a virtual machine capable of executing Java bytecode. It is the code execution component of the Java software platform.

3. Define encapsulation.

Encapsulation is a mechanism that binds together code & the data it manipulates & keeps both safe from outside interference & misuse. In other words, it is a protective wrapper that presents the code & data from being arbitrarily accessed by other code defined outside the wrappers.

4. Define Inheritance.

Inheritance is the process by which one object acquires the properties of another objects. This child inherits the properties of the parent group. Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.

5. Define Polymorphism.

Poly means "many forms" that polymorphism is a features that allows one interface to be used for a general class factions. The concept of polymorphism can be expressed as "one interface, with multiple method".

6. Define comment.

Comments are basically being used to enter a remark into a program's source file. There are 3 types of comments: (i) Single Line Comments: This begins with // & ends with the EOF. (ii) Multiple Comments: This begins with /* & ends with */. These comments with multiple lines can be given. (iii) Document Comments: This begins with /** & ends with */.

7. Briefly explain the syntax of public static void main(String a[]).

- public: This specifies that main() is accessible from outside of the class.
- Static: Specifies main()exists without any objects being defined.
- Void: Specifies that main() does not return a value.

8. Explain each words of the given statement.

- System.out.println("This is a sample line");
- System : is a name of the class that contain the object out.
- Out :is a static variable in the class System.
- Println : is a method in the object out.
-

9. What are the advantages of java comparing with other programming languages?

The advantages of java language:

- Object oriented
- dynamic
- distributed
- portable
- multithreaded language.

10. Distinguish between a class and interface.

Class	Interface
Methods in a class are declared and are implemented	Methods in an interface are declared and are not implemented (i.e) the methods do not have abody
Variable in a class can be of any type	Variable in an interface are of type final
We can't have more than one base interface	We can have any number of interface

11. What are the special features in Java that is not in C++?

- Distributed: Java programs can access data across a network.
- Compiled and interpreted: Java code we write is compiled to bytecode and interpreted when we executed the program.
- Robust: Java programs are less prone to error.
- Multithreaded: it allows multiple parts of a program to run simultaneously.
- Dynamic: maintaining different versions of an application is very easy in Java.

12. Explain the syntax of jump statement in java.

- Break - when you use the break statement in the switch, it terminate the statement sequence in a switch.
- Continue - Sometime you want to continue running the loop, but stop processing the remainder of the code in its body for this particular iteration the continue statement performs such an action.
- Return -the return statement is used to explicitly return from a method.

13. Write the syntax of iteration statement.

(i) While

```
While(condition)
{
  //body of loop
}
```

(ii) do..while

```
do
{
//body of loop
}
while(condition);
```

(iii)for:

```
for(initialization;condition;iteration)
{
        //body        }
```

14. Discuss about binary operators in java

- \+ - Addition
- \- - Subtraction
- * - Multiplication
- / - Division
- % - Modulus

15. Define Ternary operator?

Java includes a special ternary (three-way) operator that can replace certain types of if-then-else statements. This operator is the ?, general form:

expression1 ? expression2 : expression3

Here, expression1 can be any expression that evaluates to a boolean value. If expression1 is true, then expression2 is evaluated; otherwise, expression3 is evaluated.

16. What is the difference between >>= and >>>= operators?

>>= Shift right assignment-Right shift AND assignment operator for ex: C >>= 2 is same as C = C >> 2 &>>>= Shift right zero fill assignment-Zero fill right shift assignment (x = x >>> y)

17. What is Garbage Collection?

Objects are dynamically allocated by using the **new** operator, we wonder how such objects are destroyed and their memory released for later reallocation. Java handles deallocation automatically. This is called garbage collection. Garbage collection only occurs sporadically during the execution of our program. It will not occur simply because one or more objects exist that are no longer used.

18. Give an example for dynamic initialization of a variable?

```
public class MainClass {
  public static void main(String args[]) {
    double a = 3.0, b = 4.0;
    // c is dynamically initialized
    double c = Math.sqrt(a * a + b * b);

    System.out.println("Hypotenuse is " + c);
  }
}
```

19. Why java is called platform independent language?

Write once run anywhere: Java is a programming language which has its own virtual machine. When you write a program using the Java language you can use it on any operating system which has Java installed. This is different to previous programs and programming languages as they were written for individual operating systems and needed to be re-written for a different operating system.

20. What are the benefits of organizing classes into packages?

A Java package is a mechanism for organizing Javaclasses into namespaces. Java packages can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

21. What is a package in Java?

Packages are containers for classes that are used to keep the class name space compartmentalized. **General form to define a package:** Package pkg; wherepkg is the name of the package, Java uses file system directories to store package. The directory name must match the package name exactly. For Importing package: Import pkg1[.pkg2].classname.*;

22. What is the use of Throw class? Discuss with an example?

The exception class does not define any method of its own. It inherits those method provided by thowable class. Thus, all exceptions, including those that we create, have the methods defined by throwable. For example,

```
try
{  // statements
thrownewUserDefinedException( );
   // statements
}
catch (UserDefinedException e)
{
```

System.out.printIn ("User defined exception caught");
}

23.Define class. Write the structure of a class.

A class, in the context of Java, are templates that are used to create objects, and to define object data types and methods.

Structure:

```
class classname {
type instance-variable1;
type instance-variable2;
// ...
type instance-variableN;
type methodname1(parameter-list)
{ // body of method }
type methodname2(parameter-list)
{ // body of method }
// ...
type methodnameN(parameter-list)
{ // body of method }
}
```

24. Class demo

```
{
int a;
  Static int b;
}
demo t1 =new demo();
demo t2= new demo();
```

In the above program, two objects t1 and t2 are created for the class demo. How memory will be allocated for the static and non-static members of the class when the objects are created? (DEC 2014)

For static memory, only one memory can be allocated. But for non static,each individual memory can be allocated for the variable.

25.Write a java program to generate the following series : 3,8,13,18,23. (DEC 2014)

```
importjava.util.*;


class Series
{
        public staticvoid main(Stringargs[])
        {
                inta,d,n=1,t,b;


                Scanner sc = new Scanner(System.in);
                System.out.println("Enter first term of A.P");
                a=sc.nextInt();
                System.out.println("Enter difference");
                d=sc.nextInt();
```

```
System.out.println("Enter the number of terms");
b=sc.nextInt();

while(b!=n-1)
{
        t=(a+(n-1)*d);
        n++;
        System.out.print(t+" ");}}}
```

26. Define Interface.(DEC2015)

Methods in an interface are declared and are not implemented (i.e) the methods do not have a body. Variable in an interface are of type final.

27. What is meant by Exception handling?(DEC 2015)

Exception handling is the process of responding to the occurrence, during computation, of exceptions – anomalous or exceptional conditions requiring special processing – often changing the normal flow of program execution. It is provided by specialized programming language constructs or computer hardware mechanisms.

28. What is meant by abstraction? (Nov/Dec 2016)

The process of abstraction in Java is used to hide certain details and only show the essential features of the object. In other words, it deals with the outside view of an object (interface). The only good example i see for this across different sites is interface.

29.Mention the uses of interfaces. (Nov/Dec 2016)

A particular advantage of using interface in Java is that it allows multiple inheritance. The full power of Interface is utilized when dependency injection techniques is used to inject required implementation on run time.

UNIT II

1. Describe the Collections type hierarchy. What are the main interfaces, and what are the differences between them?

The *Iterable* interface represents any collection that can be iterated using the *for-each* loop.
The *Collection* interface inherits from *Iterable* and adds generic methods for checking if an element is in a collection, adding and removing elements from the collection, determining its size etc.
The *List*, *Set*, and *Queue* interfaces inherit from the *Collection* interface.
List is an ordered collection, and its elements can be accessed by their index in the list.
Set is an unordered collection with distinct elements, similar to the mathematical notion of a set.

Queue is a collection with additional methods for adding, removing and examining elements, useful for holding elements prior to processing.
Map interface is also a part of the collection framework, yet it does not extend *Collection*. This is by design, to stress the difference between collections and mappings which are hard to gather under a common abstraction. The *Map*interface represents a key-value data structure with unique keys and no more than one value for each key.

2. Describe various implementations of the *Map* interface and their use case differences.

One of the most often used implementations of the *Map* interface is the *HashMap*. It is a typical hash map data structure that allows accessing elements in constant time, or O(1), but **does not preserve order and is not thread-safe**.
To preserve insertion order of elements, you can use the *LinkedHashMap* class which extends the *HashMap* and additionally ties the elements into a linked list, with foreseeable overhead.
The *TreeMap* class stores its elements in a red-black tree structure, which allows accessing elements in logarithmic time, or O(log(n)). It is slower than the *HashMap*for most cases, but it allows keeping the elements in order according to some *Comparator*.
The **ConcurrentHashMap** is a thread-safe implementation of a hash map. It provides full concurrency of retrievals (as the *get* operation does not entail locking) and high expected concurrency of updates.
The *Hashtable* class has been in Java since version 1.0. It is not deprecated but is mostly considered obsolete. It is a thread-safe hash map, but unlike *ConcurrentHashMap*, all its methods are simply *synchronized*, which means that all operations on this map block, even retrieval of independent values.

3. Explain the difference between *LinkedList* and *ArrayList.*

ArrayList is an implementation of the *List* interface that is based on an array. *ArrayList* internally handles resizing of this array when the elements are added or removed. You can access its elements in constant time by their index in the array. However, inserting or removing an element infers shifting all consequent elements which may be slow if the array is huge and the inserted or removed element is close to the beginning of the list.
LinkedList is a doubly-linked list: single elements are put into *Node* objects that have references to previous and next *Node*. This implementation may appear more efficient than *ArrayList* if you have lots of insertions or deletions in different parts of the list, especially if the list is large.

4. What is the difference between *HashSet* and *TreeSet*?

Both *HashSet* and *TreeSet* classes implement the *Set* interface and represent sets of distinct elements. Additionally, *TreeSet* implements the *NavigableSet*interface. This interface defines methods that take advantage of the ordering of elements.
HashSet is internally based on a *HashMap*, and *TreeSet* is backed by a *TreeMap*instance, which defines their properties: *HashSet* does not keep elements in any particular order. Iteration over the elements in a *HashSet* produces them in a shuffled order. *TreeSet*, on the other hand, produces elements in order according to some predefined *Comparator*.

5. How is *HashMap* implemented in Java? How does its implementation use *hashCode* and *equals* methods of objects? What is the time complexity of putting and getting an element from such structure?

The *HashMap* class represents a typical hash map data structure with certain design choices.
The *HashMap* is backed by a resizable array that has a size of power-of-two. When the element is added to a *HashMap*, first its *hashCode* is calculated (an *int*value). Then a certain number of lower bits of this value are used as an array index. This index directly points to the cell of the array (called a bucket) where this key-value pair should be placed. Accessing an element by its index in an array is a very fast O(1) operation, which is the main feature of a hash map structure.

A *hashCode* is not unique, however, and even for different *hashCodes*, we may receive the same array position. This is called a collision. There is more than one way of resolving collisions in the hash map data structures. In Java's *HashMap*, each bucket actually refers not to a single object, but to a red-black tree of all objects that landed in this bucket (prior to Java 8, this was a linked list).

6. What is the purpose of the initial capacity and load factor parameters of a *HashMap*? What are their default values?

The *initialCapacity* argument of the *HashMap* constructor affects the size of the internal data structure of the *HashMap*, but reasoning about the actual size of a map is a bit tricky. The *HashMap*'s internal data structure is an array with the power-of-two size. So the *initialCapacity* argument value is increased to the next power-of-two (for instance, if you set it to 10, the actual size of the internal array will be 16).

The *initialCapacity* is 16 by default, and the loadFactor is 0.75 by default, so you could put 12 elements in a *HashMap* that was instantiated with the default constructor, and it would not resize. The same goes for the *HashSet*, which is backed by a *HashMap* instance internally.

7. Describe special collections for enums. What are the benefits of their implementation compared to regular collections?

EnumSet and *EnumMap* are special implementations of *Set* and *Map* interfaces correspondingly. You should always use these implementations when you're dealing with enums because they are very efficient.An *EnumSet* is just a bit vector with "ones" in the positions corresponding to ordinal values of enums present in the set. To check if an enum value is in the set, the implementation simply has to check if the corresponding bit in the vector is a "one", which is a very easy operation. Similarly, an *EnumMap* is an array accessed with enum's ordinal value as an index. In the case of *EnumMap*, there is no need to calculate hash codes or resolve collisions.

8. What is the difference between fail-fast and fail-safe iterators?

Fail-fast iterators (those returned by *HashMap*, *ArrayList*, and other non-thread-safe collections) iterate over the collection's internal data structure, and they throw *ConcurrentModificationException* as soon as they detect a concurrent modification.

Fail-safe iterators (returned by thread-safe collections such as *ConcurrentHashMap*, *CopyOnWriteArrayList*) create a copy of the structure they iterate upon. They guarantee safety from concurrent modifications. Their drawbacks include excessive memory consumption and iteration over possibly out-of-date data in case the collection was modified.

9. How can you use *Comparable* and *Comparator* interfaces to sort collections?

The *Comparable* interface is an interface for objects that can be compared according to some order. Its single method is *compareTo*, which operates on two values: the object itself and the argument object of the same type. For instance, *Integer*, *Long*, and other numeric types implement this interface. *String* also implements this interface, and its *compareTo* method compares strings in lexicographical order.

The *Comparable* interface usually is implemented using natural ordering of the elements. For instance, all *Integer* numbers are ordered from lesser to greater values. But sometimes you may want to implement another kind of ordering, for instance, to sort the numbers in descending order. The *Comparator* interface can help here.

As the *Comparator* interface is a functional interface, you may replace it with a lambda expression, as in the following example. It shows ordering a list using a natural ordering (*Integer*'s *Comparable* interface) and using a custom iterator (*Comparator<Integer>* interface).

10. What is Difference between Hashtable and HashMap in Java?

This Java collection interview question is I guess most popular one. Most of Java programmer who has at least 2 years of experience has seen this question on core Java or J2EE interview. Well, there is much difference between them but most important is thread-safety, HashMap is not thread-safe while Hashtable is a thread-safe collection. See Hashtable vs HashMap in Java for

more differences between HashMap and Hashtable in Java.

11. What is the difference between Hashtable and ConcurrentHashMap in Java?

Another frequently asked Java collection interview question post-Java 5 world which introduced Concurrent Collection classes like ConcurrentHashMap and CopyOnWriteArrayList along with Concurrency utilities e.g. CyclicBarrier and CountDownLatch. Well, both Hashtable and ConcurrentHashMap are thread-safe here but later provides more scalability than former. See Difference between ConcurrentHashMap and Hashtable in Java for the answer of this Java collection interview question.

12. What is Difference between Iterator and Enumeration in Java?

One of the classic interview Questions asked on Java collection framework, This is pretty old and programmer who has been working in Java for 4 to 6 years must have seen this question before. Well, Iterator and ListIterator in Java is a new way to iterator collection in Java and provides the ability to remove an object while traversing while Enumeration doesn't allow you to remove the object. See Iterator vs Enumeration in Java for more differences between both of them.

13. What is Difference between fail-safe and fail-fast Iterator in Java?

This is relatively new Java collection interview question because the concept of a fail-safe iterator is come along with ConcurrentHashMap and CopyOnWriteArrayList. See Difference between fail-safe and fail-fast Iterator in Java for the answer of this Java collection question.

14. How HashMap works internally in Java?

One of the most frequently asked Java interview question to experience Java programmer of 4 to 5 years of experience. I have seen this question on big companies like Morgan Stanley, JP Morgan, Nomura and other banks e.g. Barclays Capital. See How HashMapworks internally in Java for detailed answer of this Java collection interview question.

15. Can you write code to traverse Map in Java on 4 ways?

Another Java collection question which appears as part of Java Coding interview question and appeared in many interviews. As you know there are multiple ways to traverse or iterate Map in Java e.g. for loop, while loop using Iterator etc. 4 ways to iterator Map in Java has detailed explanation and sample code which is sufficient to answer this Java collection framework interview question.

16. What is the difference between Vector and ArrayList in Java?

Along with Difference between HashMap and Hashtable, this Java collection interview question is probably second in the list of frequently asked question on Java collection framework. Both ArrayList and Vector implements List interface from Java 4 but they have differences including synchronization, See the difference between Vector and ArrayList in Java for the complete answer of this collection interview question in Java.

17. What is the difference between ArrayList and LinkedList in Java?

A follow-up question which is asked in response to previous Java collection interview question. Here also both LinkedList and ArrayList are List implementation but their internal data-structure is different, one is derived from Array while other is derived from LinkedList. See LinkedList vs ArrayList in Java to answer this Java Collection interview question.

18. What is the difference between List and Set in Java?

List vs Set is one of the most important concepts to understand in Java Collection framework and this Java collection interview question focus on that. Most important difference between them is that List allows duplicates and maintain insertion order while Set doesn't allow duplicates and doesn't maintain any order. See Difference between Set and List in Java to see more differences between them.

19. How do you find if ArrayList contains duplicates or not?

Since List allows duplicates this becomes a followup question of earlier Java collection framework interview question. See How to check if ArrayList contains duplicates or not for the answer of this Java collection question.

20. Does not overriding a hashCode() method have any performance implication?

A poor hashCode function will result in frequent collisions in HashMap, which eventually increases the time for adding an object into said HashMap.From Java 8 onwards, though, collisions will not impact performance as much as in earlier versions because, after a threshold, the linked list will be replaced by the binary tree, which will give you **O(log N)** performance in the worst case, as compared to **O(N)** of a linked list.

UNIT III

1. Define RMI and List out the advantages of RMI(DEC 2015)

RMI is focused on java, with connectivity to existing systems using native methods. Thus RMI is a natural, direct and is fully-powered to provide us with a distributed computing technology.

- Object Oriented
- Distributed Garbage Collection
- Parallel computing
- Safe & Secure.

2. What is a stub ?

A Stub is a java object that resides on the client machine. Its function is to present the same interfaces as the remote server. Once you create the stub, you define its behavior in the stub data table or you enter code in the user code class that is associated with the stub.

3. List out the steps to run simple Client / Server application using RMI

Server side program: (i) create a interface (ii) create a class which implement the interface (iii) create a class which create a object for the class which implements the interface.

Client side program: (i) create a program implements a client side of this application. (ii) Generate stubs and skeletons. (iii) install files on the client and server machines. (iv) start the RMI registry on the server machine (v) Start the server (vi) Start the client.

4. What are the disadvantages of RMI?

- The most popular browsers like internet Explorer ant Netscape Navigator do not support RMI fully.
- While accepting the connections on the internet, rmi registry experiences some problem which requires the registry to be shutdown and started again.
- In environment prior to java 1.2.2, server sockets that handle the connections from rmi clients can live on indefinitely even after the clients have shut down.
- RMI registry allows client connections to be made for only one specific hostname. This means that server application cannot support multiple domains on a single web server which limits the deployment options.

5. Write short notes on JDBC.

JDBC standard is intented for people developing industrial-strength database applications.JDBC makes java effective for developing enterprise information system.java.sql is the JDBC package that contains classes & interfaces that enable a java program to interact with a database.

6. Write short notes on JDBC drivers.

A JDBC driver is basically an implementation of the function calls specified in the JDBC API for a particular vendor's RDBMS. Hence, a java program with JDBC function calls can access any RDBMS that has a JDBC driver available. A driver manager is used to keep track of all the installed drivers on the system. The operations of driver manager are getDriver, registerDriver, deregisterDriver.

7. What are the advantages of servlet over CGI?

- Performance is significantly better, servlet execute within the address space of a web server.
- Servlets are platform independent
- The java security manager on the server enforces a set of restrictions to protect the resources on a server machine.

- The full functionality of java class libraries is available to a servlet.

8. Write down the methods of servlet interface

void destroy() –called when the servlet is unloaded.

ServletConfig getServletConfig() –returns a ServletConfig object that contains any initialization parameters.

String get ServletInfo() – returns a string describing the servlet.

void init(ServletConfig sc) throws ServletException –called when the servlet is initialized .Initialization parameters for servlet can be obtained from sc. An unavailable exception should be thrown if the servlet is not initialized.

Void Service(ServletRequest req,ServletResponse res) throws ServletException, IOException-Called to process a request from a client. The request from the client can be read from req. response to the client can be written to res.

9. What are the advantages of multicasting?

- The first major advantage of using multicasting is the decrease of the network load.
- Multicasting can be very helpful is in resource discovery. There are many applications in which a host to find out whether a certain type of service is available or not
- It support for data casting applications.
- Multicasting is flexibility in joining and leaving a group provided by multicasting can make the variable membership much easier to handle.

10. What is multicasting?

Multicasting is the internet version of broadcasting .It is similar in many ways to a television or radio station that broadcasts its signal. The signal originates from one source,but it can reach everyone in the station's signal area. The information passes by on those who do not want to catch the signal or do not have the right equipment.

11. Which IP address is used for mullticasting?

Class D IPV4 is used for multicasting. The address range is 224.0.0.1 to 239.255.255.255

12. Define TTL.

Broadcast packets has a finite life is called TTL(Time To Live) in order to avoid bouncing of the packets around the network. Time-to-live (TTL) is a value in an Internet Protocol (IP) packet that tells a network router whether or not the packet has been in the network too long and should be discarded.

13. What is an inner class? Explain with an example? (DEC 2015)

A class within the class is called as an inner class.

```
class A
{   int a, b,c;
   class B
    {          // coding for inner class
}}
```

14. What is the difference between CGI and servlets?

- Performance is significantly better, servlet execute within the address space of a web server.
- Servlets are platform independent
- The java security manager on the server enforces a set of restrictions to protect the resources on a server machine.
- The full functionality of java class libraries is available to a servlet.

15. What is the purpose of Servlets?

Usually a servlets is used to develop web applications in a web server. The servlets are used to create web pages which are called dynamic web pages which mean the content of a web page can change according to the input sent from the web client. Servlets are server independent and platform independent.

16. Define serialization with an example?

Serialization is the process of writing the state of an object to a byte stream. At a later time, you may restore these objects by using the process of deserialization. Interfaces and classes support serialization

17. Define Servlet Life Cycle?

- **init( )** method - invoked when the servlet is first loaded into memory
- **service( ) -** called for each HTTP request (for processing)
- **destroy( ) -** unloads the servlet from its memory.

18. How is anonymous inner class used for Event handling?

```
// Anonymous inner class demo.
import java.applet.*;
import java.awt.event.*;
/* <applet code="AnonymousInnerClassDemo" width=200 height=100> </applet> */
public class AnonymousInnerClassDemo extends Applet
 {
public void init() {
addMouseListener(new MouseAdapter()
{
public void mousePressed(MouseEvent me)
{
showStatus("Mouse Pressed");
}}
}
}
```

19. What is the format of inner class after it is compiled?

OuterClass$InnerClass.class

20. What is MIME?

Multipurpose Internet Mail Extensions (MIME) is an Internet standard that extends the format of email to support:

- Text in character sets other than ASCII
- Non-text attachments
- Message bodies with multiple parts
- Header information in non-ASCII character sets

21. What are the two types of I/O streams?

Byte streams provide a convenient means for handling input and output of bytes. Byte streams are used, when reading or writing binary data. ***Character streams*** provide a convenient means for handling input and output of characters.

22. Distinguish between Applets and Servlets?

Applet is an application designed to be transmitted over the internet and executed by a java compatible web browser. Applets dynamically extend the functionality of a Web browser. Servlets are applet program written on the server side. Servlets are small programs that execute on the server side of a Web connection. Servlets dynamically extend the functionality of a Web server.

23. Explain how to handle character arrays in Java programs?

CharArrayReader is an implementation of an input stream that uses a character array as the source.

(i) CharArrayReader(char array[ ])

(ii) CharArrayReader(char array[ ], int start, int numChars) where ***array*** is the input source and the second constructor creates a **Reader from a subset** of your character array that begins with the character at the index specified by **start** and is **numChars** long.

24. Define Swing?

Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT. Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and, therefore, are platform-independent. The term lightweight is used to describe such elements.

25. Explain swing classes?

Class	Description
AbstractButton	Abstract superclass for Swing buttons.
ButtonGroup	Encapsulates a mutually exclusive set of buttons.
ImageIcon	Encapsulates an icon.
JApplet	The Swing version of Applet.
JButton	The Swing push button class.
JCheckBox	The Swing check box class.
JComboBox	Encapsulates a combo box (an combination of a drop-down list and text field).
JLabel.	The Swing version of a label
JRadioButton	The Swing version of a radio button.
JScrollPane	Encapsulates a scrollable window.
JTabbedPane	Encapsulates a tabbed window.
JTable	Encapsulates a table-based control.
JTextField	The Swing version of a text field.
JTree	Encapsulates a tree-based control.

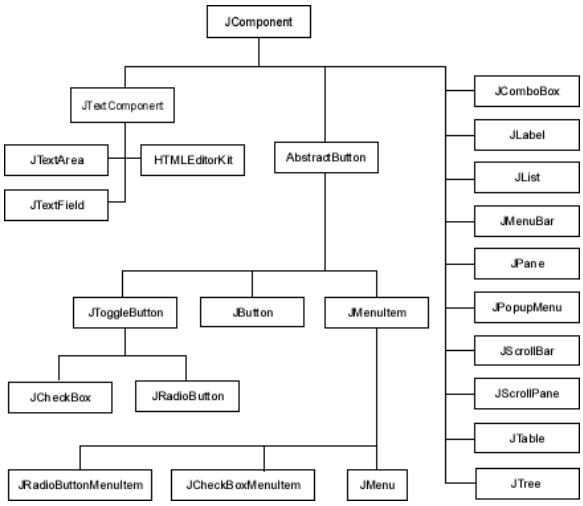
26. Specify the difference between "Get" and "Post" methods in Java servlets. (DEC 2014)

GET and POST basically allow information to be sent back to the webserver from a browser (or other HTTP client for that matter).

Imagine that you have a form on a HTML page and clicking the "submit" button sends the data in the form back to the server, as "name=value" pairs.

Choosing GET as the "method" will append all of the data to the URL and it will show up in the URL bar of your browser. The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.

27. Draw java swing class hierarchy.(DEC 2014)



28. List any four interfaces defined by java.util. (Nov/Dec 2016)

Abstract Interfaces, strictfp Interfaces, Generic Interfaces and Type Parameters, Superinterfaces and Subinterfaces, Enumeration interfaces defines method to enumerate through collection of object.

29.Mention any two features of Swing. (Nov/Dec 2016)

Lightweight Components

Swing components are lightweight as they are written entirely in Java and do not depend on native peers (platform specific code resources). Rather, they use simple drawing primitives to render themselves on the screen.

Pluggable Look and Feel

The pluggable look arid feel feature allows us to tailor the look and feel of the application and applets to the standard looks like, Windows and Motif. We can even switch to different look and feel at runtime.

UNIT IV

1. What is considered as a web component?

Java Servlet and Java Server Pages technology components are web components. Servlets are Java programming language that dynamically receives requests and makes responses. JSP pages execute as servlets but allow a more natural approach to creating static content.

2. What is JSF?

JavaServer Faces (JSF) is a user interface (UI) designing framework for Java web applications. JSF provides a set of reusable UI components, a standard for web applications. JSF is based on MVC design pattern. It automatically saves the form data to the server and populates the form date when display on the client side.

3. What is Hibernate?

Hibernate is an open source object-relational mapping and query service. In hibernate we can write HQL instead of SQL which save developers to spend more time on writing the native SQL. Hibernate has a more powerful association, inheritance, polymorphism, composition, and collections. It is a beautiful approach for persisting into the database using the Java objects. Hibernate also allows you to express queries using Java-based criteria.

4. What is the limitation of hibernate?

- Slower in executing the queries than queries are used directly.
- Only query language support for composite keys.
- No shared references to value types.

5. What are the advantages of hibernate?

- Hibernate is portable i mean database independent, Vendor independence.

- Standard ORM also supports JPA
- Mapping of the Domain object to the relational database.
- Hibernate is better than plain JDBC.
- JPA provider in JPA based applications.

6. What is ORM?

ORM stands for Object-Relational mapping. The objects in a JAVA class which is mapped into the tables of a relational database using the meta data that describes the mapping between the objects and the database. It works by transforming the data from one representation to another.

7. Differentiate between save and saveorupdate.

Save()-This method in Hibernate is used to store an object in the database. It inserts an entry if the record doesn't exist, otherwise not.

Saveorupdate()- This method in the hibernate is used for updating the object using identifier. If the identifier is missing this method calls save(). If the identifier exists, it will call update method.

8. How to invoke a stored procedure in hibernate?

{? = call thisISTheProcedure() }

9. What are the benefits of ORM?

- Productivity
- Maintainability
- Performance
- Vendor Independence

10. What are the core interfaces of Hibernate framework?

- Session interface
- Session factory interface
- Configuration interface
- Transaction interface
- Query and Criteria interface

11. What is the file extension used for hibernate mapping file?

The name of the file should be like this: filename.hbm.xml

12. What is the Hibernate proxy?

An object proxy is just a way to avoid retrieving an object until you need it. Hibernate 2 does not proxy objects by default.

13. What is HQL?

Hql STANDS FOR Hibernate Query Language. Hibernate allows to the user to express queries in its portable SQL extension, and this is called as HQL. It also allows the user to express in native SQL.

14. What are the collections types in Hibernate?

Set, List, Array, Map, Bag are collection type in Hibernate.

15. Differentiate between .ear, .jar and .war files.

.jar files: These files are with the .jar extension. The .jar files contain the libraries, resources and accessories files like property files.

.war files: These files are with the .war extension. The .war file contains JSP, HTML, javascript and other files necessary for the development of web applications.

.ear files: The .ear file contains the EJB modules of the application.

16. What are the different modules in spring?

There are seven core modules in spring

- The core container module
- O/R mapping module
- DAO module
- Application context module
- Aspect oriented Programming

- Web module
- MVC module

17. What are the benefits of Spring Framework?

- Lightweight container
- Spring can effectively organize your middle tier objects
- Initialization of properties is easy. No need to read from a property file
- Application code is much easier to unit test
- Objects are created lazily, singleton-configuration
- Spring's configuration management services can be used in any architectural layer

18. What is Play Framework?

Play Framework makes it easy to build scalable, fast and real-time web applications with Java and Scala. In other words Play Framework is a core offering of the Type safe Reactive Paltform. It is a web app framework, written in Scala and Java that provides iterative and Reactive application development very simple. The Play is a clean alternative to the legacy Enterprise Java stacks.

19. What do you mean by the Java Collection Framework?

Collections are utilized in various programming languages and basic release contained some classes for collections such as Vector, Stack, Hashtable, and Array. But the more scope and uses Java 1.2 came up with collections Framework the collections interfaces, implementations and different type algorithms.

20. What are the advantages of Collections Framework?

Some of the advantages of collections framework are:

- The reduce development effort by using core collection classes rather than defining collection classes.
- The code quality is improved with the use of completely tested collections framework classes.
- Reduce some effort for code maintenance by utilizing collection classes with the JDK.
- Collection framework also provides reusability and Interoperability.

UNIT V

1. Define Java jar Files

A JAR (Java Archive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file to distribute application software or libraries on the Java platform.
In other words, a JAR file is a file that contains compressed version of .class files, audio files, image files or directories. We can imagine a .jar files as a zipped file(.zip) that is created by using WinZip software. Even , WinZip software can be used to used to extract the contents of a .jar

2. How to create Java jar files?

To create a .jar file , we can use jar cf command in the following way:

Jar cfjarfilenameinputfiles

Here, cf represents create the file. For example , assuming our package pack is available in C:\directory

3. How to view a Java jar files?

Viewing a JAR file: To view the contents of .jar files, we can use the command as:

Jar tfjarfilename

Here ,tf represents table view of file contents. For example, to view the contents of our pack.jar file ,

4. Define Introspection.

Introspection is the automatic process of analyzing a bean's design patterns to reveal the bean's properties, events, and methods. This process controls the publishing and discovery of bean operations and properties.

5. What are the different advantages in Java Introspection?

Introspection is an automatic process in which the bean's design patterns are analyzed to extract the bean's properties, events, and methods. Introspection has some great advantages as under:

- Portability.
- Re-usability.

6. What are the Different classes in Java Introspection?

BeanDescriptor, BeanInfo, FeatureDescriptor, EventSetDescriptor
MethodDescriptor, PropertyDescriptor, IndexedPropertyDescriptor
Introspector ,SimpleBeanInfo

7. What is meant by garbage collection?

Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program.

8. What are the advantages of Java Garbage Collection?

The biggest benefit of Java garbage collection is that it automatically handles deletion of unused objects or objects that are out of reach to free up vital memory resources. Programmers working in languages without garbage collection (like C and C++) must implement manual memory management in their code.

9.Define Java virtual Machine.

JVM(Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the **main** method present in a java code. JVM is a part of JRE(Java Run Environment).

10.What is meant by Java Virtual machine?

A virtual machine (VM) *is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine*. Originally, Java was designed to run based on a virtual machine separated from a physical machine for implementing **WORA** (*Write Once Run Anywhere*), although this goal has been mostly forgotten

11.What are the main features of Java Virtual Machine?

Stack-based virtual machine, Symbolic reference, Garbage collection, Guarantees platform independence by clearly defining the primitive data type, Network byte order

12. What is meant by Java Byte Code?

To implement WORA, the JVM uses Java bytecode, a middle-language between Java (user language) and the machine language. This Java bytecode is the smallest unit that deploys the Java code.

13. What are sockets?

A socket is an endpoint for communication. Socket is an abstraction for an endpoint of communication that can be manipulated with a file descriptor. Scope of a socket is the communication domain in which it exists. A socket address is the combination of an IP address and a port which is mapped to the application program process.

14.How do you create a socket?

Socket is created using the socket( ) system call. The syntax is as follows

sockfd = socket(int family, int type, int protocol).

*sockfd r*efers to file descriptor for created socket and *family* represents address family
It can be AF_INET (IPV4) AF_INET6 (IPV6). Here type represents the socket type.

15.What is the use of recv() function?

This function is used to receive data from remote peer. The syntax is

*recv(int sockfd, char *buffer, int nbytes, int flags)*

Argument*sockfd* represents the socket file descriptor to read from, *buffer* represents the address of buffer to read data into and nbytes refers the number of bytes to read.

16.Give the use of bind method

*int bind(intsockfd,conststructsockaddr*myaddr,socklen_taddrlen);*

This function is used to assign a local protocol address ("name) to a socket. This association with an address must be performed with the bind() system call before the socket can accept connections to other hosts.

17. Give a brief note on accept( ) function

This function Wait for and accept an incoming connection. It creates a new socket for the accepted connection (listen socket continues "listening"). The syntax is

*newsockfd = accept(intsockfd, structsockaddr_in *peername, intpeername_len)*

newsockfd refers to sockfd for newly accepted socket, *sockfd*represents the listening socket descriptor and *peername* is the socket address of the connected peer process.

18. Define Copy-On-Write

Instead of making a complete copy of parent's dataspace, heap, stack,parent& child can share these regions(read-only copy). If either of the process tries to modify this region, the kernel then makes a copy of that memory alone.

19. Give the characteristics of connectionless (datagram) network.

In connectionless (datagram) network, no dedicatedpath is required between two nodes. Each packet contains the full source and destination address. Each packet is routed independently. It is suitable for dynamic bandwidth environment.

20. Write short note son java Inetaddress.

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.javatpoint.com, www.google.com, www.facebook.com etc.

21.Write an example of Inet address.

```
import java.io.*;
import java.net.*;
public class InetDemo{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.javatpoint.com");

System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
}catch(Exception e){System.out.println(e);}
}
}
```

PART B

UNIT I

1. Explain the features of Java.

Compiled & Interpreted

A computer language is either compiled or interpreted. Java combines both these approaches together to make java as a two stage systems.

- Simple
- Object oriented
- Platform independent & Portable
- Multithreading
- High Performance:
- Robust & Secure
- Distributed

2. Explain package in detail with example.

- Packages are containers for classes that are used to keep the class name space separately.
- A unique name is to be used for each class to avoid name collisions.

Defining a package:

- To create a package, include a package command as the first statement in java source file.
- The ***package statement*** defines a name space in which classes are stored.
  Syntax: package Mypackage; Here Mypackage is the name of the package.
- We can have a hierarchy of packages.
  Syntax:package pkg1[.pkg2][.pkg3];

Inside pack1 folder:

```
package pack1;
public class student
{
 String name;
 String course;
 int tot, avg;
 public void initialise(String s1,String c)
 {
  name=s1;
  course=c;
 }
 public void calc(int m1,int m2,int m3)
 {
  tot=m1+m2+m3;
  avg=tot/3;
 }
 public void display()
 {
  System.out.println("\nStudent name : " +name);
  System.out.println("\nCourse       : " +course);
  System.out.println("\nTotal        : " +tot);
  System.out.println("\nAverage      : " +avg);

 }
}
```

MainClass(outside pack1)

```
import java.io.*;
import pack1.*;
class packeg extends student
{
 public static void main(String args[])throws IOException
 {
  int i;
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
  String s,s1,s2;
```

```
 int m1,m2,m3;
 int no;
 System.out.println("\nEnter the number of students :");
 s=br.readLine();
 no=Integer.parseInt(s);
 for(i=1;i<=no;i++)
{
 System.out.println("\nEnter Name:");
 s1=br.readLine();
 System.out.println("\nEnter Course :");
 s2=br.readLine();
 packeg p=new packeg();
 p.initialise(s1,s2);
 System.out.println("\nEnter the marks :");
 m1=Integer.parseInt(br.readLine());
 m2=Integer.parseInt(br.readLine());
 m3=Integer.parseInt(br.readLine());
 p.calc(m1,m2,m3);
 p.display();
 }
 }
}
```

3. Explain interface in detail with example.(DEC2014)

Defining Interfaces:

- ❖ It is basically a kind of class. Like classes, interfaces contain methods and variables.
- ❖ But the interface defines only abstract methods and final fields.

Syntax: interface interfacename {
variable declaration;
method declaration; }

Extending interfaces:

- ➢ Like classes, interfaces can also be extended.
- ➢ The new sub interface will inherit all the members of the super interface.
- ➢ This is done by extends keyword.

Implementing interfaces:

Interfaces are used as "super classes" whose properties are inherited by classes.

```
class class-name implements interface-name
  {     Body of class-name    }
```

The class class-name implements the interface interface-name.

```
interface area        //Interface defined
  {
        final static float pi=3.14;
        float compute(float x,float y);
  }
Class rectangle implements area
  {
        public float compute(float x,float y)
          {
                return(x*y);
```

```
        }
  }
Class circle implements area
  {
        public float compute(float x,float y)
          {
                return(pi*x*x);
           }
  }
class test
  {
        public static void main(String args[])
          {
                rectangle r=new rectangle();
                circle c=new circle();
                area a;       //Interface object
                a=r;
                System.out.println("Area of rect:"+a.compute(10,20));
                a=c;
                System.out.println("Area of circle:"+a.compute(10,0));

           }
   }
```

4. Illustrate with an example how exception handling is done in Java.

i) Exception Handling

- A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code.
- When an exceptional condition arises, an object representing that 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.
- Exceptions can be generated by the Java run-time system, or they can be manually generated by your code. Manually generated exceptions are typically used to report some error condition to the caller of a method.

Five keywords

try, catch, throw, throws and finally.

General form of an exception-handling block

Types of Exception: (i) Built – in Exception (ii) User defined exception

```
class Exc2 {
public static void main(String args[ ]) {
int d, a;
try { // monitor a block of code.
d = 0;
a = 42 / d;
System.out.println("This will not be printed.");
} catch (ArithmeticException e) {// catch divide-by-zero error
System.out.println("Division by zero.");
}
System.out.println("After catch statement.");
```

```
}
}
```

ii) What are the uses of keyword final in java?

- A java variable can be declared using the keyword final. Then the final variable can be assigned only once.
- A variable that is declared as final and not initialized is called a blank final variable. A blank final variable forces the constructors to initialise it.
- Java classes declared as final cannot be extended. Restricting inheritance!
- Methods declared as final cannot be overridden. In methods private is equal to final, but in variables it is not.
- final parameters – values of the parameters cannot be changed after initialization. Do a small java exercise to find out the implications of final parameters in method overriding.

```
class FinallyDemo {

// Through an exception out of the method.

static void procA() {
try
{
System.out.println("inside procA");
throw new RuntimeException("demo");
}
finally
{
System.out.println("procA's finally");
}
}
// Return from within a try block.
static void procB() {
try
{
System.out.println("inside procB");
return;
}
finally
{
System.out.println("procB's finally");
}
}
// Execute a try block normally.
static void procC() {
try
{
System.out.println("inside procC");
}
finally
{
System.out.println("procC's finally");
```

```
}
}
public static void main(String args[]) {
try
{
procA();
}
catch (Exception e)
{
System.out.println("Exception caught");
}
procB();
procC();
}
}
```

5. Write a java program to perform the following:

(i) To find the transpose of a given matrix 'A'

```
public class Trans {
        public static void main(String args[]) {
    // initialize the variable
    int[][] a = { { 5, 1, 1 }, { 3, 6, 0 }, { 0, 5, 9 } };
    // print the matrix
    for (int i = 0; i < 3; i++) {
       for (int j = 0; j < 3; j++) {
          System.out.print(a[i][j]);
    }
       System.out.println();
     }
    // operation for transpose
    for (int i = 0; i < 3; i++) {
       for (int j = i + 1; j < 3; j++) {
          int temp = a[i][j];
          a[i][j] = a[j][i];
          a[j][i] = temp;
       }
       System.out.println();
    }
    System.out.println("Transpose matrix:");
    // After Transpose the matrix print the result
    for (int i = 0; i < 3; i++) {
       for (int j = 0; j < 3; j++) {
```

```
            System.out.print(a[i][j]);
        }
        System.out.println();
    }
  }
}
```

(ii) to find the sum of the diagonal elements for a given matrix 'B'

```
import java.io.*;
import java.lang.*;
public class Sum_Diagonal
{
        public static void main(String args[])throws IOException {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter the size of 2D array :");
        int i=Integer.parseInt(br.readLine());
        int d[][]=new int;
        int j,k;
        int sum1=0,sum2=0;
        BufferedReader br1=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter the values of 2D array of "+i+" * "+i+" matrix ");
        for(j=0;j<i;j++) {
        for(k=0;k<i;k++)   {
        d[j][k]=Integer.parseInt(br1.readLine());          }
        System.out.println(); }
        for(j=0;j<i;j++)     {
        for(k=0;k<i;k++)
        System.out.print(d[j][k]+" ");
        System.out.println() ; }
        for(j=0;j<i;j++)     {
        sum1=sum1+d[j][j];           }
        k=i-1;
        for(j=0;j<i;j++)     {
        if(k>=0)  {
        sum2=sum2+d[j][k];
        k--; } }
        System.out.println("Sum of Digonal elements are  :"+sum1+" "+sum2);       } }
```

6. Write Java program to Check whether the given matrix is upper triangular or not.

```
import java.io.*;
class uppertri
{
void upper( )

{
int count;
int d[][] = { { 1, 2, 6 }, { 3, 8, 5 }, { 5, 6, 7 } };
```

```
        int k = 0, j = 0;

        int sum1 = 0, sum2 = 0;

        for (j = 0; j < d.length; j++)
            {
                for (k = 0; k < d.length; k++)

                 System.out.print(d[j][k] + " ");

                 System.out.println();

            }
    for(i = 0; i <= d.length; i++)

    {
    for(j = 0; j <= d.length; j++)

    {
    if(i <= j)

    {
    if(d[i][j] == 0)

    count++;
    }
    }
    }
    if(count == 3)

    System.out.println("\nThe Matrix is upper triangular\n");

    else
    System.out.println ("\nThe Matrix is not upper triangular\n");
    }
    }
public static void main(String args[])
{
classtri c = new classtri( );
c.upper( );
```

7. (i) Describe the different levels of access protection available in Java.

Java addresses four categories of visibility for class members:

- Subclasses in the same package
- Non-subclasses in the same package
- Subclasses in different packages
- Classes that are neither in the same package nor subclasses

- Anything declared **public** can be accessed from anywhere
- Anything declared **private** cannot be seen outside of its class.

❖ When a member does not have an explicit access specification, it is visible to subclasses as well as to other classes in the **same package**. This is the **default access**.

(ii) Name and explain the uses of various bit–wise operators in Java.

Operator	Name	Example	Result	Description
`a & b`	and	3 & 5	1	1 if both bits are 1.
`a | b`	or	3 | 5	7	1 if either bit is 1.
`a ^ b`	xor	3 ^ 5	6	1 if both bits are different.
`~a`	not	~3	-4	Inverts the bits.
`n << p`	left shift	3 <<< 2	12	Shifts the bits of n left p positions. Zero bits are shifted into the low-order positions.

8. (i) Explain the operators in Java for arithmetic and logical shift operations. Show using a Java program how multiplication by 2 can be implemented using a logical shift operation (DEC 2015)

Operators
Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.

With higher precedence are evaluated before operators with relatively lower precedence. Operators on the same line have equal precedence. When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first.

Assignment operators are evaluated right to left.

Operator Precedence

Operators	**Precedence**
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* - *expr* ~ !
multiplicative	* / %
additive	+ -

Control Statements
The statements inside your source files are generally executed from top to bottom, in the order that they appear. *Control flow statements*, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to *conditionally* execute particular blocks of code. This section describes the decision-making statements (if-then, if-then-else, switch), the looping statements (for, while, do-while), and the branching statements (break, continue, return) supported by the Java programming language.

The if-then and if-then-else Statements

The if-then Statement

The if-then statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code *only if* a particular test evaluates to true. For example, the Bicycle class could allow the brakes to decrease the bicycle's speed *only if* the bicycle is already in motion. One possible implementation of the applyBrakes method could be as follows:

```
void applyBrakes() {
    // the "if" clause: bicycle must be moving if
    (isMoving){
        // the "then" clause: decrease current speed
        currentSpeed--;
    }
}
```

If this test evaluates to false (meaning that the bicycle is not in motion), control jumps to the end of the if-then statement.

In addition, the opening and closing braces are optional, provided that the "then" clause contains only one statement:

```
void applyBrakes() {
    // same as above, but without braces if
    (isMoving)
        currentSpeed--;
}
```

Deciding when to omit the braces is a matter of personal taste. Omitting them can make the code more brittle. If a second statement is later added to the "then" clause, a common mistake would be forgetting to add the newly required braces. The compiler cannot catch this sort of error; you'll just get the wrong results.

The if-then-else Statement

The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false. You could use an if-then-else statement in the applyBrakes method to take some action if the brakes are applied when the bicycle is not in motion. In this case, the action is to simply print an error message stating that the bicycle has already stopped.

```
void applyBrakes() { if
    (isMoving) {
        currentSpeed--; }
    else {
```

```
            System.err.println("The bicycle has already stopped!");
        }
}
```

The following program, IfElseDemo, assigns a grade based on the value of a test score: an A for a score of 90% or above, a B for a score of 80% or above, and so on.

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

        int testscore = 76; char
        grade;

        if (testscore >= 90) { grade
            = 'A';
        } else if (testscore >= 80) { grade =
            'B';
        } else if (testscore >= 70) { grade =
            'C';
        } else if (testscore >= 60) { grade =
            'D';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

The output from the program is:

```
Grade = C
```

You may have noticed that the value of testscore can satisfy more than one expression in the compound statement: 76 >= 70 and 76 >= 60. However, once a condition is satisfied, the appropriate statements are executed (grade = 'C';) and the remaining conditions are not evaluated.

The switch Statement

Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths. A switch works with the byte, short, char, and int primitive data types. It also works with *enumerated types* (discussed in Enum Types), the String class, and a few special

classes that wrap certain primitive types: Character, Byte, Short, and Integer (discussed in Numbers and Strings).

The following code example, SwitchDemo, declares an int named month whose value represents a month. The code displays the name of the month, based on the value of month, using the switch statement.

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

        int month = 8; String
        monthString; switch
        (month) {
            case 1: monthString = "January"; break;
            case 2: monthString = "February"; break;
            case 3: monthString = "March";
                    break;
            case 4: monthString = "April"; break;
            case 5: monthString = "May";
                    break;
            case 6: monthString = "June"; break;
            case 7: monthString = "July"; break;
            case 8: monthString = "August"; break;
            case 9: monthString = "September"; break;
            case 10: monthString = "October";
                    break;
            case 11: monthString = "November";
                    break;
            case 12: monthString = "December";
                    break;
            default:  monthString = "Invalid month";
                    break;
        }
        System.out.println(monthString);
    }
}
```

In this case, August is printed to standard output.

The while and do-while Statements

The while statement continually executes a block of statements while a particular condition is true. Its syntax can be expressed as:

```
while (expression) {
     statement(s)
```

}

The while statement evaluates *expression*, which must return a boolean value. If the expression evaluates to true, the while statement executes the *statement*(s) in the while block. Thewhile statement continues testing the expression and executing its block until the expression evaluates to false. Using

the while statement to print the values from 1 through 10 can be accomplished as in the following WhileDemo program:

```
class WhileDemo {
    public static void main(String[] args){ int count
        = 1;
        while (count < 11) { System.out.println("Count is: " +
            count); count++;
        }
    }
}
```

You can implement an infinite loop using the while statement as follows:

```
while (true){
    // your code goes here
}
```

The Java programming language also provides a do-while statement, which can be expressed as follows:

```
do {
     statement(s)
} while (expression);
```

The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once, as shown in the following DoWhileDemo program:

```
class DoWhileDemo {
    public    static void main(String[] args){
        int count =    1;
        do  {
            System.out.println("Count is: " + count); count++;
        } while (count < 11);
    }
```

}

The for Statement

The for statement provides a compact way to iterate over a range of values. Programmers often refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied. The general form of the for statement can be expressed as follows:

```
for (initialization; termination;
      increment) {
     statement(s)
}
```

When using this version of the for statement, keep in mind that:

- The *initialization* expression initializes the loop; it's executed once, as the loop begins.
- When the *termination* expression evaluates to false, the loop terminates.
- The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment *or* decrement a value.

The following program, ForDemo, uses the general form of the for statement to print the numbers 1 through 10 to standard output:

```
class ForDemo {
    public static void main(String[] args){ for(int
        i=1; i<11; i++){
            System.out.println("Count is: " + i);
        }
    }
}
```

The output of this program is:

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
```

```
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```

(ii) Demonstrate the if-else ladder using a sample Java code (DEC 2015)

Syntax : if-else-if Ladder

The if-then-else Statement

The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false. You could use an if-then-else statement in the applyBrakes method to take some action if the brakes are applied when the bicycle is not in motion. In this case, the action is to simply print an error message stating that the bicycle has already stopped.

```
void applyBrakes() {
    if (isMoving) {
        currentSpeed--;
    } else {
        System.err.println("The bicycle has already stopped!");
    }
}
```

The following program, IfElseDemo, assigns a grade based on the value of a test score: an A for a score of 90% or above, a B for a score of 80% or above, and so on.

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

        int testscore = 76;
        char grade;

        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) { grade
            = 'B';
        } else if (testscore >= 70) { grade
            = 'C';
        } else if (testscore >= 60) { grade
            = 'D';
        } else {
```

```
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

The output from the program is:

Grade = C

You may have noticed that the value of testscore can satisfy more than one expression in the compound statement: 76 >= 70 and 76 >= 60. However, once a condition is satisfied, the appropriate statements are executed (grade = 'C';) and the remaining conditions are not evaluated.

9. (i) Explain how Java differs from C and C++.

How Java Differs from C

- Major Difference – Java is a Object_oriented language.
- Java doesnot include C unique stmts, keywords sizeof, typeof
- Java doesnot contain datatypes struct & union
- Java doesnot define the type modifier keywords auto, extern, register, signed and unsigned.

How Java differs from C++

- Java is a true Object – Oriented Language while c++ is c with object-oriented extension.
- Java does not support Operator overloading, template classes
- Java does not support multiple inheritance. This is implemented in java called as "Interface"
- Java does not use pointers , global variables.
- Java has replaced the destructor function with a finalize() function
- No header files in java

(ii) Explain the various Data types of Java.

Java defines eight simple (or elemental) types of data: **byte, short, int, long, char, float, double, and boolean.** These can be put in four groups:

-
- Integers
- Floating-point numbers
- Characters

10. Write a Java program for constructor overloading.

```
class OverloadDemo {
void test() {
System.out.println("No parameters");
}
// Overload test for one integer parameter.
void test(int a) {
```

```
System.out.println("a: " + a);
}
// Overload test for two integer parameters.
void test(int a, int b) {
System.out.println("a and b: " + a + " " + b);
}
// overload test for a double parameter
double test(double a) {
System.out.println("double a: " + a);
return a*a;
}
}
class Overload {
public static void main(String args[]) {
OverloadDemo ob = new OverloadDemo();
double result;
// call all versions of test()
ob.test();
ob.test(10);
ob.test(10, 20);
result = ob.test(123.25);
System.out.println("Result of ob.test(123.25): " + result);
}
}
```

11. Create a package called "LIBRARY", which contains one class "BOOK" to maintain all the book information. In another program, import the package and create two classes "STAFF" and "STUDENT" from the super class "BOOK". In the main( ) method, create array of objects for the classes book and call the method in book class to read the details of the available books and display the menu as 1. staff 2. student. If the selected choice is 1, call the read method in staff class to read the details of the staff and the books they want to take (Check whether the given book ids are available in the book list). If the selected choice is 2, call the read method in student dais to read details of the student and the books they want to take (Check whether the given book ids are available in the book list). Before issuing each book, check that the book is available is in. the library or not. After reading the values, use the corresponding display methods to display the details of the staff or student with the books they have taken (DEC2014)

```
public class Book
{
  private String isbn;
  private String title;
  private String author;

  public Book(String isbnIn, String titleIn, String authorIn)
  {
    isbn = isbnIn;
    title = titleIn;
    author = authorIn;
  }
```

```
  public String getISBN()
  {
    return isbn;
  }

  public String getTitle()
  {
    return title;
  }

  public String getAuthor()
  {
    return author;
  }
public String   toString()
     {
       return "(" + isbn  + "  ,   " + author  + ",  " + title  +  ")\n";
     }
    public boolean equals (Object  objIn)
          {
Book   bookIn    =   (Book)    objIn;
       return  isbn.equals(bookIn.isbn);
     }
  public  int  hashCode()
        {
        return  isbn.hashCode();
       }

}
import java.util.*;

public class Library
{
    Map <String, Book> books; // declare map collection

    // create empty map
    public Library()
    {
      books = new HashMap<String, Book>();
    }
    // add the given book into the collection
    public boolean addBook(Book bookIn)
    {
      String keyIn = bookIn.getISBN(); // isbn will be key of map
      if (books.containsKey(keyIn)) // check if isbn already in use
      {
        return false; // indicate error
      }
      else // ok to add this book
```

```
    {
      books.put(keyIn, bookIn); // add key and book pair into map
      return true;
    }
  }

  // remove the book with the given isbn
  public boolean removeBook(String isbnIn)
  {
    if (books.remove(isbnIn)!= null) // check if item was removed
    {
      return true;
    }
    else // when item is not removed
    {
      return false;
    }
  }

  // return the number of books in the collection
  public int getTotalNumberOfBooks()
  {
    return books.size();
  }

  // return the book with the given isbn or null if no such book
  public Book getBook (String isbnIn)
  {
    return books.get(isbnIn);
  }

  // return the set of books in the collection
  public Set<Book> getAllBooks ()
  {
    Set<Book> bookSet = new HashSet<Book>(); // to store the set of books
    Set<String> theKeys = books.keySet(); // get the set of keys
    // iterate through the keys and put each value in the bookSet
    for (String isbn : theKeys)
    {
      Book theBook =  books.get(isbn);
      bookSet.add(theBook);
    }
    return bookSet; // return the set of books
  }
}
```

12. Explain in detail about Java classes and Java packages(DEC 2015)

Class Definitions: A java program may contain multiple class definitions. classes are the primary and essential elements of a java program

Main Method Class:- Since every java standalone program requires a main method as its starting point, this class is the essential part of a java program.

```
/*
* First Java program, which says "Hello, world!"
*/
public class Hello { // Save as "Hello.java"
public static void main(String[] args) {
System.out.println("Hello, world!"); // print message
}
}
```

- Packages are containers for classes that are used to keep the class name space separately.
- A unique name is to be used for each class to avoid name collisions.

Defining a package:

- To create a package, include a package command as the first statement in java source file.
- The ***package statement*** defines a name space in which classes are stored.
  Syntax: package Mypackage; Here Mypackage is the name of the package.
- We can have a hierarchy of packages.
  Syntax:package pkg1[.pkg2][.pkg3];

Inside pack1 folder:

```
package pack1;
public class student
{
 String name;
 String course;
 int tot, avg;
 public void initialise(String s1,String c)
 {
  name=s1;
  course=c;
 }
 public void calc(int m1,int m2,int m3)
 {
  tot=m1+m2+m3;
  avg=tot/3;
 }
 public void display()
 {
  System.out.println("\nStudent name : " +name);
  System.out.println("\nCourse       : " +course);
  System.out.println("\nTotal        : " +tot);
  System.out.println("\nAverage      : " +avg);
```

```
 }
}
```

MainClass(outside pack1)

```
import java.io.*;
import pack1.*;
class packeg extends student
{
 public static void main(String args[])throws IOException
 {
  int i;
  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
  String s,s1,s2;
  int m1,m2,m3;
  int no;
  System.out.println("\nEnter the number of students :");
  s=br.readLine();
  no=Integer.parseInt(s);
  for(i=1;i<=no;i++)
 {
  System.out.println("\nEnter Name:");
  s1=br.readLine();
  System.out.println("\nEnter Course :");
  s2=br.readLine();
  packeg p=new packeg();
  p.initialise(s1,s2);
  System.out.println("\nEnter the marks :");
  m1=Integer.parseInt(br.readLine());
  m2=Integer.parseInt(br.readLine());
  m3=Integer.parseInt(br.readLine());
  p.calc(m1,m2,m3);
  p.display();
 }
 }
```

13. Discuss about the expressions, operators and various control structures in Java. (DEC 2015)

Operators

Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.

with higher precedence are evaluated before operators with relatively lower precedence. Operators on the same line have equal precedence. When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first.

assignment operators are evaluated right to left.

Operator Precedence

Operators	**Precedence**
postfix	*expr++ expr--*
unary	*++expr --expr +expr -expr ~ !*
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :

Control Statements

The statements inside your source files are generally executed from top to bottom, in the order that they appear. *Control flow statements*, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to *conditionally* execute particular blocks of code. This section describes the decision-making statements (if-then, if-then-else, switch), the looping statements (for, while, do-while), and the branching statements (break, continue, return) supported by the Java programming language.

The if-then and if-then-else Statements

The if-then Statement

The if-then statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code *only if* a particular test evaluates to true. For example, the Bicycle class could allow the brakes to decrease the bicycle's speed *only if* the bicycle is already in motion. One possible implementation of the applyBrakes method could be as follows:

```
void applyBrakes() {
    // the "if" clause: bicycle must be moving if
    (isMoving){
        // the "then" clause: decrease current speed
        currentSpeed--;
```

```
    }
}
```

If this test evaluates to false (meaning that the bicycle is not in motion), control jumps to the end of the if-then statement.

In addition, the opening and closing braces are optional, provided that the "then" clause contains only one statement:

```
void applyBrakes() {
    // same as above, but without braces if
    (isMoving)
        currentSpeed--;
}
```

Deciding when to omit the braces is a matter of personal taste. Omitting them can make the code more brittle. If a second statement is later added to the "then" clause, a common mistake would be forgetting to add the newly required braces. The compiler cannot catch this sort of error; you'll just get the wrong results.

The if-then-else Statement

The if-then-else statement provides a secondary path of execution when an "if" clause evaluates

to false. You could use an if-then-else statement in the applyBrakes method to take some action if the brakes are applied when the bicycle is not in motion. In this case, the action is to simply print an error message stating that the bicycle has already stopped.

```
void applyBrakes() {
    if (isMoving) {
        currentSpeed--;
    } else {
        System.err.println("The bicycle has already stopped!");
    }
}
```

The following program, IfElseDemo, assigns a grade based on the value of a test score: an A for a score of 90% or above, a B for a score of 80% or above, and so on.

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

```
        int testscore = 76;
        char grade;

        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) { grade
            = 'B';
        } else if (testscore >= 70) { grade
            = 'C';
        } else if (testscore >= 60) { grade
            = 'D';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

The output from the program is:

```
Grade = C
```

You may have noticed that the value of testscore can satisfy more than one expression in the compound statement: 76 >= 70 and 76 >= 60. However, once a condition is satisfied, the appropriate statements are executed (grade = 'C';) and the remaining conditions are not evaluated.

The switch Statement

Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths. A switch works with the byte, short, char, and int primitive data types. It also works with *enumerated types* (discussed in Enum Types), the String class, and a few special classes that wrap certain primitive types: Character, Byte, Short, and Integer (discussed in Numbers and Strings).

The following code example, SwitchDemo, declares an int named month whose value represents a month. The code displays the name of the month, based on the value of month, using the switch statement.

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

```
        int month = 8;
        String monthString;
        switch (month) {
            case 1: monthString = "January";
                    break;
            case 2: monthString = "February";
                    break;
            case 3: monthString = "March";
                    break;
            case 4: monthString = "April";
                    break;
            case 5: monthString = "May";
                    break;
            case 6: monthString = "June";
                    break;
            case 7: monthString = "July";
                    break;
            case 8: monthString = "August";
                    break;
            case 9: monthString = "September";
                    break;
            case 10: monthString = "October";
                    break;
            case 11: monthString = "November";
                    break;
            case 12: monthString = "December";
                    break;
            default:  monthString = "Invalid month";
                    break;
        }
        System.out.println(monthString);
    }
}
```

In this case, August is printed to standard output.

The while and do-while Statements

The while statement continually executes a block of statements while a particular ondition is true. Its syntax can be expressed as:

```
while (expression) {
     statement(s)
}
```

The while statement evaluates *expression*, which must return a boolean value. If the expression evaluates to true, the while statement executes the *statement*(s) in the while block. Thewhile statement continues testing the expression and executing its block until the expression evaluates to false. Using

the while statement to print the values from 1 through 10 can be accomplished as in the following WhileDemo program:

```
class WhileDemo {
    public static void main(String[] args){ int
        count = 1;
        while (count < 11) { System.out.println("Count is: "
            + count); count++;
        }
    }
}
```

You can implement an infinite loop using the while statement as follows:

```
while (true){
    // your code goes here
}
```

The Java programming language also provides a do-while statement, which can be expressed as follows:

```
do {
     statement(s)
} while (expression);
```

The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once, as shown in the following DoWhileDemo program:

```
class DoWhileDemo {
    public     static void main(String[] args){
        int count =    1;
        do   {
            System.out.println("Count is: " + count);
            count++;
        } while (count < 11);
    }
}
```

The for Statement

The for statement provides a compact way to iterate over a range of values. Programmers often refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied. The general form of the for statement can be expressed as follows:

```
for (initialization; termination;
      increment) {
     statement(s)
}
```

When using this version of the for statement, keep in mind that:

- The *initialization* expression initializes the loop; it's executed once, as the loop begins.
- When the *termination* expression evaluates to false, the loop terminates.
- The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment *or* decrement a value.

The following program, ForDemo, uses the general form of the for statement to print the numbers 1 through 10 to standard output:

```
class ForDemo {
    public static void main(String[] args){ for(int
        i=1; i<11; i++){
            System.out.println("Count is: " + i);
        }
    }
}
```

The output of this program is:

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```

14. i) Explain Any Two Control Structure Supported By Java with Suitable Example (DEC 2016)

A programming language uses control statements to cause the flow of execution to advance and branch based on changes to the state of a program. Java's program control statements can be put into the following categories:

Selection
iteration
jump

Selection statements

Selection statements allows program to choose different paths of execution based upon the outcome of an expression or the state of a variable.

Iteration statements

Iteration statements enable program execution to repeat one or more statements (that is, iteration statements form loops).

Jump statements

Jump statements allows program to execute in a nonlinear fashion. All of Java's control statements are examined here.

Selection Statements

1. if Statement

The if statement is Java's conditional branch statement. It can be used to route program execution through two different paths. Here is the general form of the if statement:

```
if (condition)
{ statement1; }
else  { statement2; }
```

Here, each statement may be a single statement or a compound statement enclosed in curly braces (that is, a block). The condition is any expression that returns a boolean value. The else clause is optional. The if statement works like this:

If the condition is true, then statement1 is executed. Otherwise, statement2 (if it exists) is executed.

Solution:

Example

```
class Man { public static void main(String args[]){
int age= 49;
if(age>=35) System.out.println("Old");
else System.out.println("Young");
} }
```

Output: Old

2. Nested ifs

A nested if is an if statement that is the target of another if or else. Nested ifs are very common in programming. When you nest ifs, the main thing to remember is that an else statement always refers to the nearest if statement that is within the same block as the else and that is not already associated with an else.

3. The if-else-if Ladder

A common programming construct that is based upon a sequence of nested ifs is the if-else-if ladder. It looks like this:

```
if (condition1)
 statement1;
else if (condition2)
statement2; …….. ……..
```

else statement3;

The if statements are executed from the top to down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Example

```
class Result { public static void main(String args[])
{
int mark= 76;
if(mark>=60) System.out.println("1st Division");
else if(mark>=50 && mark<60) System.out.println("2nd Division");
else System.out.println("3rd  Division");  } }
```

Output: 1st Division

Switch Statements

The switch statement is Java's multi way branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression. As such, it often provides a better alternative than a large series of if-else-if statements. Here is the general form of a switch statement:

```
switch (expression)
{
case value1: // statement sequence break
; case value2: // statement sequence break;
... case valueN:
// statement sequence break; default: // default statement sequence
}
```

Solution:

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

for(int i=0; i<6; i++)
switch(i)
{
case 0: System.out.println("i is zero.");
break;
case 1: System.out.println("i is one.");
break;
case 2: System.out.println("i is two.");
break;
case 3: System.out.println("i is three.");
break;
default: System.out.println("i is greater than 3."); } } }
```

Output:

i is zero. i is one. i is two. i is three.
i is greater than 3. i is greater than 3.

14. ii) Declare An Interface Called Sports With Appropritate Methods. Design And Implements Football And Volleyball Interface Which Extends Sports Interface. Write Java Classes Which Implements Football And Volleyball Interface.(DEC 2016)

```
Interface sports{
 Float Sport_wt=10.0f;
}
Interface football
{
Void putplayer();}
Interface volleyball(){
Void putplayer1();
}
Class main1{
Int  f_player,v_player;
Void getplayer( int p1){
F_player=p1;
}
Void getplayer2(int  p2){
V_player=p2
}}
Class main2 extends main1 im plements sports,football,volletball{
Public void putplayer(){
Getplayer();
System.out.println("team football player is:"+f_player);
}
Public void putplayer(){
Getplayer2();
System.out.println("team football player is:"+v_player);
}}
Class main3{
Public static void main(String args[]){
Main2 a=new main2();
a.geplayer1(4);
a.getplayer2(10);
a.putplayer1();
a.putplayer();}}
```

15. Explain Exception Handling in Java with appropite Examples(DEC 2016)

Error occurred in a program can be of two types: syntax error or logical error. An exception is an abnormal condition that arises in a code sequence at run time. In other words, an exception is a run-time error.

A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. Exception Handling is a mechanism by which exception occurred in a program is handled.

Java exception handling is managed via five keywords:

Try
Catch
Thro
throws
finally.

Keyword and their meaning

catch block If an exception occurs within the try block, it is throws an exception to the catch block which handle it in some rational manner.

throw To manually throw an exception, use the keyword throw.

throwsA throws clause lists the types of exceptions that a method might throw

finallyAny code that absolutely must be executed before a method returns is put in a finally block.

Execution Flow of the try, catch and finally block

The general form of an exception-handling block is:

```
try  { // block of code to monitor for errors }
catch (ExceptionType1 exOb)
{ // exception handler for ExceptionType1 }
catch (ExceptionType2 exOb)
{ // exception handler for ExceptionType2 } …….. ……..
finally
{ // block of code to be executed before try block ends }
```

Exception Types

All exception types are subclasses of the built-in class Throwable. Thus, Throwable is at the top of the exception class hierarchy.

The Throwable class has two subclasses Error and Exception. Error and Exception classes are used for handling errors in java.

Object
Throwable
Exception
AWT Error
Error
SQL Exceptionn
Thread
Class not Found
Runtime Exception
Number Format

Using try and catch

The following program includes a try block and a catch clause which processes the ArithmeticException generated by the division-by-zero error.

Example:

```
class Ex {
public static void main(String args[])
{
int x, y; try  { x = 0; y= 1/ x;
System.out.println("This will not be printed."); }}
catch (ArithmeticException e)
 {
System.out.println("Division by zero."); }
System.out.println("After catch statement."); }
```

UNIT II

1. What are the categories of Annotations in JAVA? Explain in detail.

Java Annotations

Java annotation is used to provide the extra or supplement information about the program. Annotations in Java are utilized to give supplement data about a program.

- Java Annotations begin with '@'.
- Java Annotations don't change the activity of an ordered program.
- Annotations in Java help to relate metadata (data) to the program components i.e. case factors, constructors, strategies, classes, and so on.
- Annotations in Java are not unadulterated remarks as they can change the way a program is dealt with by compiler. See beneath code for instance.

Java Annotation Example –

```
1.  class Base
2.  {
3.  public void display()
4.  {
5.  System.out.println("Base display()");
6.  }
7.  }
8.  class Derived extends Base
9.  {
10. @Override
11. public void display(int x)
12. {
13. System.out.println("Derived display(int )");
14. }
15. public static void main(String args[])
16. {
17. Derived obj = new Derived();
18. obj.display();
19. }
20. }
```

a. Categories of Java Annotations

There are 3 types of Annotations in Java.

Types of Annotations in Java

i. Marker Java Annotations

The main design is to mark an annotation in Java. These Java annotations contain no individuals and don't comprise any information. Along these lines, its quality as a comment is adequate. Since, marker interface contains no individuals, just deciding if it is available or missing is adequate. @Override is a case of Marker Annotation.

Example: – @TestAnnotation()

Read about Loops in Java – Types and Examples of Looping in Java

ii. Single Value Java Annotations

These annotations contain just a single part and permit a shorthand type of determining the estimation of the part. We just need to indicate the incentive for that part when the comment is connected and don't have to determine the name of the part. However, to utilize this shorthand, the name of the part should value.

iii. Full Java Annotations

They contain multiple values, pairs, data members, etc.

Example:- @TestAnnotation(owner="Shadow", value="Class Flair")

b. Predefined/ Standard Java Annotations

There are seven built-in annotations in Java.

- The four imported from java.lang.annotation

@Retention, @Documented, @Target, and @Inherited

- The three in java.lang

@Deprecated, @Override, and @SuppressWarnings

Let's Discuss Java Number – Number Methods with Syntax and Examples

Java Annotations

i. @Deprecated Annotation

- @ Deprecated Java Annotation used to indicate that a declaration has become old and has been replaced by a newer one, thus it is a marker annotation
- The Javadoc @deprecated tag ought to utilize when a component has been deployed.
- A @deprecated tag is for documentation and @Deprecated annotation is for runtime reflection.
- A @deprecated tag has a higher need than @Deprecated annotation when both areas one utilized.

```
1.  public class DeprecatedTest
2.  {
3.  @Deprecated
4.  public void Display()
5.  {
6.  System.out.println("Deprecatedtest display()");
7.  }
8.  public static void main(String args[])
9.  {
10. DeprecatedTest d1 = new DeprecatedTest();
11. d1.Display();
12. }
13. }
```

Read About Java Regular Expression (Java Regex) with Examples

ii. @Override Annotation

@Override Java Annotations are a marker annotation that can utilize just on strategies. A technique clarified with @Override must supersede a strategy from a superclass. In the event that it doesn't, an order time error will come about (see this for instance). It is utilized to guarantee that a superclass technique is really superseded, and not just over-loaded.

```
1.  class Base
2.  {
3.  public void Display()
4.  {
5.  System.out.println("Base display()");
6.  }
7.  public static void main(String args[])
8.  {
9.  Base t1 = new Derived();
10. t1.Display();
11. }
12. }
13. class Derived extends Base
14. {
15. @Override
16. public void Display()
17. {
18. System.out.println("Derived display()");
19. }
20. }
```

Output –

Derived displays

iii. @SuppressWarnings Annotation

These Java Annotations are utilizing to educate the compiler to smother indicated compiler notices. The warnings are indicated by the name, in string structure. This sort of annotation can connect to a statement.

Java warnings notices under two classifications. They are depreciation and unchecked. Any unchecked cautioning is created when an inheritance code interfaces with a code that utilization generics.

```
1.  class DeprecatedTest
2.  {
3.  @Deprecated
4.  public void Display()
5.  {
6.  System.out.println("Deprecatedtest display()");
7.  }
8.  }
9.  public class SuppressWarningTest
10. {
11. @SuppressWarnings({"checked", "deprecation"})
12. public static void main(String args[])
13. {
14. DeprecatedTest d1 = new DeprecatedTest();
15. d1.Display();
16. }
17. }
```

Output-

Deprecatedtest display()

iv. @Documented Annotations

It is a marker interface that tells an apparatus that an explanation is to archive. Java Annotations are excluded by Javadoc remarks. Utilization of @Documented comment in the code empowers devices like Javadoc to process it and incorporate the explanation compose data in the produced archive.

v. @Target Annotations

It is intended to utilize just as an explanation for another annotation. @Target takes one argument, which must be consistent with the ElementType count. This annotation determines the kind of assertions to which Java annotation can connect. The constants are appeared beneath alongside the sort of affirmation to which they relate.

Table 1 – @ Target Java Annotations

Target Constant	Annotations Can be Applied To
ANNOTATION_TYPE	Another annotation
CONSTRUCTOR	Constructor
FIELD	Field
LOCAL_VARIABLE	Local variable
METHOD	Method
PACKAGE	Package
PARAMETER	Parameter
TYPE	Class, Interface, or enumeration

We can indicate at least one of these qualities in a @Targetannotation. To determine different esteems, we should indicate them inside a props delimited rundown. For instance, to indicate that an annotation applies just to fields and neighborhood factors, you can utilize this @Target comment: @Target({ElementType.FIELD, ElementType.LOCAL_VARIABLE}) @Retention Annotation It figures out where and to what extent the comment is retent. The 3 esteems that the @Retention explanation can have:

- **SOURCE:** Annotations will be held at the source level and disregarded by the compiler.
- **CLASS:** Annotations will be held at order time and overlooked by the JVM.
- **RUNTIME:** These will be held at runtime.

vi. @Inherited Annotations

@Inherited Java annotations are marker annotation that can utilize just on annotation affirmation. It influences just explanations that will utilize for class presentations. @Inherited makes the comment for a superclass acquire by a subclass. In this way, when a demand for a particular comment is made to the subclass, if that comment is absent in the subclass, at that point its superclass is checked. In the event that that comment is available in the superclass, and in the event that it is clarified with @Inherited, at that point that comment will return.

vii. User-defined/ Custom Annotations

Client characterized Java explanations can utilize to clarify program components, i.e. factors, constructors, techniques, and so forth. These explanations can connect just before the statement of a component (constructor, strategy, classes, and so forth).

Syntax –

```
1. [Access Specifier] @interface<AnnotationName>
2. {
3. DataType <Method Name>() [default value];
4. }
```

- AnnotationName is an identifier.
- Parameter ought not relate to strategy assertions and tosses provision ought not utilize with technique revelation.
- Parameters won't have an invalid esteem yet can have a default esteem.
- default esteem is discretionary.
- Return kind of strategy ought either crude, enum, string, class name or exhibit of crude, enum, string or class name write.

Let's Look at Java Generics Tutorial- Class, Functions of Generics in Java

Example –

```
1. import java.lang.annotation.Documented;
2. import java.lang.annotation.Retention;
3. import java.lang.annotation.RetentionPolicy;
4. @Documented
5. @Retention(RetentionPolicy.RUNTIME)
6. @ interface TestAnnotation
7. {
8. String Developer() default "Rahul";
9. String Expirydate();
10. }
11. public class Test
12. {
13. @TestAnnotation(Developer="data", Expirydate="01-10-2020")
14. void fun1()
15. {
16. System.out.println("Test method 1");
```

```
17. }
18.
19. @TestAnnotation(Developer="fair", Expirydate="01-10-2020")
20. void fun2()
21. {
22. System.out.println("Test method 2");
23. }
24.
25. public static void main(String args[])
26. {
27. System.out.println("Hello");
28. }
29. }
```

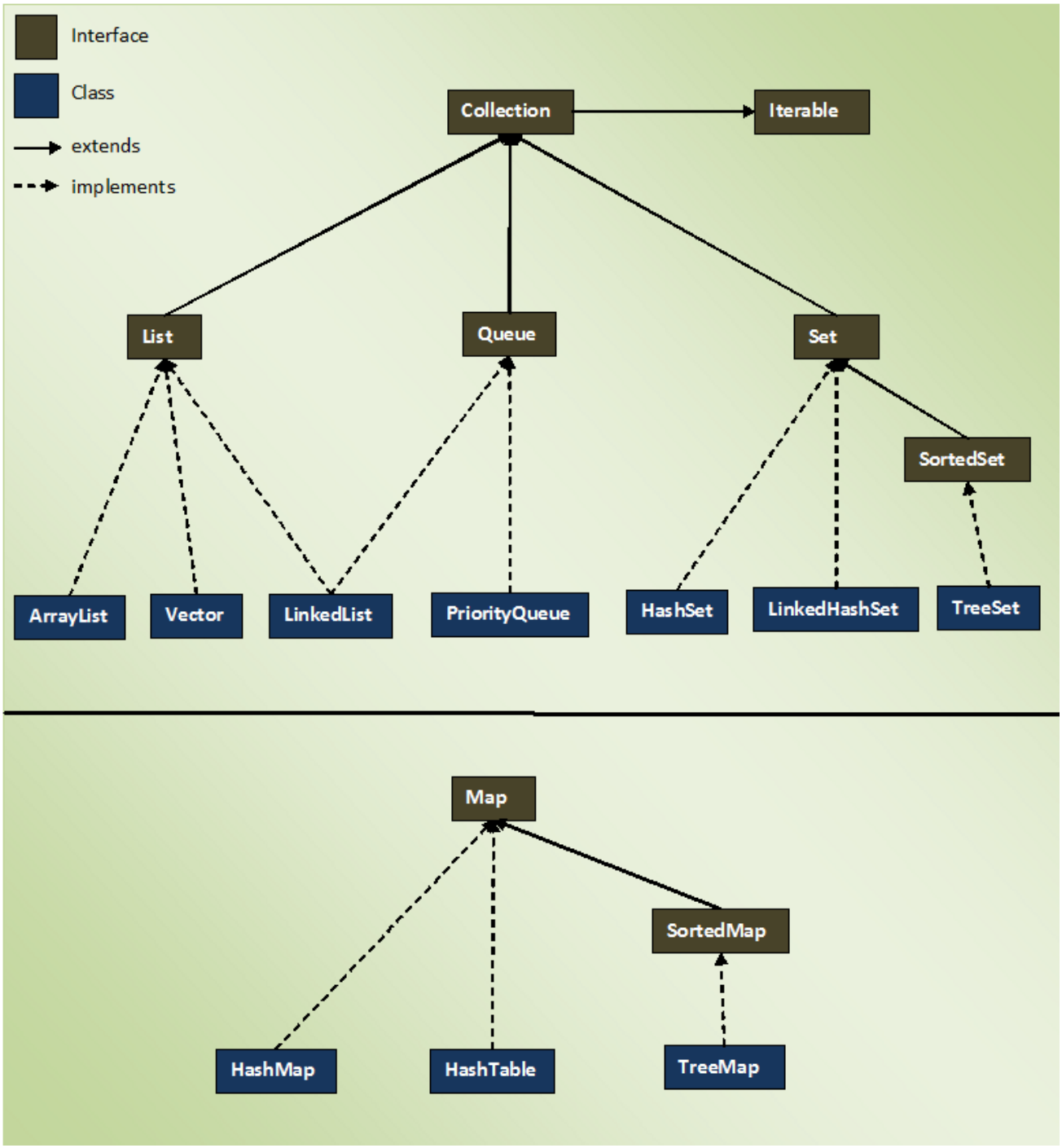
Output –

Hello

2. Write down the hierarchy of Collection framework.

Class Hierarchy Of Collection Framework :

All classes and interfaces related to Collection Framework are placed in java.util package. java.util.Collection class is at the top of class hierarchy of Collection Framework. Below diagram shows the class hierarchy of collection framework.



The entire collection framework is divided into four interfaces.

1) List —> It handles sequential list of objects. ArrayList, Vector and LinkedList classes implement this interface.

2) Queue —> It handles special list of objects in which elements are removed only from the head. LinkedList and PriorityQueue classes implement this interface.

3) Set —> It handles list of objects which must contain unique element. This interface is implemented by HashSetand LinkedHashSet classes and extended by SortedSet interface which in turn, is implemented by TreeSet.

4) Map —> This is the one interface in Collection Framework which is not inherited from Collection interface. It handles group of objects as Key/Value pairs. It is implemented by HashMap and HashTable classes and extended by SortedMap interface which in turn is implemented by TreeMap.

Three of above interfaces (List, Queue and Set) inherit from Collection interface. Although, Map is included in collection framework it does not inherit from Collection interface.

3. What is the need of collection framework? Give an example. Write down the advantages.

Since all the data operations(sorting/adding/deleting) are possible with Arrays and moreover array is suitable for memory consumption and performance is also better compared with Collections.

- Arrays are not resizable.
- Java Collections Framework provides lots of different useful data types, such as linked lists (allows insertion anywhere in constant time), resizeable array lists (like Vector but cooler), red-black trees, hash-based maps (like Hashtable but cooler).
- Java Collections Framework provides abstractions, so you can refer to a list as a List, whether backed by an array list or a linked list; and you can refer to a map/dictionary as a Map, whether backed by a red-black tree or a hashtable.

In other words, Java Collections Framework allows you to use the right data structure, because one size does not fit all.

- Java's collection classes provides a higher level interface than arrays.
- Arrays have a fixed size. Collections (see ArrayList) have a flexible size.
- Efficiently implementing a complicated data structures (e.g., hash tables) on top of raw arrays is a demanding task. The standard HashMap gives you that for free.
- There are different implementation you can choose from for the same set of services: ArrayList vs. LinkedList, HashMap vs. TreeMap, synchronized, etc.
- Finally, arrays allow covariance: setting an element of an array is not guaranteed to succeed due to typing errors that are detectable only at run time. Generics prevent this problem in arrays.

Take a look at this fragment that illustrates the covariance problem:

```
String[] strings = new String[10];
Object[] objects = strings;

objects[0] = new Date();  // <- ArrayStoreException: java.util.Date
```

collections offer Lists which are somewhat similar to arrays, but they offer many more things that are not. I'll assume you were just talking about List (and even Set) and leave Map out of it.

Yes, it is possible to get the same functionality as List and Set with an array, however there is a lot of work involved. The whole point of a library is that users do not have to "roll their own" implementations of common things.

Once you have a single implementation that everyone uses it is easier to justify spending resources optimizing it as well. That means when the standard collections are sped up or have

their memory footprint reduced that all applications using them get the improvements for free.

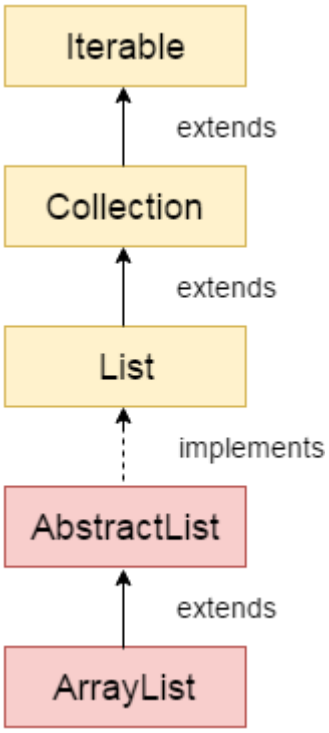
A single interface for each thing also simplifies every developers learning curve - there are not umpteen different ways of doing the same thing.

If you wanted to have an array that grows over time you would probably not put the growth code all over your classes, but would instead write a single utility method to do that. Same for deletion and insertion etc...

Also, arrays are not well suited to insertion/deletion, especially when you expect that the .length member is supposed to reflect the actual number of contents, so you would spend a huge amount of time growing and shrinking the array. Arrays are also not well suited for Sets as you would have to iterate over the entire array each time you wanted to do an insertion to check for duplicates. That would kill any perceived efficiency.

4. Write in detail about Array list. Write the code to create generic ArrayList.

Java ArrayList class



Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.

- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Hierarchy of ArrayList class

As shown in above diagram, Java ArrayList class extends AbstractList class which implements List interface. The List interface extends Collection and Iterable interfaces in hierarchical order.

ArrayList class declaration

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

1. public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cl oneable, Serializable

Constructors of Java ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

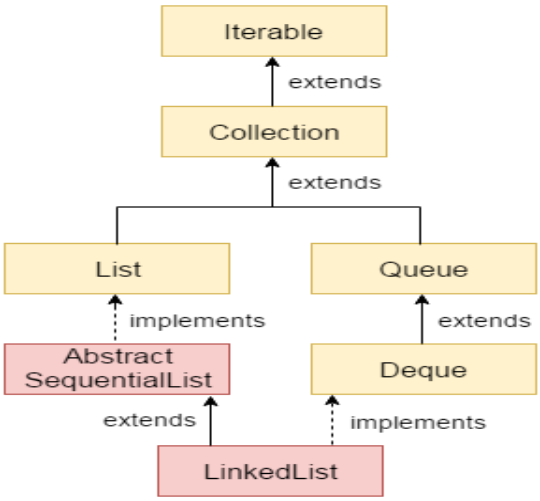
Methods of Java ArrayList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
void clear()	It is used to remove all of the elements from this list.

int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
Object[] toArray(Object[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean addAll(int index, Collection c)	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
Object clone()	It is used to return a shallow copy of an ArrayList.
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.

5. Write down the methods included in Linked list.Write a Java code for linked list implementation.

Java LinkedList class



Java LinkedList class uses doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
- Java LinkedList class can be used as list, stack or queue.

Hierarchy of LinkedList class

As shown in above diagram, Java LinkedList class extends AbstractSequentialList class and implements List and Deque interfaces.

Doubly Linked List

In case of doubly linked list, we can add or remove elements from both side.

fig- doubly linked list

LinkedList class declaration

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

1. public class LinkedList<E> extends AbstractSequentialList<E> implements List<E >, Deque<E>, Cloneable, Serializable

Constructors of Java LinkedList

Constructor	Description
LinkedList()	It is used to construct an empty list.
LinkedList(Collection c)	It is used to construct a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Java LinkedList Example

```
1.  import java.util.*;
2.  public class TestCollection7{
3.   public static void main(String args[]){
4.
5.   LinkedList<String> al=new LinkedList<String>();
6.   al.add("Ravi");
7.   al.add("Vijay");
8.   al.add("Ravi");
9.   al.add("Ajay");
10.
11.  Iterator<String> itr=al.iterator();
12.  while(itr.hasNext()){
```

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
void addFirst(Object o)	It is used to insert the given element at the beginning of a list.
void addLast(Object o)	It is used to append the given element to the end of a list.
int size()	It is used to return the number of elements in a list
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean contains(Object o)	It is used to return true if the list contains a specified element.
boolean remove(Object o)	It is used to remove the first occurence of the specified element in a list.
Object getFirst()	It is used to return the first element in a list.
Object getLast()	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.

```
13.             System.out.println(itr.next());
14.           }
15.         }
16.       }
```

Test it Now

Output:Ravi
Vijay
Ravi
Ajay

Java LinkedList Example: Book

```
1.        import java.util.*;
2.        class Book {
3.        int id;
4.        String name,author,publisher;
5.        int quantity;
6.        public Book(int id, String name, String author, String publisher, int quantity) {
7.            this.id = id;
8.            this.name = name;
9.            this.author = author;
10.           this.publisher = publisher;
11.           this.quantity = quantity;
12.       }
13.       }
14.       public class LinkedListExample {
15.       public static void main(String[] args) {
16.           //Creating list of Books
17.           List<Book> list=new LinkedList<Book>();
18.           //Creating Books
19.           Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20.           Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc
    Graw Hill",4);
21.           Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.           //Adding Books to list
23.           list.add(b1);
24.           list.add(b2);
25.           list.add(b3);
26.           //Traversing list
27.           for(Book b:list){
```

```
28.             System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
29.         }
30.     }
31.     }
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Methods of ArrayList class

1) add(Object o): This method adds an object o to the arraylist.

```
obj.add("hello");
```

This statement would add a string hello in the arraylist at last position.

2) add(int index, Object o): It adds the object o to the array list at the given index.

```
obj.add(2, "bye");
```

It will add the string bye to the 2nd index (3rd position as the array list starts with index 0) of array list.

3) remove(Object o): Removes the object o from the ArrayList.

```
obj.remove("Chaitanya");
```

This statement will remove the string “Chaitanya” from the ArrayList.

4) remove(int index): Removes element from a given index.

```
obj.remove(3);
```

It would remove the element of index 3 (4th element of the list – List starts with o).

5) set(int index, Object o): Used for updating an element. It replaces the element present at the specified index with the object o.

```
obj.set(2, "Tom");
```

It would replace the 3rd element (index =2 is 3rd element) with the value Tom.

6) int indexOf(Object o): Gives the index of the object o. If the element is not found in the list then this method returns the value -1.

```
int pos = obj.indexOf("Tom");
```
This would give the index (position) of the string Tom in the list.

7) Object get(int index): It returns the object of list which is present at the specified index.

```
String str= obj.get(2);
```
Function get would return the string stored at 3rd position (index 2) and would be assigned to the string "str". We have stored the returned value in string variable because in our example we have defined the ArrayList is of String type. If you are having integer array list then the returned value should be stored in an integer variable.

8) int size(): It gives the size of the ArrayList – Number of elements of the list.

```
int numberofitems = obj.size();
```
9) boolean contains(Object o): It checks whether the given object o is present in the array list if its there then it returns true else it returns false.

```
obj.contains("Steve");
```
It would return true if the string "Steve" is present in the list else we would get false.

10) clear(): It is used for removing all the elements of the array list in one go. The below code will remove all the elements of ArrayList whose object is obj.

```
obj.clear();
```

6. Write down the impotant methods included in Hash set.Write a Java program to demonstrate working of HashSet

HashSet extends AbstractSet and implements the Set interface. It creates a collection that uses a hash table for storage.

A hash table stores information by using a mechanism called hashing. In hashing, the informational content of a key is used to determine a unique value, called its hash code.

The hash code is then used as the index at which the data associated with the key is stored. The transformation of the key into its hash code is performed automatically.

Following is the list of constructors provided by the HashSet class.

Sr.No.	Constructor & Description
1	HashSet( ) This constructor constructs a default HashSet.
2	HashSet(Collection c) This constructor initializes the hash set by using the elements of the collection c.
3	HashSet(int capacity) This constructor initializes the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.
4	HashSet(int capacity, float fillRatio) This constructor initializes both the capacity and the fill ratio (also called load capacity) of the hash set from its arguments. Here the fill ratio must be between 0.0 and 1.0, and it determines how full the hash set can be before it is resized upward. Specifically, when the number of elements is greater than the capacity of the hash set multiplied by its fill ratio, the hash set is expanded.

Apart from the methods inherited from its parent classes, HashSet defines following methods –

Sr.No.	Method & Description
1	boolean add(Object o) Adds the specified element to this set if it is not already present.
2	void clear() Removes all of the elements from this set.
3	Object clone()

	Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
4	boolean contains(Object o) Returns true if this set contains the specified element.
5	boolean isEmpty() Returns true if this set contains no elements.
6	Iterator iterator() Returns an iterator over the elements in this set.
7	boolean remove(Object o) Removes the specified element from this set if it is present.
8	int size() Returns the number of elements in this set (its cardinality).

Example

The following program illustrates several of the methods supported by HashSet −

Live Demo

```
import java.util.*;
public class HashSetDemo {

  public static void main(String args[]) {
    // create a hash set
    HashSet hs = new HashSet();

    // add elements to the hash set
    hs.add("B");
    hs.add("A");
```

```
    hs.add("D");
    hs.add("E");
    hs.add("C");
    hs.add("F");
    System.out.println(hs);
  }
}
```

This will produce the following result −

Output

```
[A, B, C, D, E, F]
```

7. What are the four constructors available in Tree set? Write down the methods included in constructors.

TreeSet in Java with Examples

java.util.TreeSet is implementation class of SortedSet Interface. TreeSet has following important properties.

1. TreeSet implements the SortedSet interface so duplicate values are not allowed.
2. TreeSet does not preserve the insertion order of elements but elements are sorted by keys.
3. TreeSet does not allow to insert Heterogeneous objects. It will throw classCastException at Runtime if trying to add hetrogeneous objects.
4. TreeSet is basically implementation of a self-balancing binary search tree like Red-Black Tree. Therefore operations like add, remove and search take O(Log n) time. And operations like printing n elements in sorted order takes O(n) time.

Constructors:

Following are the four constructors in TreeSet class.

1. **TreeSet t = new TreeSet();**
   This will create empty TreeSet object in which elements will get stored in default natural sorting order.
2. **TreeSet t = new TreeSet(Comparator comp);**
   This constructor is used when you externally wants to specify sorting order of elements getting stored.

3. **TreeSet t = new TreeSet(Collection col);**
   This constructor is used when we want to convert any Collection object to TreeSet object.
4. **TreeSet t = new TreeSet(SortedSet s);**
   This constructor is used to convert SortedSet object to TreeSet Object.

Synchronized TreeSet:
Implementation of TreeSet class is not synchronized. If there is need of synchronized version of TreeSet, it can be done externally using Collections.synchronizedSet() method.

```
TreeSet ts = new TreeSet();
Set syncSet = Collections.synchronziedSet(ts);
```

Adding (or inserting) Elements to TreeSet:
TreeSet supports add() method to insert elements to it.

```
// Java program to demonstrate insertions in TreeSet
import java.util.*;

class TreeSetDemo
{
   public static void main (String[] args)
   {
      TreeSet ts1= new TreeSet();
      ts1.add("A");
      ts1.add("B");
      ts1.add("C");

      // Duplicates will not get insert
      ts1.add("C");

      // Elements get stored in default natural
      // Sorting Order(Ascending)
      System.out.println(ts1);  // [A,B,C]

      // ts1.add(2) ; will throw ClassCastException
      //           at run time
   }
}
```

Output :

```
[A, B, C]
```

Null Insertion:
If we insert null in a TreeSet, it throws NullPointerException because while inserting null it will get compared to existing elements and null can not be compared to any value.

```
// Java program to demonstrate null insertion
// in TreeSet
import java.util.*;

class TreeSetDemo
{
   public static void main (String[] args)
   {
      TreeSet ts2= new TreeSet();
      ts2.add("A");
```

```
        ts2.add("B");
        ts2.add("C");
        ts2.add(null); // Throws NullPointerException
    }
}
```

Output :

```
Exception in thread "main" java.lang.NullPointerException
    at java.util.TreeMap.put(TreeMap.java:563)
    at java.util.TreeSet.add(TreeSet.java:255)
    at TreeSetDemo.main(File.java:13)
```

Note: For empty tree-set, when you try to insert null as first value, you will get NPE from JDK 7.From 1.7 onwards null is not at all accepted by TreeSet. However upto JDK 6, null will be accepted as first value, but any if we insert any more value in TreeSet, it will also throw NullPointerException. Hence it was considered as bug and thus removed in JDK 7.

Methods:

TreeSet implements SortedSet so it has availability of all methods in Collection, Set and SortedSet interfaces. Following are the methods in Treeset interface.

1. **void add(Object o):** This method will add specified element according to some sorting order in TreeSet. Duplicate entires will not get added.
2. **boolean addAll(Collection c):** This method will add all elements of specified Collection to the set. Elements in Collection should be homogeneous otherwise ClassCastException will be thrown.Duplicate Entries of Collection will not be added to TreeSet.

```
// Java program to demonstrate TreeSet creation from
// ArrayList
import java.util.*;

class TreeSetDemo
{
    public static void main (String[] args)
    {
        ArrayList al = new ArrayList();
        al.add("GeeksforGeeks");
        al.add("GeeksQuiz");
        al.add("Practice");
        al.add("Compiler");
        al.add("Compiler"); //will not be added

        // Creating a TreeSet object from ArrayList
        TreeSet ts4 = new TreeSet(al);

        // [Compiler,GeeksQuiz,GeeksforGeeks,Practice]
        System.out.println(ts4);
    }
```

}

Output :

[Compiler, GeeksQuiz, GeeksforGeeks, Practice]

void clear() : This method will remove all the elements.
Comparator comparator(): This method will return Comparator used to sort elements in TreeSet or it will return null if default natural sorting order is used.
boolean contains(Object o): This method will return true if given element is present in TreeSet else it will return false.

Object first() : This method will return first element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.
Object last(): This method will return last element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.
SortedSet headSet(Object toElement): This method will return elements of TreeSet which are less than the specified element.
SortedSet tailSet(Object fromElement): This method will return elements of TreeSet which are greater than or equal to the specified element.
SortedSet subSet(Object fromElement, Object toElement): This method will return elements ranging from fromElement to toElement. fromElement is inclusive and toElement is exclusive.

```
// Java program to demonstrate TreeSet creation from
// ArrayList
import java.util.*;

class TreeSetDemo
{
   public static void main (String[] args)
   {

      TreeSet ts5 = new TreeSet();

      // Uncommenting below  throws NoSuchElementException
      // System.out.println(ts5.first());

      // Uncommenting below throws NoSuchElementException
      // System.out.println(ts5.last());

      ts5.add("GeeksforGeeks");
      ts5.add("Compiler");
      ts5.add("practice");

      System.out.println(ts5.first()); // Compiler
      System.out.println(ts5.last()); //Practice

      // Elements less than O. It prints
      // [Compiler,GeeksforGeeks]
      System.out.println(ts5.headSet("O"));
```

```
        // Elements greater than or equal to G.
        // It prints [GeeksforGeeks, Practice]
        System.out.println(ts5.tailSet("G"));

        // Elements ranging from C to P
        // It prints [Compiler,GeeksforGeeks]
        System.out.println(ts5.subSet("C","P"));

        // Deletes all elements from ts5.
        ts5.clear();

        // Prints nothing
        System.out.println(ts5);
    }
}
```

Output :

```
Compiler
practice
[Compiler, GeeksforGeeks]
[GeeksforGeeks, practice]
[Compiler, GeeksforGeeks]
[]
```

8. What are the two methods in which the comparator interface can be used?Explain with program.

Comparator Interface

In Java, Comparator interface is used to order(sort) the objects in the collection in your own way. It gives you the ability to decide how elements will be sorted and stored within collection and map.

Comparator Interface defines compare() method. This method has two parameters. This method compares the two objects passed in the parameter. It returns 0 if two objects are equal. It returns a positive value if object1 is greater than object2. Otherwise a negative value is returned. The method can throw a ClassCastException if the type of object are not compatible for comparison.

Rules for using Comparator interface:

1. If you want to sort the elements of a collection, you need to implement Comparator interface.

2. If you do not specify the type of the object in your Comparator interface, then, by default, it assumes that you are going to sort the objects of type Object. Thus, when you override the compare() method ,you will need to specify the type of the parameter as Object only.
3. If you want to sort the user-defined type elements, then while implementing the Comparator interface, you need to specify the user-defined type generically. If you do not specify the user-defined type while implementing the interface,then by default, it assumes Object type and you will not be able to compare the user-defined type elements in the collection

For Example:

If you want to sort the elements according to roll number, defined inside the class Student, then while implementing the Comparator interface, you need to mention it generically as follows:

```
class MyComparator implements Comparator<Student>{}
```

If you write only,

```
class MyComparator implements Comparator {}
```

Then it assumes, by default, data type of the compare() method's parameter to be Object, and hence you will not be able to compare the Student type(user-defined type) objects.

Example

Student class

```
class Student
int roll;
  String name;
  Student(int r,String n)
  {
     roll = r;
     name = n;
  }
  public String toString()
  {
     return roll+" "+name;
  }
```

MyComparator class

This class defines the comparison logic for Student class based on their roll. Student object will be sorted in ascending order of their roll.

```
class MyComparator implements Comparator<Student>
{
  public int compare(Student s1,Student s2)
    {
       if(s1.roll == s2.roll) return 0;
       else if(s1.roll > s2.roll) return 1;
       else return -1;
    }
}
public class Test
{

   public static void main(String[] args)
   {
      TreeSet< Student> ts = new TreeSet< Student>(new MyComparator());
      ts.add(new Student(45, "Rahul"));
      ts.add(new Student(11, "Adam"));
      ts.add(new Student(19, "Alex"));
      System.out.println(ts);
   }

}
```

```
[ 11 Adam, 19 Alex, 45 Rahul ]
```

As you can see in the ouput Student object are stored in ascending order of their roll.

Note:

- When we are sorting elements in a collection using Comparator interface, we need to pass the class object that implements Comparator interface.
- To sort a TreeSet collection, this object needs to be passed in the constructor of TreeSet.

- If any other collection, like ArrayList,was used, then we need to call sort method of Collections class and pass the name of the collection and this object as a parameter.
- For example, If the above program used ArrayList collection, the public class test would be as follows:

```
public class Test
{
  public static void main(String[] args)
  {
     ArrayList< Student> ts = new ArrayList< Student>();
     ts.add(new Student(45, "Rahul"));
     ts.add(new Student(11, "Adam"));
     ts.add(new Student(19, "Alex"));
     Collections.sort(ts,new MyComparator()); /*passing the name of the ArrayList and the object of the class that implements Comparator in a predefined sort() method in Collections class*/
     System.out.println(ts);
  }
}
```

---

9. Explain about Premain method. Create a class with premain method.

As the name indicates premain methods are the methods which can be executed before main method. Java Instrumentation will give a demonstration of how powerful Java is. Most importantly, this power can be realized by a developer for innovative means. For example using Java instrumentation, we can access a class that is loaded by the Java classloader from the JVM and modify its bytecode by inserting our custom code, all these done at runtime. Don´t worry about security, these are governed by the same security context applicable for Java classes and respective classloaders.

let us learn to instrument Java byte code using Java instrumentation. Mostly profilers, application monitoring agents, event loggers use Java instrumentation. This will serve as introductory level tutorial and once it is done, you can write a basic Java agent and do instrumentation on the Java byte code.

Note:

Package java.lang.instrument provides services that allow Java programming language agents to instrument programs running on the JVM. The mechanism for instrumentation is modification of the byte-codes of methods.

Key Components of Java Instrumentation:

Agent ? is a jar file containing agent and transformer class files.
Agent Class ? A java class file, containing a method named ´premain´.
Manifest ? manifest.mf file containing the "premain-class" property.
Transformer ? A Java class file implementing the interface ClassFileTransformer.

Instrumentation Agent Class:

Agent class contains the premain method and that is key in Java insturmentation. This is similar to the ´main´ method. This class is loaded by the same system classloader as it loads the other Java classes. premain method can have the following signatures,

1. public static void premain(String agentArgs, Instrumentation inst);
2. public static void premain(String agentArgs);

Example of Premain method or Writing Your Own Agent:

A Java Agent, once registered with the class loader, has a single method:

```
public class SkeletonClassFileTransformer implements ClassFileTransformer {
public byte[] transform(ClassLoader loader, String className, Class<?>
classBeingRedefined,
ProtectionDomain protectionDomain, byte[] classfileBuffer) throws
IllegalClassFormatException {
return null;
}
}
```

A Java Agent is given the option of modifying the array of bytes representing the class before the class loader finishes the class loading process. Here is a specification of class loading. In a nutshell, the bytes are given to the Java Agent after the class loader has retrieved them but before Linking. Your Java Agent can create a new byte array in a valid class file format and return it or, if it is not performing a transformation, return null.

Following is an example that simply prints a message like the following:

Class: StringCoding in: java/lang
Class: StringCoding$CharsetSE in: java/lang
Class: StringCoding$StringEncoder in: java/lang
Class: Main in: in/anyforum

```
Class: Shutdown in: java/lang
Class: Shutdown$Lock in: java/lang

package in.anyforum;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;
import java.security.ProtectionDomain;

public class ClassAndPackageNamePrintingClassFileTransformer implements
ClassFileTransformer {

public byte[] transform(ClassLoader loader, String fullyQualifiedClassName, Class<?>
classBeingRedefined,
ProtectionDomain protectionDomain, byte[] classofileBuffer) throws
IllegalClassFormatException {
String className = fullyQualifiedClassName.replaceAll(".*/", "");
String pacakge = fullyQualifiedClassName.replaceAll("/[a-zA-Z$0-9_]*$", "");
System.out.printf("Class: %s in: %s\n", className, pacakge);
return null;
}
}
```

How do you get all of this to work? Here´s what you´ll need to do:

---

1. Create an Implementation of ClassFileTransformer
2. Create a class with a premain method (could be in the first class, but but that would violate The Single Responsibility Principle).
3. Create jar file
4. Star the VM with a command line parameter.

1. Create an Implementation of ClassFileTransformer:

The class above is a complete example of a class that can “transform” a just-loaded class. By itself, it really does not do much. However, if you´d like to perform some custom transformation, you could create a new byte array, add in some Java Bytecodes and then return that class instead.

Why would you do this? Here are a few examples:

1. You´re adding logging code to a class
2. Custom implementation of AOP
3. Instrument a class to better thread-based testing (my reason for looking into this in the first place)
4. Etc

2. Create a class with a premain method:

The class file transformer is not directly added to the class loader. Instead, you create another

class with a method called premain that instantiates the class and registers it. Here is a complete example:

```
package in.anyforum;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.Instrumentation;

public class RegisterMyClassFileTransformer {
public static void premain(String agentArguments, Instrumentation instrumentation) {
instrumentation.addTransformer(new
ClassAndPackageNamePrintingClassFileTransformer());
}
}
```

3. Create Jar File:

When you add a class file transformer to the class loader, you must specify the name of a jar file. You cannot simply name a class in the classpath. So if the ClassAndPackageNamePrintingClassFileTransformer is in the class path, then you need to add the class RegisterMyClassFileTransformer to a jar file and add a manifest file to specify it.

The jar file needs to have the following structure:
Top Of Jar File
<sub directory>in
<sub directory>anyforum
<file>RegisterMyClassFileTransformer.class
<sub directory>META-INF
<file>MANIFEST.MF
The contents of the MANIFEST.MF file, at a minimum, would be:

```
Manifest-Version: 1.0
Premain-Class: in.anyforum.RegisterMyClassFileTransformer
```

4. Start the VM:

Finally, we need to star the VM:

```
java –javaagent:MyJarFile.jar <A Regular Class With A Main>
```

10. Explain about Generics in JAVA.

Generics

Generics was first introduced in Java5. Now it is one of the most profound feature of java programming language. Generic programming enables the programmer to create classes,interfaces and methods in which type of data is specified as a parameter. It provides a facility to write an algorithm independent of any specific type of data. Generics also provide

type safety. Type safety means ensuring that an operation is being performed on the right type of data before executing that operation.

Using Generics, it has become possible to create a single class ,interface or method that automatically works with all types of data(Integer, String, Float etc). It has expanded the ability to reuse the code safely and easily.

Before Generics was introduced, generalized classes,interfaces or methods were created using references of type Object because Object is the super class of all classes in Java, but this way of programming did not ensure type safety.

Syntax for creating an object of a generic type

```
Class_name <data type> reference_name = new Class_name<data type> ();
OR
Class_name <data type> reference_name = new Class_name<>();
```

This is also known as Diamond Notation of creating an object of Generic type.

Example of Generic class

```
class Gen <T> //<> brackets indicates that the class is of generic type
{
T ob;     //an object of type T is declared
Gen(T o)  //constructor
{
  ob = o;
}
public T getOb()
{
  return ob;
}
}

class Test
{
public static void main (String[] args)
{
  Gen < Integer> iob = new Gen<>(100);     //instance of Integer type Gen Class.
  int  x = iob.getOb();
  System.out.println(x);
```

```
  Gen < String> sob = new Gen<>("Hello");   //instance of String type Gen Class.
  String str = sob.getOb();
  System.out.println(str);
}
}
```

```
100
Hello
```

In the above program, we first passed an Integer type parameter to the Generic class. Then, we passed a String type parameter to the same Generic class. Hence, we reused the same class for two different data types. Thus, Generics helps in code reusability with ease.

Generics Work Only with Objects

Generics work only with objects i.e the type argument must be a class type.You cannot use primitive datatypes such as int, char etc. with Generics type. It should always be an object. We can use all the Wrapper Class objects and String class objects as Generic type.

```
Gen<int> iOb = new Gen<int>(07);    //Error, can't use primitive type
```

Generics Types of different Type Arguments are never same

Reference of one generic type is never compatible with other generic type unless their type argument is same. In the example above we created two objects of class Gen, one of type Integer, and other of type String, hence,

```
iob = sob;    //Absolutely Wrong
```

An array of Generic type cannot be created

Creation of a generic type array is not allowed in Generic programming. We can make a reference of an array, but we cannot instantiate it.

For example, In the above program, in class Gen,

T a[]; //this is allowed

T a[] = new T[10]; //this is not allowed

Generic Type with more than one parameter

In Generic parameterized types, we can pass more than 1 data type as parameter. It works the same as with one parameter Generic type.

```
class Gen <T1,T2>
{
T1 name;
T2 value;

Gen(T1 o1,T2 o2)
{
  name = o1;
  value = o2;
}
public T1 getName()
{
  return name;
}

public T2 getValue()
{
  return value;
}

}

class Test
{
public static void main (String[] args)
{
  Gen < String,Integer> obj = new Gen<>("StudyTonight",1);

  String  s = obj.getName();
  System.out.println(s);

  Integer i = obj.getValue();
```

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

Note: Since there are two parameters in Generic class - T1 and T2, therefore, while creating an instance of this Generic class, we need to mention two data types that needs to be passed as parameter to this class.

Generic Methods

You can also create generic methods that can be called with different types of arguments. Based on the type of arguments passed to generic method, the compiler handles each method.

The syntax for a generic method includes a type-parameter inside angle brackets, and should appear before the method's return type.

```
<type-parameter> return_type method_name (parameters) {...}
```

Example of Generic method

```
class GenTest
{
 static <b>< V, T></b> void display (V v, T t)
 {
  System.out.println(v.getClass().getName()+" = " +v);
  System.out.println(t.getClass().getName()+" = " +t);
 }
 public static void main(String[] args)
 {
  display(88,"This is string");
 }
}
```

```
java lang.Integer = 88
java lang.String = This is string
```

Generic Constructors

It is possible to create a generic constructor even if the class is not generic.

Example of Generic Constructor

```
class Gen
{
 private double val;
 < T extends Number> Gen(T ob)
 {
  val=ob.doubleValue();
 }
 void show()
 {
  System.out.println(val);
 }
}

class Test
{
 public static void main(String[] args)
 {
  Gen g = new Gen(100);
  Gen g1 = new Gen(121.5f);
  g.show();
  g1.show();
 }
}
```

```
100.0
121.5
```

Generic Interface

Like classes and methods, you can also create generic interfaces.

```
interface MyInterface< T >
{ .. }
```

Generic Bounded type Parameter

You can also set restriction on the type that will be allowed to pass to a type-parameter. This is done with the help of extends keyword when specifying the type parameter.

```
< T extends Number >
```

Here we have taken Number class, it can be any wrapper class name. This specifies that T can be only be replaced by Number class data itself or any of its subclass.

Generic Method with bounded type Parameters.

```
class Gen
{
 static < T, V extends number> void display(T t, V v)
 {
  System.out.println(v.getClass().getName()+" = " +v);
  System.out.println(t.getClass().getName()+" = " +t);
 }
 public static void main(String[] args)
 {
 // display(88,"This is string");
  display ("this is string",99);
 }
}
```

```
java.lang.String = This is string
java.lang.Double = 99.O
```

UNIT III

1. (i) Give a detailed note on I/O packages

- **Byte streams** provide a convenient means for handling input and output of bytes. Byte streams are used, when reading or writing binary data.
- **Character streams** provide a convenient means for handling input and output of characters.

The Byte Stream Classes

- Byte streams are defined by using two class hierarchies.
- InputStream and OutputStream are designed for byte streams.

Important methods are, Read & Write - read and write bytes of data

The Character Stream Classes

- Character streams are defined by using two class hierarchies.

- Abstract classes :Reader and Writer
- Methods : read( ) and write( ) - which read and write characters of data

(ii) Illustrate with a Java program to define and use an inner class

```
class Outer {
int outer_x = 100;
void test() {
Inner inner = new Inner();
inner.display();
}
// this is an inner class
class Inner {
void display() {
System.out.println("display: outer_x = " + outer_x);
}}}
class InnerClassDemo {
public static void main(String args[]) {
Outer outer = new Outer();
outer.test();
}}
```

2. What is a servlet? Explain briefly the Servlet life cycle and Servlet HTTP package.

A servlet is a Java programming language class used to extend the capabilities of a server. Although servlets can respond to any types of requests, they are commonly used to extend the applications hosted by web servers

Life cycle of a servlet:

- init( ) - invoked when the servlet is first loaded into memory
- service( ) - called for each HTTP request (for processing)
- destroy( ) - unloads the servlet from its memory

Interface: HttpServletRequest, HttpServletResponse, HttpSession, HttpSessionBindingListener

Class: Cookie, HttpServlet, HttpSessionEvent, HttpSessionBindingEvent

3. Write short notes on

(i) Inner Classes

An inner class is a non-static nested class. It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other non-static members of the outer class do. Thus, an inner class is fully within the scope of its enclosing class.

```
import java.applet.*;
import java.awt.event.*;
/* <applet code="AnonymousInnerClassDemo" width=200 height=100></applet>*/
public class AnonymousInnerClassDemo extends Applet {
public void init() {
addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent me) {
showStatus("Mouse Pressed");
}}}}
```

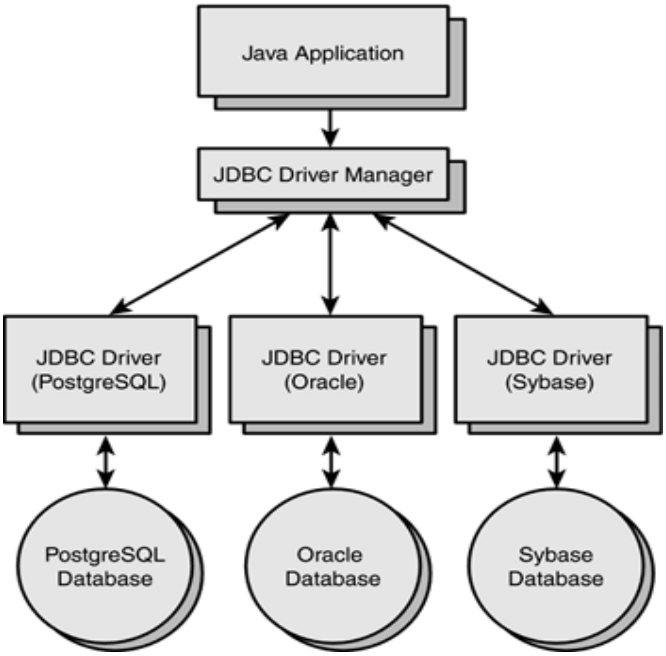
(ii) JDBC :

JDBC is used for accessing databases from Java applications. Information is transferred from

relations to objects and vice-versa

- databases *optimized for* searching/indexing
- objects *optimized for* engineering/flexibility

JDBC Architecture.



The topmost layer in this model is the **Java application.**
A Java application that uses JDBC can talk to many databases.
Like ODBC, JDBC provides a consistent way to connect to a database, execute commands, and retrieve the results.

The JDBC DriverManager

The JDBC `DriverManager` class is responsible for locating a JDBC driver needed by the application.
When a client application requests a database connection, the request is expressed in the form of a URL (Uniform Resource Locator).
URL might look like

jdbc:`odbc:DSNName`

Steps:

- Load the driver
- Define the connection URL
- Establish the connection
- Create a Statement object
- Execute a query using the Statement
- Process the result
- Close the connection

```
import java.sql.*;
public class jdbcodbc
{
```

```
public static void main(String arg[])throws Exception,SQLException
{
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch(Exception e){}
Connection cn=DriverManager.getConnection("jdbc:odbc:sam");
Statement s=cn.createStatement();
s.executeUpdate("insert into student values('monisha',66)");
String e="select  *  from student";
ResultSet rs=s.executeQuery(e);
while(rs.next())
{
   System.out.println("Name:"+rs.getString(1));
   System.out.println("no:"+rs.getInt(2));
}
s.close();
cn.close();
}
}
```

4. Write a program to perform arithmetic operation using RMI.

AddServerIntf.java

```
import java.rmi.*;
public interface AddServerIntf extends Remote
{
 double add(double d1,double d2)throws RemoteException;
 double sub(double d1,double d2)throws RemoteException;
}
```

AddServerImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
public class AddServerImpl extends UnicastRemoteObject implements AddServerIntf
{
 public AddServerImpl()throws RemoteException
{
}
public double add(double d1,double d2)throws RemoteException
{
 return d1+d2;
}
public double sub(double d1,double d2)throws RemoteException
{
 return d1-d2;
}
}
```

AddServer.java

```
import java.net.*;
```

```
import java.rmi.*;
public class AddServer
{
 public static void main(String args[])
 {
  try
 {
  AddServerImpl addserverimpl=new AddServerImpl();
  Naming.rebind("AddServer",addserverimpl);
 }
 catch(Exception e)
 {
 System.out.println("\nException:" +e);
 }
}
}
```

AddClient.java

```
import java.rmi.*;
public class AddClient
{
 public static void main(String args[])
 {
  try
 {
  String addserverurl="rmi://" +args[0]+ "/AddServer";
  AddServerIntf addserverintf=(AddServerIntf)Naming.lookup(addserverurl);
  System.out.println("\nthe first number is : " +args[1]);
  double d1=Double.valueOf(args[1]).doubleValue();
  System.out.println("\nthe second number is : " +args[2]);
  double d2=Double.valueOf(args[2]).doubleValue();
  System.out.println("\nSum is : " +addserverintf.add(d1,d2));
  System.out.println("\nSubtraction  : " +addserverintf.sub(d1,d2));
}
 catch(Exception e)
 {
 System.out.println("\nException : " +e);
 }
 }
}
```

5. Design an application for Student Result Checking using Servlet and JDBC.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
public class JdbcStudent extends HttpServlet
{
                Connection dbConn2;
                public void init(ServletConfig config)throws ServletException
```

```
        {
        super.init(config);
        try
        {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                dbConn2=DriverManager.getConnection("jdbc:odbc:wind");
        }
        catch(ClassNotFoundException e)
        {
                System.out.println("JDBC-ODBC bridge not found");
                return;
        }
        catch(SQLException e)
        {
                System.out.println("SQL exception thrown in init");
                return;
        }
        }
public void doGet(HttpServletRequest request,HttpServletResponse response)throws
        ServletException,IOException
        {
        try
        {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                Statement stat=dbConn2.createStatement();
                ResultSet students=stat.executeQuery(
                "SELECT studentid,studentname,grade,email FROM " +
                 " Student");
                out.println("<HTML>");
                out.println("<HEAD><TITLE> Student List </TITLE></HEAD>");
                out.println("<BODY>");
                out.println("<H2> Student List </H2>");
                out.println("<TABLE BORDER=1>");
                out.println("<TR><TH>Student ID</TH>");
                out.println("<TH> Student Name </TH>");
                out.println("<TH> Grade </TH>");
                out.println("<TH> Email </TH></TR>");
                while(students.next())
                {
                        out.println("<TR><TD>" +
                        students.getString("studentid") + "</TD><TD>" +
                        students.getString("studentname") + "</TD><TD>" +
                        students.getString("grade") + "</TD><TD>" +
                        students.getString("email") + "</TD><TR>");
                }
                out.println("</TABLE>");
                out.println("</BODY></HTML>");
                out.close();
        }
```

```
                catch (Exception e)
                {
                        e.printStackTrace();
                }
                }
                public String getServletInfo()
                {
                return "Sample JDBC Servlet";
                }
}
```

6. Write a Java swing program to perform personal information system.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class personal_info extends JFrame{

        //Initializing Components
        privateJTextField inputs[];
        privateJButton add, reset;
        privateJLabel labels[];
        private final String fldLabel[] = {"First Name: ","Last Name: ","Age: "," Address:","Country:","State:","City:","Phoneno:","Email ID:"};
        privateJPanel p1;

        Connection con;
        Statement st;
        String url;

        //Setting up GUI -Interface
publicpersonal_info() {

        //Setting up the Title of the Window
        super("Personal Information");

        //Set Size of the Window (WIDTH, HEIGHT)
        setSize(300,180);

        //Exit Property of the Window
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                //Constructing Components
        inputs = new JTextField[9];
        labels = new JLabel[9];
        add = new JButton("Add");
```

```
        reset = new JButton("Reset");
        p1 = new JPanel();

        //Setting Layout on JPanel 1 with 5 rows and 2 column
        p1.setLayout(new GridLayout(5,2));

        //Setting up the container ready for the components to be added.
        Container pane = getContentPane();
        setContentPane(pane);

        //Setting up the container layout
        GridLayout grid = new GridLayout(1,1,0,0);
        pane.setLayout(grid);

        //Creating a connection to MS Access and fetching errors using "try-catch" to check if it is successfully connected or not.
        try {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                con = DriverManager.getConnection("jdbc:odbc:sam");
                st = con.createStatement();

                JOptionPane.showMessageDialog(null,"Successfully Connected to Database","Confirmation", JOptionPane.INFORMATION_MESSAGE);

           }
        catch (HeadlessException | ClassNotFoundException | SQLException e)
        {
                JOptionPane.showMessageDialog(null,"Failed to Connect to Database","Error Connection", JOptionPane.ERROR_MESSAGE);
                System.exit(0);
        }

        //Constructing JLabel and JTextField using "for loop" in their desired order
        for(int count=0; count<inputs.length&& count<labels.length; count++)
        {
                labels[count] = new JLabel(fldLabel[count]);
                inputs[count] = new JTextField(20);

                //Adding the JLabel and the JTextFied in JPanel 1
                p1.add(labels[count]);
                p1.add(inputs[count]);
        }

        //Implemeting Even-Listener on JButton add
                add.addActionListener(
                newActionListener() {

                        //Handle JButton event if it is clicked

                public void actionPerformed(ActionEvent event) {
```

```
                //checking weather the fields are empty
                boolean pass=true;
                for(int i=0;i<inputs.length;i++)
                {

                        if(inputs[i].getText().equals(""))
                                pass=false;

                }
                if(pass ==false)
                        JOptionPane.showMessageDialog(null,"Fill up all the Fields","Error
Input", JOptionPane.ERROR_MESSAGE);
                else
                try {

                        String add = "insert into person
(fname,lname,age,address,country,state,city,phone,mail)"+"values
('"+inputs[0].getText()+"','"+inputs[1].getText()+"','"+inputs[2].getText()+"','"+inputs[3].getT
ext()+"','"+inputs[4].getText()+"','"+inputs[5].getText()+"','"+inputs[6].getText()+"','"+inputs
[7].getText()+"','"+inputs[8].getText()+"')";
                        st.execute(add); //Execute the add sql
JOptionPane.showMessageDialog(null,"Data Successfully Inserted","Confirmation",
JOptionPane.INFORMATION_MESSAGE);
        }
                catch (NumberFormatException e)
                {
                        JOptionPane.showMessageDialog(null,"Please enter an integer on the
Field AGE","Error Input", JOptionPane.ERROR_MESSAGE);
                } catch (SQLException ex) {
                        ex.printStackTrace();
                        Logger.getLogger(personal_info.class.getName()).log(Level.SEVERE,
null, ex);
                }
        }
        }
        );

                //Implemeting Even-Listener on JButton reset
                reset.addActionListener(
                newActionListener() {

                        //Handle JButton event if it is clicked
        public void actionPerformed(ActionEvent event)
    {
                                inputs[0].setText(null);
                                inputs[1].setText(null);
                                inputs[2].setText(null);
                                inputs[3].setText(null);
                                inputs[4].setText(null);
```

```
                                inputs[5].setText(null);
                                inputs[6].setText(null);
                                inputs[7].setText(null);
                                inputs[8].setText(null);
        }
        }
        );

                //Adding JButton "add" and "reset" to JPanel 1 after the JLabel and JTextField
                p1.add(add);
                p1.add(reset);

                //Adding JPanel 1 to the container
                pane.add(p1);

        /**Set all the Components Visible.
         * If it is set to "false", the components in the container will not be visible.
         */
        setVisible(true);
   }
        //Main Method
public static void main (String[] args) throws SQLException {
        personal_info aid = new personal_info();
        }
}
```

7. List out the classes and interfaces available in javax.servlet.http package.

Interface	**Description**
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.
HttpSessionBindingListener	Informs an object that it is bound to or unbound from a session.

Class	**Description**
Cookie	Allows state information to be stored on a client machine.
HttpServlet	Provides methods to handle HTTP requests and responses.
HttpSessionEvent	Encapsulates a session-changed event.
HttpSessionBindingEvent	Indicates when a listener is bound to or unbound from a session value, or that a session attribute changed.

8. Write short notes on the following servlet classes
GenericServlet, ServletInputStream, ServletOutputStream and ServletException

- init( ) method - invoked when the servlet is first loaded into memory
- service( ) - called for each HTTP request (for processing)
- destroy( ) - unloads the servlet from its memory

- The server calls the destroy( ) method to relinquish any resources such as file handles that are allocated for the servlet

GenericServlet - Implements the Servlet and ServletConfig interfaces.
ServletInputStream - Provides an input stream for reading requests from a client.
ServletOutputStream - Provides an output stream for writing responses to a client.
ServletException - Indicates a servlet error occurred

GenericServlet

- The GenericServlet class provides implementations of the basic life cycle methods for a servlet
- GenericServlet implements the Servlet and ServletConfig interfaces.
- a method log() is available to append a string to the server log file.

Methods:

- void log(String *s*)
- void log(String *s*, Throwable *e*) *s* is the string to be appended to the log, and *e* is an exception that occurred.

The ServletInputStream Class

- The ServletInputStream class extends InputStream.
- It is implemented by the server and provides an input stream that a servlet developer can use to read the data from a client request.

Method :

- int readLine(byte[ ] *buffer*, int *offset*, int *size*) throws IOException
- read bytes from the stream.
- *buffer* is the array into which *size* bytes are placed starting at *offset*.

The ServletOutputStream Class

- The **ServletOutputStream** class extends **OutputStream**.
- It is implemented by the server and provides an output stream that a servlet developer can use to write data to a client response.
- It also defines the **print( )** and **println( )** methods, which output data to the stream.

The ServletException Class

- ServletException, which indicates that a servlet problem has occurred.
- The second is UnavailableException, which extends ServletException. It indicates that a servlet is unavailable**.**

9. Explain JApplet, Icons and JLabel in detail.

Fundamental to Swing is the **JApplet** class, which extends **Applet**. Applets that use Swing must be subclasses of **JApplet**. **JApplet** is rich with functionality that is not found in **Applet**. For example, **JApplet** supports various "panes," such as the content pane, the glass pane, and the root pane. When adding a component to an instance of **JApplet**, do not invoke the **add( )** method of the applet. Instead, call **add( )** for the *content pane* of the **JApplet** object. The content pane can be obtained via the method shown here:
Container getContentPane( )
The **add( )** method of **Container** can be used to add a component to a content pane. Its form is shown here:
void add(*comp*)
Here, *comp* is the component to be added to the content pane.
Icon: In Swing, icons are encapsulated by the **ImageIcon** class, which paints an icon from an image. Two of its constructors are shown here:
ImageIcon(String *filename*)
ImageIcon(URL *url*)

The first form uses the image in the file named *filename*. The second form uses the image in the resource identified by *url*.
Labels: Swing labels are instances of the **JLabel** class, which extends **JComponent**. It can display text and/or an icon. Some of its constructors are shown here:
JLabel(Icon *i*)
Label(String *s*)
JLabel(String *s*, Icon *i*, int *align*)
Here, *s* and *i* are the text and icon used for the label. The *align* argument is either **LEFT**, **RIGHT**, **CENTER**, **LEADING**, or **TRAILING**. These constants are defined in the **SwingConstants** interface, along with several others used by the Swing classes. The icon and text associated with the label can be read and written by the following methods:
Icon getIcon( )
String getText( )
void setIcon(Icon *i*)
void setText(String *s*)
Here, *i* and *s* are the icon and text, respectively

10. Give an example program for JCheckbox in Java Swing.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JCheckBoxDemo" width=400 height=50>
</applet>
*/
public class JCheckBoxDemo extends JApplet
implements ItemListener {
JTextField jtf;
public void init() {
// Get content pane
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
// Create icons
ImageIcon normal = new ImageIcon("normal.gif");
ImageIcon rollover = new ImageIcon("rollover.gif");
ImageIcon selected = new ImageIcon("selected.gif");
// Add check boxes to the content pane
JCheckBox cb = new JCheckBox("C", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("C++", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("Java", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
```

```
cb.addItemListener(this);
contentPane.add(cb);
cb = new JCheckBox("Perl", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemListener(this);
contentPane.add(cb);
// Add text field to the content pane
jtf = new JTextField(15);
contentPane.add(jtf);
}
public void itemStateChanged(ItemEvent ie) {
JCheckBox cb = (JCheckBox)ie.getItem();
jtf.setText(cb.getText());
}
}
```

11. The idea of random mono alphabetic cipher is to use random letters for 'encrypting the input text. Suppose the keyword is FEATHER. Then first remove duplicate letters, yielding FEATHR, and append the other letters of the alphabet in reverse order. Now encrypt the letters as follows: ABCDEFGHIJKLMNOPQRSTUVWXYZ EATHRZYXWVUSQPONMLKJIGDCB

(DEC2014)

Write a Java program that encrypts and decrypts a file using this cipher.

```
private static char[] alphabet = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z'};

public static char[] shiftAlphabet(int shift)
{
   char[] newAlpha = new char[26];
   for (int i = 0; i < 26; i++)
   {
      if(((i + shift) < 26) && ((i + shift) >= 0))
      {
         newAlpha[i]  = alphabet[i + shift];
      }
      else if ((i + shift) >= 26)
      {
         newAlpha[i] = alphabet[i + shift - 26];
      }
   }
   return newAlpha;
}
public static String encrypt(String s, int shift)
{
   String e = "";
   for(int i = 0; i < s.length(); i++)
   {
      char letter = s.charAt(i);
      if (letter != ' ')
```

```
        {
            int f = find(alphabet, letter);
            if(((f + shift) < 26) && ((f + shift) >= 0))
            {
                letter  = alphabet[f + shift];
            }
            else if ((f + shift) >= 26)
            {
                letter = alphabet[f + shift - 26];
            }
            e = e + String.valueOf(letter);
        }
        else
        {
            e = e + " ";
        }
    }
    return e;
}
public static int find(char[] c, char c2)
{
    int w = 0;
    for(int i = 0; i < c.length; i++)
    {
        if(c[i] == (c2))
            w = i;
    }
    return w;
}
```

12.Create a table (ORACLE) namely "ernp" with the fields eid., ename, hra, da, pf and lie (ii) Insert 10 records into the table (iii) Alter the table by adding one more field namely netpay (iv) Compute the netpay of each employee and display it on the screen and also store it in the table. (DEC2014)

```
package payslips;

import java.util.*;

import payslips.Employee;
import payslips.Payslip;

public class MainProgramme
{
    public static String name;
    public static String street;
    public static String town;
    public static String postcode;
    public static int payrollNo;
    public static char taxcode;
```

```
   public static String type;

   static Scanner sc = new Scanner(System.in);
   static Scanner sd = new Scanner(System.in);
   static int tempvar;
   static int temppayrollNo;

   static ArrayList<Employee> list = new ArrayList<Employee>();

   static String names[] = { "John Hepburn", "David Jones", "Louise White",
      "Harry Martin", "Christine Robertson" };

   static String streets[] = { "50 Granton Road", "121 Lochend Park",
         "100 Govan Avenue", "51 Gorgie Road", "1 Leith Street" };

   static String towns[] = { "Edinburgh", "Edinburgh", "Glasgow", "Edinburgh",
         "Edinburgh" };

   static String postcodes[] = { "EH6 7UT", "EH1 1BA", "G15 5GG", "EH5 2PR",
         "EH4 4ST" };

   static int payrollNos[] = { 10001, 10002, 10003, 10004, 10005 };

   static char taxcodes[] = { 'C', 'B', 'C', 'C', 'B' };

   static String types[] = { "Monthly", "Weekly", "Monthly", "Monthly","Weekly" };

   public static void main(String[] args)
   {
      for (int i = 0; i < 5; i++) {
         name = names[i];
         street = streets[i];
         town = towns[i];
         postcode = postcodes[i];
         payrollNo = payrollNos[i];
         taxcode = taxcodes[i];
         type = types[i];
         Employee e = new Employee(name, street, town, postcode, payrollNo,taxcode, type);

         list.add(e);

      }

      // statements and prompts within the console for the user to follow
      System.out.println("Welcome to your Payroll System");
      System.out.println();
      System.out.println("Please enter your choice below from the following options");
      System.out.println();
      System.out.println("View all current weekly employees = 1 ");
      System.out.println("View all current monthly employees = 2 ");
```

```
        System.out.println("Delete an employee = 3 ");
        System.out.println("Add an employee = 4 ");
        System.out.println("Print an employee payslip = 5");
        System.out.println("To exit the system = 0 ");

        // allows user to enter number of choice and this reflects which statement is ran in
userChoice method
        tempvar = sc.nextInt();

        // runs the userChoice method
        userChoice();
    }

    // method to determine what statement runs according to which choice user makes
    public static void userChoice()
    {
        Employee tempEmployee = new Employee();
        boolean foundEmployee = false;

        // if user enters 1 it prints out the employee list.
        if (tempvar == 1)
        {
            Weekly.printWeekly();

        }
        else if (tempvar == 2)
        {
            Monthly.printMonthly();

        }
        else if (tempvar == 3)
        {
            printEmployeelist();
            System.out.println("");
            System.out.println("Above are a list of all employees.");
            System.out.println("Please enter the payroll number of the employee you wish to
delete from the system");
            temppayrollNo = sc.nextInt();

            // while loop to search on payroll number, deletes the employee if correct, error
message if not
            if (list.isEmpty() == false)
            {
                int a = 0;
                while (a < list.size())
                {
                    tempEmployee = list.get(a);
                    if (tempEmployee.payrollNo == temppayrollNo)
                    {
                        foundEmployee = true;
```

```
                    break;
                }
                a++;
            }
            if (foundEmployee == true)
            {
                System.out.println("You have deleted : "+ tempEmployee.getName());
                System.out.println();
                list.remove(tempEmployee);
                printEmployeelist();
            }
            else
            {
                System.out.println("The payroll number you have entered is incorrect");
            }
        }
    }

    else if (tempvar == 4) // allows the user to add an employee to the employee list, entering details using scanner
    {
        // initialises variables for entering title
        String tempstring1;
        int stringlength;
        int whitespace;
        String tempstring2;
        String tempstring3;
        // initialises variables for entering title
        String tempstring4;
        int stringlength2;
        int whitespace2;
        String tempstring5;
        String tempstring6;

        String tempstring7;

        System.out.println("You have chosen to add an employee to the system");
        System.out.println();

        // block of code that builds string together to get employee name
        System.out.println("Please enter the name of the new employee: ");
        tempstring1 = sd.nextLine(); // takes in string using scanner
        stringlength = tempstring1.length(); // saves length of string
        if (tempstring1.contains(" ")) // if statement to see if the string contains a space
        {
            whitespace = tempstring1.indexOf(" "); // finds the whitespace
            tempstring2 = tempstring1.substring((0), (whitespace));// creates string from start of input to whitespace
            tempstring3 = tempstring1.substring((whitespace) + 1,(stringlength));// creates string from whitespace plus one and adds on rest of the string
```

```
            tempEmployee.setName(tempstring2 + " " + tempstring3); // combines tempstring1
and tempstring2 together to complete full string
        }
        else // else statement that just enters the string if it is just one word
        {
            tempEmployee.setName(tempstring1);
        }

        // block of code that repeats same process as above to get street name
        System.out.println("Please enter the address of the employee: ");
        tempstring4 = sd.nextLine();
        stringlength2 = tempstring4.length();
        if (tempstring4.contains(" ")) {
            whitespace2 = tempstring4.indexOf(" ");
            tempstring5 = tempstring4.substring((0), (whitespace2));
            tempstring6 = tempstring4.substring((whitespace2) + 1,(stringlength2));
            tempEmployee.setStreet(tempstring5 + " " + tempstring6);
        }
        else
        {
            tempEmployee.setStreet(tempstring4);
        }

        System.out.println("Please enter the town: ");
        tempEmployee.setTown(sd.nextLine());// takes in town using scanner
        System.out.println("Please enter the postcode: ");
        tempstring7 = sd.nextLine(); //post code using scanner

        if (tempstring7.length() > 5 && tempstring7.length() < 9) // sets the length of string

        {
            tempEmployee.setPostcode(tempstring7);
        }
        else
        {
            tempEmployee.setPostcode("You have not entered a valid UK postcode");
        }

        tempEmployee.setPayrollNo(payrollNo + 1); // sets payroll number to next in
sequence

        System.out.println("Please enter your Tax code (A, B or C): ");
        tempEmployee.setTaxcode(sd.next().charAt(0));// takes in tax code using scanner

        System.out.println("Please enter Employee Type (Weekly or Monthly): ");
        tempEmployee.setType(sd.next()); //takes in type of employee

        list.add(tempEmployee);// creates temp employee and adds to list
        printEmployeelist();// prints employee list to view
    }
```

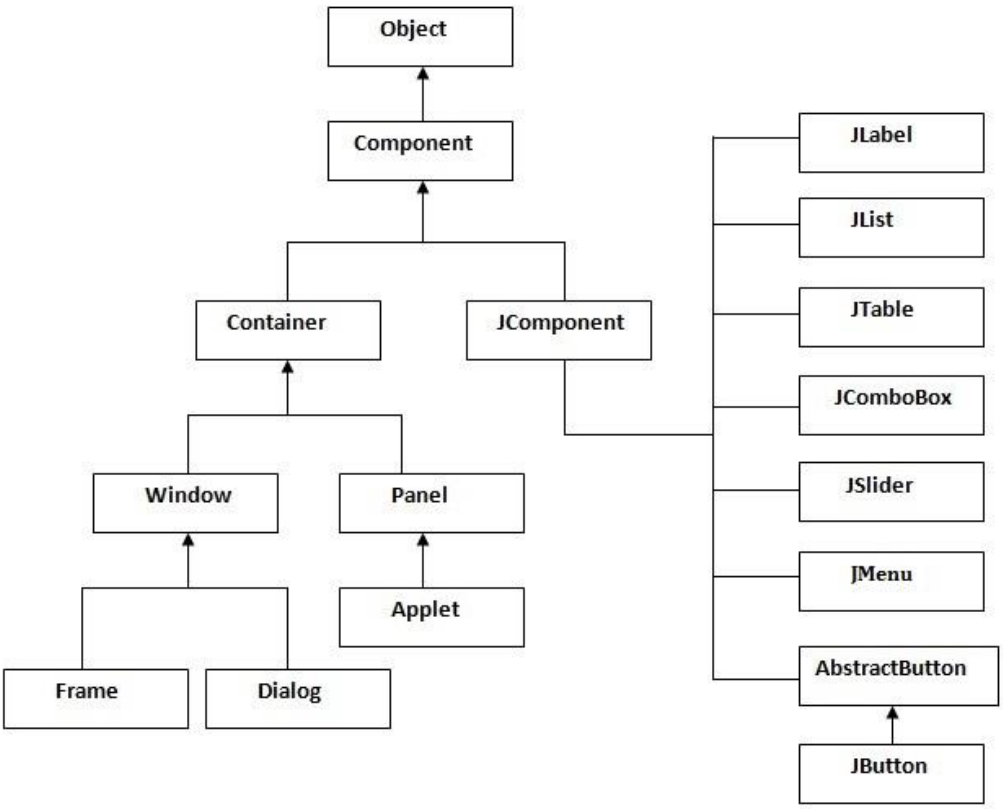
```
        else if (tempvar == 5)
        {
            Payslip.Payslips(); //runs payslip method from payslip class
        }

        else if (tempvar == 0) // if user hits 0 it allows them to exit the programme

        {
            System.out.println("You have exited the system");
            System.exit(0);
        }

        else // if any other choice entered they will be met with this message
        {
            System.out.println("You have entered the wrong choice");
        }
    }

    // method to create the book list using a for loop
        public static void printEmployeelist() {
        for (int i = 0; i < list.size(); i++)
            System.out.println(list.get(i));
    }
}
```

13. Explain about swing fundamentals and swing classes.(DEC 2015)

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.



Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```
1.  import javax.swing.*;
2.  public class FirstSwingExample {
3.  public static void main(String[] args) {
4.  JFrame f=new JFrame();//creating instance of JFrame
5.
6.  JButton b=new JButton("click");//creating instance of JButton
7.  b.setBounds(130,100,100, 40);//x axis, y axis, width, height
8.
```

```
9.          f.add(b);//adding button in JFrame
10.
11.         f.setSize(400,500);//400 width and 500 height
12.         f.setLayout(null);//using no layout managers
13.         f.setVisible(true);//making the frame visible
14.         }
15.         }
```

14. Write a Java Program to display the Details of a given Employee Using JDBC (DEC 2016)

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.*;
import java.io.*;
import java.net.*;

public class jdbcex {

  public static void main(String[] argv) {
System.out.println("-------- MySQL JDBC Connection Testing ------------");
Connection connection = null;
Statement st=null;
try {
Class.forName("com.mysql.jdbc.Driver");
connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/employ","root",
"tiger");
 st = connection.createStatement();

} catch (Exception e) {
System.out.println("Where is your MySQL JDBC Driver?");
System.out.println("Connection Failed! Check output console");

}
System.out.println("MySQL JDBC Driver Registered!");
if (connection != null) {
System.out.println("You made it, take control your database now!");
```

```
} else {
System.out.println("Failed to make connection!");
}

try
{
int ch=' ';
System.out.println("\n1.Select Employee\n2.Add Employee\n3Update Emp\n4.Delete
Employee\n5.Exit\n");
while(ch<='5')
{
System.out.println("\nEnter your choice\n");
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

 ch=Integer.parseInt(br.readLine());
       switch (ch)
{
          case 1:    ResultSet rs = st.executeQuery("select * from emp");
System.out.println("EMPLOYEE TABLE DETAILS");
System.out.println("Emp No \t Emp Name\n");
while(rs.next())
{
String s1= rs.getString("empno");
String s2=rs.getString("empname");
System.out.println(s1+"\t"+s2+"\n");
}
                    break;

          case 2:  System.out.println("\nADD DATA TO EMPLOYEE\n");
System.out.println("Enter the empno\n");

//int empno=Integer.parseInt(br.readLine());
String empno=br.readLine();
System.out.println("\nEnter the empname\n");
                     String empname=br.readLine();
                    String strsql="INSERT INTO emp(empno,empname) VALUES
                              ("'+empno+"','"+empname+"')";
 st.executeUpdate(strsql);
System.out.println(strsql);
System.out.println("\nThe above data is Inserted.\n");
 break;

          case 3:System.out.println("Enter the empno you want to update\n");

//int empno=Integer.parseInt(br.readLine());
String no=br.readLine();
System.out.println("\nEnter the new  empname\n");
String name=br.readLine();
  String strql="update emp set empname='"+name+"' where empno='"+no+"'";
st.executeUpdate(strql);
```

```
System.out.println(strql);
System.out.println("\nThe above data is updated.\n");
break;

        case 4: System.out.println("Enter the empno you want to delete\n");

//int empno=Integer.parseInt(br.readLine());
 no=br.readLine();
                //String sql="delete from emp where empno='3'";
String sql="delete from emp where empno='"+ no +"'";
st.executeUpdate(sql);
System.out.println(sql);
System.out.println("\nThe above data is deleted.\n") ;
break;

        case 5:
System.exit(0);
 break;
        default: System.out.println("choice");
break;
      }}
            connection.close();
      }
 catch (Exception e) {
        System.err.println("Got an exception! ");
        System.err.println(e.getMessage());       }

  }
}
```

15. Explain various methods of Httpservlet Class which are used to handle Http Request. (DEC 2016)

Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of HttpServlet must override at least one method, usually one of these:

- doGet, if the servlet supports HTTP GET requests
- doPost, for HTTP POST requests
- doPut, for HTTP PUT requests
- doDelete, for HTTP DELETE requests
- init and destroy, to manage resources that are held for the life of the servlet
- getServletInfo, which the servlet uses to provide information about itself

There's almost no reason to override the service method. service handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the doXXX methods listed above).

Likewise, there's almost no reason to override the doOptions and doTrace methods.

Servlets typically run on multithreaded servers, so be aware that a servlet must handle concurrent requests and be careful to synchronize access to shared resources. Shared

resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections.

doGet

protected void **doGet**(HttpServletRequest req,
HttpServletResponse resp)
throwsServletException,
java.io.IOException

Called by the server (via the service method) to allow a servlet to handle a GET request.

Overriding this method to support a GET request also automatically supports an HTTP HEAD request. A HEAD request is a GET request that returns no body in the response, only the request header fields.

When overriding this method, read the request data, write the response headers, get the response's writer or output stream object, and finally, write the response data. It's best to include content type and encoding. When using a PrintWriter object to return the response, set the content type before accessing the PrintWriter object.

The servlet container must write the headers before committing the response, because in HTTP the headers must be sent before the response body.

Where possible, set the Content-Length header (with the ServletResponse.setContentLength(int) method), to allow the servlet container to use a persistent connection to return its response to the client, improving performance. The content length is automatically set if the entire response fits inside the response buffer.

When using HTTP 1.1 chunked encoding (which means that the response has a Transfer-Encoding header), do not set the Content-Length header.

The GET method should be safe, that is, without any side effects for which users are held responsible. For example, most form queries have no side effects. If a client request is intended to change stored data, the request should use some other HTTP method.

The GET method should also be idempotent, meaning that it can be safely repeated. Sometimes making a method safe also makes it idempotent. For example, repeating queries is both safe and idempotent, but buying a product online or modifying data is neither safe nor idempotent.

If the request is incorrectly formatted, doGet returns an HTTP "Bad Request" message.

Parameters:
req - an HttpServletRequest object that contains the request the client has made of the servlet
resp - an HttpServletResponse object that contains the response the servlet sends to

the client
Throws:
java.io.IOException - if an input or output error is detected when the servlet handles the GET request
ServletException - if the request for the GET could not be handled
See Also:
ServletResponse.setContentType(java.lang.String)

getLastModified

protected long **getLastModified**(HttpServletRequest req)
Returns the time the HttpServletRequest object was last modified, in milliseconds since midnight January 1, 1970 GMT. If the time is unknown, this method returns a negative number (the default).

Servlets that support HTTP GET requests and can quickly determine their last modification time should override this method. This makes browser and proxy caches work more effectively, reducing the load on server and network resources.

Parameters:
req - the HttpServletRequest object that is sent to the servlet
Returns:
a long integer specifying the time the HttpServletRequest object was last modified, in milliseconds since midnight, January 1, 1970 GMT, or -1 if the time is not known

doHead

protected void **doHead**(HttpServletRequest req,
HttpServletResponse resp)
throwsServletException,
java.io.IOException

Receives an HTTP HEAD request from the protected service method and handles the request. The client sends a HEAD request when it wants to see only the headers of a response, such as Content-Type or Content-Length. The HTTP HEAD method counts the output bytes in the response to set the Content-Length header accurately.

If you override this method, you can avoid computing the response body and just set the response headers directly to improve performance. Make sure that the doHead method you write is both safe and idempotent (that is, protects itself from being called multiple times for one HTTP HEAD request).

If the HTTP HEAD request is incorrectly formatted, doHead returns an HTTP "Bad Request" message.

Parameters:
req - the request object that is passed to the servlet
resp - the response object that the servlet uses to return the headers to the client.
Throws:
java.io.IOException - if an input or output error occurs

ServletException - if the request for the HEAD could not be handled

doPost

protected void **doPost**(HttpServletRequest req,
HttpServletResponse resp)
throwsServletException,
java.io.IOException

Called by the server (via the service method) to allow a servlet to handle a POST request. The HTTP POST method allows the client to send data of unlimited length to the Web server a single time and is useful when posting information such as credit card numbers.

When overriding this method, read the request data, write the response headers, get the response's writer or output stream object, and finally, write the response data. It's best to include content type and encoding. When using a PrintWriter object to return the response, set the content type before accessing the PrintWriter object.

The servlet container must write the headers before committing the response, because in HTTP the headers must be sent before the response body.

Where possible, set the Content-Length header (with the ServletResponse.setContentLength(int) method), to allow the servlet container to use a persistent connection to return its response to the client, improving performance. The content length is automatically set if the entire response fits inside the response buffer.

When using HTTP 1.1 chunked encoding (which means that the response has a Transfer-Encoding header), do not set the Content-Length header.

This method does not need to be either safe or idempotent. Operations requested through POST can have side effects for which the user can be held accountable, for example, updating stored data or buying items online.

If the HTTP POST request is incorrectly formatted, doPost returns an HTTP "Bad Request" message.

Parameters:
req - an HttpServletRequest object that contains the request the client has made of the servlet
resp - an HttpServletResponse object that contains the response the servlet sends to the client
Throws:
java.io.IOException - if an input or output error is detected when the servlet handles the request
ServletException - if the request for the POST could not be handled

doPut

protected void **doPut**(HttpServletRequest req,
HttpServletResponse resp)
throwsServletException,
java.io.IOException

Called by the server (via the service method) to allow a servlet to handle a PUT request. The PUT operation allows a client to place a file on the server and is similar to sending a file by FTP.

When overriding this method, leave intact any content headers sent with the request (including Content-Length, Content-Type, Content-Transfer-Encoding, Content-Encoding, Content-Base, Content-Language, Content-Location, Content-MD5, and Content-Range). If your method cannot handle a content header, it must issue an error message (HTTP 501 - Not Implemented) and discard the request. For more information on HTTP 1.1, see RFC 2616 .

This method does not need to be either safe or idempotent. Operations that doPut performs can have side effects for which the user can be held accountable. When using this method, it may be useful to save a copy of the affected URL in temporary storage.

If the HTTP PUT request is incorrectly formatted, doPut returns an HTTP "Bad Request" message.

Parameters:
req - the HttpServletRequest object that contains the request the client made of the servlet
resp - the HttpServletResponse object that contains the response the servlet returns to the client
Throws:
java.io.IOException - if an input or output error occurs while the servlet is handling the PUT request
ServletException - if the request for the PUT cannot be handled

doDelete

protected void **doDelete**(HttpServletRequest req,
HttpServletResponse resp)
throwsServletException,
java.io.IOException

Called by the server (via the service method) to allow a servlet to handle a DELETE request. The DELETE operation allows a client to remove a document or Web page from the server.

This method does not need to be either safe or idempotent. Operations requested through DELETE can have side effects for which users can be held accountable. When using this method, it may be useful to save a copy of the affected URL in temporary storage.

If the HTTP DELETE request is incorrectly formatted, doDelete returns an HTTP "Bad Request" message.

Parameters:
req - the HttpServletRequest object that contains the request the client made of the servlet
resp - the HttpServletResponse object that contains the response the servlet returns to the client
Throws:
java.io.IOException - if an input or output error occurs while the servlet is handling the DELETE request
ServletException - if the request for the DELETE cannot be handled

doOptions

protected void **doOptions**(HttpServletRequest req,
HttpServletResponse resp)
throwsServletException,
java.io.IOException

Called by the server (via the service method) to allow a servlet to handle a OPTIONS request. The OPTIONS request determines which HTTP methods the server supports and returns an appropriate header. For example, if a servlet overrides doGet, this method returns the following header:

Allow: GET, HEAD, TRACE, OPTIONS

There's no need to override this method unless the servlet implements new HTTP methods, beyond those implemented by HTTP 1.1.

Parameters:
req - the HttpServletRequest object that contains the request the client made of the servlet
resp - the HttpServletResponse object that contains the response the servlet returns to the client
Throws:
java.io.IOException - if an input or output error occurs while the servlet is handling the OPTIONS request
ServletException - if the request for the OPTIONS cannot be handled

doTrace

protected void **doTrace**(HttpServletRequest req,
HttpServletResponse resp)
throwsServletException,
java.io.IOException

Called by the server (via the service method) to allow a servlet to handle a TRACE request. A TRACE returns the headers sent with the TRACE request to the client, so that they can be used in debugging. There's no need to override this method.

Parameters:
req - the HttpServletRequest object that contains the request the client made of the servlet
resp - the HttpServletResponse object that contains the response the servlet returns to the client
Throws:
java.io.IOException - if an input or output error occurs while the servlet is handling the TRACE request
ServletException - if the request for the TRACE cannot be handled

Service

protected void **service**(HttpServletRequest req,
HttpServletResponse resp)
throwsServletException,
java.io.IOException

Receives standard HTTP requests from the public service method and dispatches them to the doXXX methods defined in this class. This method is an HTTP-specific version of theServlet.service(javax.servlet.ServletRequest, javax.servlet.ServletResponse) method. There's no need to override this method.
Parameters:
req - the HttpServletRequest object that contains the request the client made of the servlet
resp - the HttpServletResponse object that contains the response the servlet returns to the client
Throws:
java.io.IOException - if an input or output error occurs while the servlet is handling the HTTP request
ServletException - if the HTTP request cannot be handled

Service

public void **service**(ServletRequest req,
ServletResponse res)
throwsServletException,
java.io.IOException

Dispatches client requests to the protected service method. There's no need to override this method.
Specified by:
service in interface Servlet
Specified by:
service in class GenericServlet
Parameters:
req - the HttpServletRequest object that contains the request the client made of the servlet
res - the HttpServletResponse object that contains the response the servlet returns to the client
Throws:

java.io.IOException - if an input or output error occurs while the servlet is handling the HTTP request
ServletException - if the HTTP request cannot be handled

16. Explain Jlist And Jtabbed Pane Components.(DEC 2016)

A tabbed pane is a component that appears as a group of folders in a file cabinet. Each folder has a title. When a user selects a folder, its contents become visible. Only one of the folders may be selected at a time. Tabbed panes are commonly used for setting configuration options. Tabbed panes are encapsulated by the JTabbedPane class, which extends JComponent. We will use its default constructor. Tabs are defined via the following method:
void addTab(String str, Component comp)
Here, str is the title for the tab, and comp is the component that should be added to the tab. Typically, a JPanel or a subclass of it is added.
The general procedure to use a tabbed pane in an applet is outlined here:
1. Create a JTabbedPane object.
2. Call addTab( ) to add a tab to the pane. (The arguments to this method define the title of the tab and the component it contains.)
3. Repeat step 2 for each tab.
4. Add the tabbed pane to the content pane of the applet.

Demonstration of Tabbed Panes.

Solution:
```
import javax.swing.*; /* <applet code="JTabbedPaneDemo" width=400
height=100></applet> */
public class JTabbedPaneDemo extends JApplet
{
public void init()
 {
JTabbedPane jtp = new JTabbedPane();
jtp.addTab("Cities", new CitiesPanel());
jtp.addTab("Colors", new ColorsPanel());
jtp.addTab("Flavors", new FlavorsPanel());
getContentPane().add(jtp);
}
}
class CitiesPanel extends JPanel
{
public CitiesPanel()
 {
JButton b1 = new JButton("New York"); add(b1);
 JButton b2 = new JButton("London");
add(b2);
 JButton b3 = new JButton("Hong Kong");
add(b3);
JButton b4 = new JButton("Tokyo");
add(b4);
 }
}
class ColorsPanel extends JPanel
{
public ColorsPanel()
```

```
{
JCheckBox cb1 = new JCheckBox("Red");
add(cb1);
  JCheckBox cb2 = new JCheckBox("Green");
add(cb2);  JCheckBox cb3 = new JCheckBox("Blue");
add(cb3);
 }
}
class FlavorsPanel extends JPanel
 {
public FlavorsPanel()
{
  JComboBox jcb = new JComboBox();
jcb.addItem("Vanilla");  jcb.addItem("Chocolate");
jcb.addItem("Strawberry");  add(jcb);
  }
 }
```

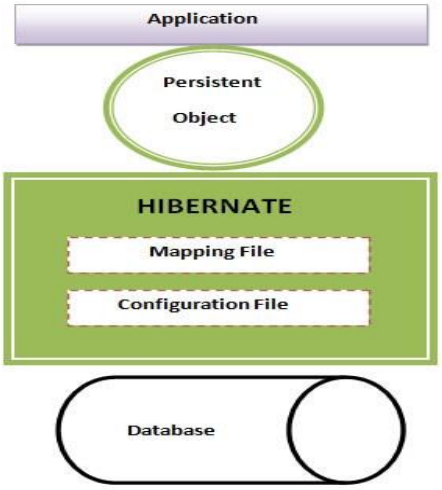
UNIT IV

1. What is hibernate? Explain about the architecture of the hibernate with neat diagram with features

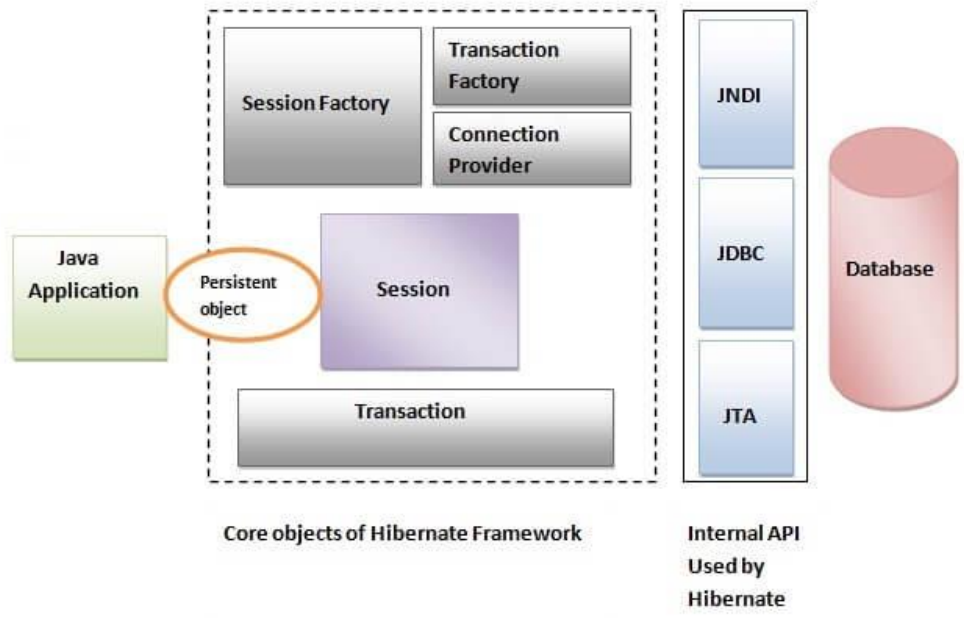
Hibernate Architecture

The Hibernate architecture includes many objects persistent object, session factory, transaction factory, connection factory, session, transaction etc.

There are 4 layers in hibernate architecture java application layer, hibernate framework layer, backhand api layer and database layer.Let's see the diagram of hibernate architecture:



This is the high level architecture of Hibernate with mapping file and configuration file.



Hibernate framework uses many objects session factory, session, transaction etc. alongwith existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

Elements of Hibernate Architecture

For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows:

SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

Session

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

ConnectionProvider

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

TransactionFactory

It is a factory of Transaction. It is optional.

2. Explain tiered application development in JAVA.

Overview of Enterprise Applications

The Java EE platform is designed to help developers create large-scale, multi-tiered, scalable, reliable, and secure network applications. A shorthand name for such applications is "enterprise applications," so called because these applications are designed to solve the problems encountered by large enterprises. Enterprise applications are not only useful for large corporations, agencies, and governments, however. The benefits of an enterprise application are helpful, even essential, for individual developers and small organizations in an increasingly networked world.

The features that make enterprise applications powerful, like security and reliability, often make these applications complex. The Java EE platform is designed to reduce the complexity of enterprise application development by providing a development model, API, and runtime environment that allows developers to concentrate on functionality.

Tiered Applications

In a multi-tiered application, the functionality of the application is separated into isolated functional areas, called tiers. Typically, multi-tiered applications have a client tier, a middle tier, and a data tier (often called the enterprise information systems tier). The client tier consists of a client program that makes requests to the middle tier. The middle tier's business functions handle client requests and process application data, storing it in a permanent datastore in the data tier.

Java EE application development concentrates on the middle tier to make enterprise application management easier, more robust, and more secure.

The Client Tier

The client tier consists of application clients that access a Java EE server and that are usually located on a different machine from the server. The clients make requests to the server. The server processes the requests and returns a response back to the client. Many different types of applications can be Java EE clients, and they are not always, or even often Java

applications. Clients can be a web browser, a standalone application, or other servers, and they run on a different machine from the Java EE server.

The Web Tier

The web tier consists of components that handle the interaction between clients and the business tier. Its primary tasks are the following:

- Dynamically generate content in various formats for the client.
- Collect input from users of the client interface and return appropriate results from the components in the business tier.
- Control the flow of screens or pages on the client.
- Maintain the state of data for a user's session.
- Perform some basic logic and hold some data temporarily in JavaBeans components.

Java EE Technologies Used in the Web Tier

The following Java EE technologies are used in the web tier in Java EE applications.

Table 1.Web-Tier Java EE Technologies

Technology	Purpose
Servlets	Java programming language classes that dynamically process requests and construct responses, usually for HTML pages
JavaServer Faces technology	A user-interface component framework for web applications that allows you to include UI components (such as fields and buttons) on a page, convert and validate UI component data, save UI component data to server-side data stores, and maintain component state.
JavaServer Faces Facelets technology	Facelets applications are a type of JavaServer Faces applications that use XHTML pages rather than JSP pages.
Expression Language	A set of standard tags used in JSP and Facelets pages to refer to Java EE components.
JavaServer Pages (JSP)	Text-based documents that are compiled into servlets and define how dynamic content can be added to static pages, such as HTML pages.
JavaServer Pages Standard Tag Library	A tag library that encapsulates core functionality common to JSP pages
JavaBeans Components	Objects that act as temporary data stores for the pages of an application

The Business Tier

The business tier consists of components that provide the business logic for an application. Business logic is code that provides functionality to a particular business domain, like the financial industry, or an e-commerce site. In a properly designed enterprise application, the core functionality exists in the business tier components.

Java EE Technologies Used in the Business Tier
The following Java EE technologies are used in the business tier in Java EE applications:

Table 2 Business Tier Java EE Technologies

Technology	Description
Enterprise JavaBeans (enterprise bean) components	Enterprise beans are managed components that encapsulate the core functionality of an application.
JAX-RS RESTful web services	An API for creating web services that respond to HTTP methods (for example GET or POST methods). JAX-RS web services are developed according to the principles of REST, or representational state transfer.
JAX-WS web service endpoints	An API for creating and consuming SOAP web services.
Java Persistence API entities	An API for accessing data in underlying data stores and mapping that data to Java programming language objects.
Java EE managed beans	Managed components that may provide the business logic of an application, but do not require the transactional or security features of enterprise beans.

The Enterprise Information Systems Tier

The enterprise information systems (EIS) tier consists of database servers, enterprise resource planning systems, and other legacy data sources, like mainframes. These resources typically are located on a separate machine than the Java EE server, and are accessed by components on the business tier.

Java EE Technologies Used in the EIS Tier
The following Java EE technologies are used to access the EIS tier in Java EE applications:

Table 3. EIS Tier Java EE Technologies

Technology	Description
The Java Database Connectivity API (JDBC)	A low-level API for accessing and retrieving data from underlying data stores. A common use of JDBC is to make SQL queries on a particular database.
The Java Persistence API	An API for accessing data in underlying data stores and mapping that data to Java programming language objects. The Java Persistence API is a much higher-level API than JDBC, and hides the complexity of JDBC from the user.
The Java EE Connector Architecture	An API for connecting to other enterprise resources, like enterprise resource planning or customer management system software.
The Java Transaction	An API for defining and managing transactions, including distributed

API (JTA)	transactions or transactions that cross multiple underlying data sources.

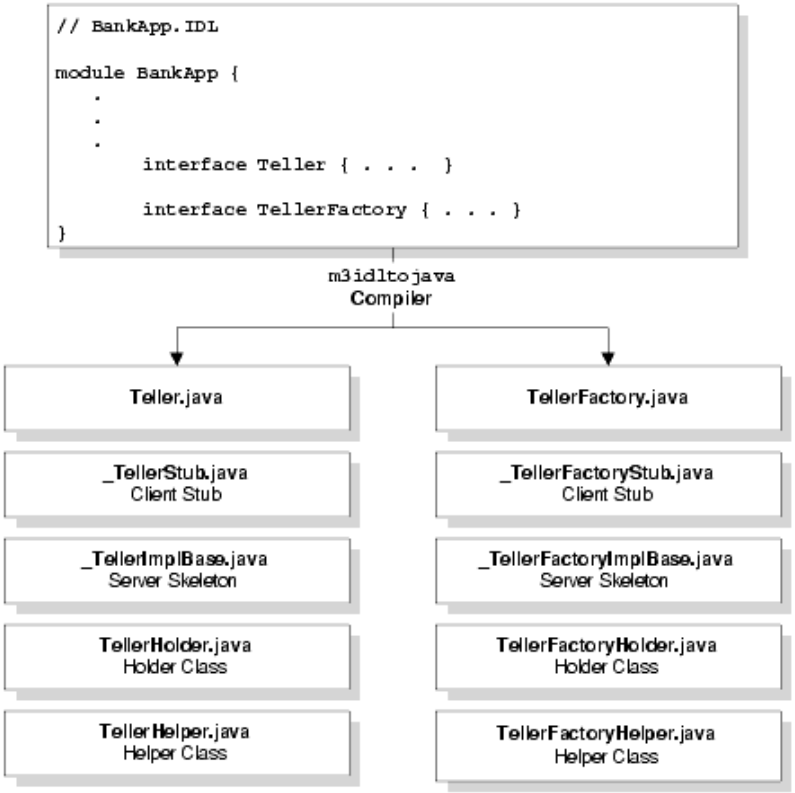
3. Explain how the JAVA servers are used in developing an application.

The basic steps involved in the creation of a server application are summarized in the following table:

Step 1: Compile the OMG IDL File for the Server Application
Step 2: Write the Methods That Implement Each Interface's Operations
Step 3: Create the Server Object
Step 4: Compile the Java Source Files
Step 5: Define the Object Activation and Transaction Policies
Step 6: Verify the Environment Variables
Step 7: Finish the Server Description File
Step 8: Deploy the Server Application

Step 1: Compile the OMG IDL File for the Server Application

The basic structure of the client and server portions of the application that runs in the WebLogic Enterprise domain are determined by statements in the application's OMG IDL file. When you compile your application's OMG IDL file, the m3idltojava compiler generates many files, some of which are shown in the following diagram:



Step 2: Write the Methods That Implement Each Interface's Operations

As the server application programmer, your task is to write the methods that implement the operations for each interface you have defined in your application's OMG IDL file.

The Java object implementation file contains:

- Method declarations for each operation specified in the OMG IDL file
- Your application's business logic
- Constructors for each interface implementation (implementing these is optional)
- Optionally,
  the com.beasys.Tobj_Servant.activate_object and com.beasys.Tobj_Servant.deactivate_object methods

  Within the activate_object and deactivate_object methods, you write code that performs any particular steps related to activating or deactivating an object. This includes reading and writing the object's durable state from and to disk, respectively.

Creating an Object Implementation File

Although you can create your server application's object implementation file manually, you can save time and effort by using the m3idltojava compiler to generate a file for each interface. The interface.java file contains Java signatures for the methods that implement each of the operations defined for your application's interfaces.

To take advantage of this shortcut, use the following steps:

1. Create a copy of the interface.java file, which was created when you compiled your OMG IDL file with the m3idltojava command, and name it interfaceImpl.java. For example, using the Bankapp sample file names, you would copy Teller.java to a new file named TellerImpl.java.
2. Open the new interfaceImpl.java file. For example, in the previously unedited TellerImpl.java file, we changed:

   public interface Teller extends org.omg.CORBA.Object {

   to:

   public class TellerImpl extends Bankapp._TellerImplBase {

   Bankapp._TellerImplBase is the class defined in the server skeleton file that was generated by the m3idltojava compiler for the Teller object.

3. For each method in TellerImpl.java, we added the public keyword. For example, we changed:

   float deposit(int accountID, float amount)

   to:

   public float deposit(int accountID, float amount)

Repeat this procedure to create interfaceImpl.java object implementation files for your interfaces, and add the business logic for your Java server application.

Step 3: Create the Server Object

In Java, you use a Server object to initialize and release the server application. You implement this Server object by creating a new class that derives from the com.beasys.Tobj.Server class and overrides the initialize and release methods. In the server application code, you can also write a public default constructor.

For example:

```
import com.beasys.Tobj.*;
/**
 * Provides code to initialize and stop the server invocation.
 * BankAppServerImpl is specified in the BankApp.xml input file
 * as the name of the Server object.
 */
public class BankAppServerImpl
        extends com.beasys.Tobj.Server {
        public boolean initialize(string[] args)
                throws com.beasys.TobjS.InitializeFailed;
        public boolean release()
                throws com.beasys.TobjS.ReleaseFailed;
}
```

In the XML-coded Server Description File, which you process with the buildjavaserver command, you identify the name of the Server object.

The create_servant method, used in the C++ environment of WebLogic Enterprise, is not used in the Java environment. In Java, objects are created dynamically, without prior knowledge of the classes being used. In the Java environment of WebLogic Enterprise, a servant factory is used to retrieve an implementation class, given the interface repository ID. This information is stored in a server descriptor file created by the buildjavaserver command for each implementation class. When a method request is received, and no servant is available for the interface, the servant factory looks up the interface and creates an object of the appropriate implementation class.

This collection of the object's implementation and data compose the run-time, active instance of the CORBA object.

When your Java server application starts, the TP Framework creates the Server object specified in the XML file. Then, the TP Framework invokes the initialize method. If the method returns true, the server application starts. If the method throws the com.beasys.TobjS.InitializeFailed exception, or returns false, the server application does not start.

When the server application shuts down, the TP Framework invokes the release method on the Server object.

Any command-line options specified in the CLOPT parameter for your specific server application in the SERVERS section of the WebLogic Enterprise domain's UBBCONFIG file are passed to the public boolean initialize(string[] args) operation as args. For more information about passing arguments to the server application, see the Administration Guide. For examples of passing arguments to the server application, see the *Guide to the Java Sample Applications*.

Within the initialize method, you can include code that does the following, if applicable:

- Creates and registers factories

- Allocates any machine resources
- Initializes any global variables needed by the server application
- Opens the databases used by the server application
- Opens the XA resource manager

Step 4: Compile the Java Source Files

- After you have implemented your application's objects and the Server object, use the javac compiler to create the bytecodes for all the class files that comprise your application. This set of files includes the *.java source files generated by the m3idltojava compiler, plus the object implementation files and server class file that you created.

Step 5: Define the Object Activation and Transaction Policies

As stated in the section Managing Object State, you determine what events cause an object to be deactivated by assigning object activation policies, transaction policies, and, optionally, using the application-controlled deactivation feature.

You specify default object activation and transaction policies in the Server Description File, which is expressed in XML, and you implement application-controlled deactivation via the com.beasys.Tobj.TP.deactivateEnable method in your Java code.

Step 6: Verify the Environment Variables

Several environment variables are defined by the WebLogic Enterprise software when the product is installed, but it is always a good idea to verify the following key environment variables prior to the buildjavaserver compilation step. The environment variables are:

- JAVA_HOME, the directory where the JDK is installed
- CLASSPATH, which must point to:
    - The location of the WebLogic Enterprise JAR archive, which contains all the class files
    - The location of the WebLogic Enterprise message catalogs
- TUXDIR, the directory where the WebLogic Enterprise software is installed

Step 7: Finish the Server Description File

- After you have compiled the Java source code and defined the environment variables, enter additional information in the XML-based Server Description File, and then supply the Server Description File as input to the buildjavaserver command.
- Edit your Server Description File to identify the Server object and the name of the file that will contain your Java application's server descriptor. This portion of the XML file is called the server declaration; its location in the file is immediately after the prolog. The required prolog contains the following two lines:
- <?xml version="1.0"?>
  <!DOCTYPE M3-SERVER SYSTEM "m3.dtd">

Step 8: Deploy the Server Application

You or the system administrator deploy the WebLogic Enterprise server application by using the procedure summarized in this section. For complete details on building and

deploying the WebLogic Enterprise Bankapp sample application, see the Guide to the Java Sample Applications.

To deploy the server application:

1. Place the server application JAR file in the directory listed in APPDIR. On NT systems, this directory must be on a local drive (not a networked drive). On Solaris, the directory can be local or remote.
2. If your Java server application uses an XA-compliant resource manager such as Oracle, you must build an XA-specific version of the JavaServer by using the buildXAJS command at a system prompt. Provide as input to the command the resource manager that is associated with the server. In your application's UBBCONFIG file, you also must use the JavaServerXA element in place ofJavaServer to associate the XA resource manager with a specified server group. See the Commands, System Process, and MIB Reference for details about the buildXAJS command.
3. Create the application's configuration file, also known as the UBBCONFIG file, in a text editor. Include the parameters to start JavaServer or JavaServerXA.

4. Explain the containers in JAVA with an illustration.

A container is a component which can contain other components inside itself. It is also an instance of a subclass of java.awt.Container. java.awt.Container extends java.awt.Component so containers are themselves components.

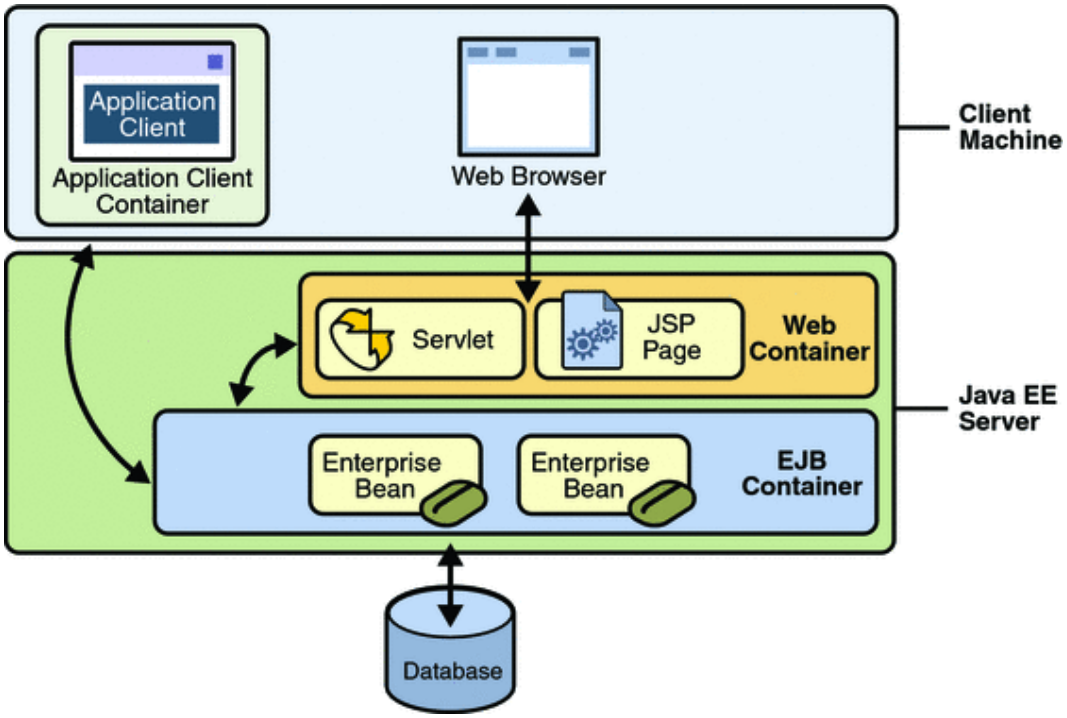
In general components are contained in a container. An applet is a container. Other containers include windows, frames, dialogs, and panels. Containers may contain other containers.

Every container has a LayoutManager that determines how different components are positioned within the container.

In short containers contain components. Components are positioned inside the container according to a LayoutManager. Since containers are themselves components, containers may by placed inside other containers. This is really a lot simpler than it sounds. Applets provide a ready-made container and a default LayoutManager, a FlowLayout.

Container Types

The deployment process installs Java EE application components in the Java EE containers as illustrated in the following figure.



- **Java EE server**: The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.
- **Enterprise JavaBeans (EJB) container**: Manages the execution of enterprise beans for Java EE applications. Enterprise beans and their container run on the Java EE server.
- **Web container**: Manages the execution of JSP page and servlet components for Java EE applications. Web components and their container run on the Java EE server.
- **Application client container**: Manages the execution of application client components. Application clients and their container run on the client.
- **Applet container**: Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.

5. Discuss about the play framework used in enterprise application development.

The Play framework is a clean option to bloated Enterprise Java stacks. It concentrates on designers' profit and targets tranquil architectures. Play framework is an immaculate friendly for nimble programming improvement. The Play framework's objective is to provide web applications improvement while staying with Java.

A Java framework without Pain

Play is an unadulterated Java framework and permits you to keep your favoured improvement on devices and libraries. On the off chance that you as of now utilize Java as an improvement stage you don't have to switch to an alternate dialect, an alternate IDE and different libraries. Simply switch to a more beneficial Java environment.

Fix the bug and hit Reload

The Java stage is scandalous for its low benefit, primarily because of rehashed and repetitive order bundle send cycles. That is the reason we re-examined the advancement cycle to make creating with play an effective methodology.

HTTP to code mapping

In case you're as of now utilizing an alternate Java Web framework like the Servlet API or the Struts framework, you have officially utilized a dynamic perspective of the HTTP convention with weird Java APIs and ideas. We think in an unexpected way. A Web application framework ought to provide for you full, direct get to HTTP and its ideas. This is a basic contrast in the middle of Play and other Java web application frameworks. HTTP, the Request/Response design, the REST compositional style, substance sort transaction, URI are all significant ideas for the play framework.

For example, tying a URI example to a Java call is only one line:

```
GET /customers/{id}  Clients.show
```

On the off chance that AJAX, REST and overseeing again/forward development between site pages are a percentage of the issues you confront in your normal web advancement, simply try play out.

Full Stack Application Framework

The play framework was at first enlivened by our own particular Java applications. It has all the devices required to make a current web application:

- Relational Database help through JDBC.
- Object-Relational Mapping utilizing Hibernate (with the JPA API).
- Integrated Cache help, with simple utilization of the dispersed memory stored framework if necessary.
- Straightforward Web administrations utilization either in JSON or XML.
- OpenId help for conveyed verification.
- Image control API.

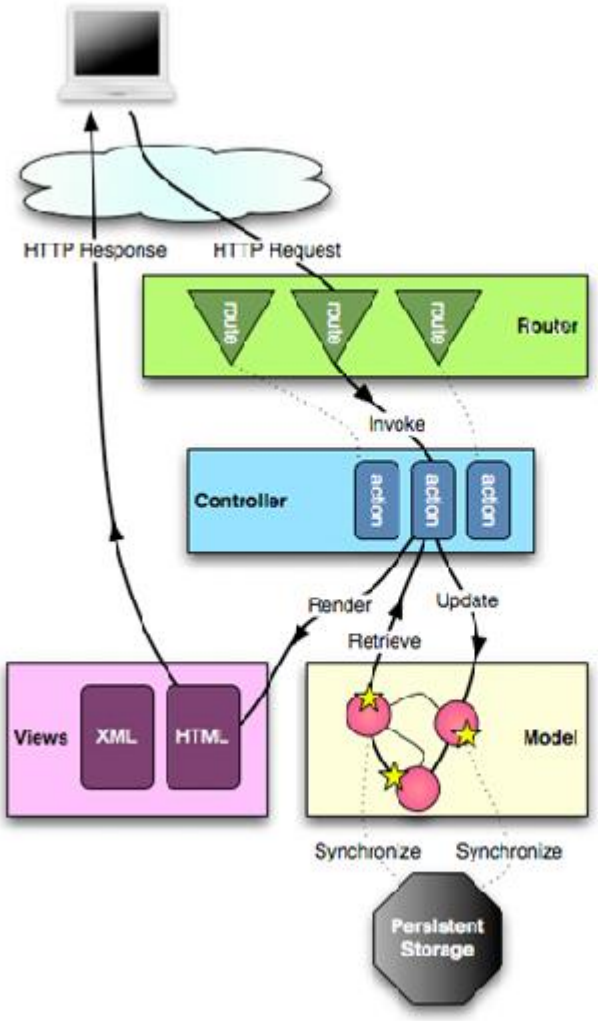
The particular construction modeling gives you a chance to join together a web application with numerous others. Because of re-use of modules, you can use your java code, templates and static assets, for example, Javascript and CSS records in a straightforward manner.

The request life cycle

The play structure is completely stateless and just Request/Response situated. All HTTP Requests take after the same way:

- A HTTP Request is gotten by the structure.
- The Router part tries to discover the most particular course ready to acknowledge this request. The comparing activity strategy is then summoned.
- The application code is executed.
- In the event that a complex view needs to be produced, a layout document is rendered.

The after-effect of the activity technique (HTTP Response code, Content) is then composed as a HTTP Response. This chart outlines the HTTP Request way:



The Standard Application Format
The format of a play application is standardized to keep things as straightforward as could be allowed.

(A) The application directory
This directory holds all executable antiquities: Java source code and perspectives layouts.

There are three standard bundles in the application directory, one for each one layer of the MVC structural example. You container obviously include your own particular bundles like for instance an utils packages.

Also, the perspectives bundle is further composed into sub-bundles:

- Tags, has application labels, e.g. reusable bits of layouts.
- One sees organizer for every Controller, by meeting layouts identified with every Controller are put away in their own particular sub-bundle.

(B) The open directory
Assets put away in general society directory are static possessions and are served straightforwardly by the Web server.

This directory is part into three standard sub-registries: for pictures, CSS templates and Javascript records. You ought to attempt to arrange your static holdings like this to keep all play applications predictable.

(C) The conf directory
The conf directory holds all arrangement documents for the application.

There are two obliged arrangements documents:

- application.conf, the primary arrangement document for the application. It holds standard design alternatives.
- routes, the courses definition document.

6. Create a web application using JAVA servlets.

Why we need Servlet and JSPs?

Web servers are good for static contents HTML pages but they don't know how to generate dynamic content or how to save data into databases, so we need another tool that we can use to generate dynamic content. There are several programming languages for dynamic content like PHP, Python, Ruby on Rails, Java Servlets and JSPs.

Java Servlet and JSPs are server side technologies to extend the capability of web servers by providing support for dynamic response and data persistence.

Java Web Development

First Web Application with Servlet and JSP

We will use "Eclipse IDE for Java EE Developers" for creating our first servlet application. Since servlet is a server side technology, we will need a web container that supports Servlet technology, so we will use Apache Tomcat server. For ease of development, we can add configure Tomcat with Eclipse, it helps in easy deployment and running applications.

Go to Eclipse Preference and select Server Runtime Environments and select the version of your tomcat server, mine is Tomcat 7.

Provide the apache tomcat directory location and JRE information to add the runtime environment.

Now go to the Servers view and create a new server like below image pointing to the above added runtime environment.

Now we are ready with our setup to create first servlet and run it on tomcat server.

Select File > New > Dynamic Web Project and use below image to provide runtime as the server we added in last step and module version as 3.0 to create our servlet using Servlet 3.0 specs.

You can directly click Finish button to create the project or you can click on Next buttons to check for other options.

Now select File > New > Servlet and use below image to create our first servlet. Again we can click finish or we can check other options through next button.

When we click on Finish button, it generates our Servlet skeleton code, so we don't need to type in all the different methods and imports in servlet and saves us time.

Now we will add some HTML with dynamic data code in **doGet()** method that will be invoked for HTTP GET request. Our first servlet looks like below.

```
package com.journaldev.first;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class FirstServlet
 */
@WebServlet(description = "My First Servlet", urlPatterns = { "/FirstServlet" ,
"/FirstServlet.do"}, initParams =
{@WebInitParam(name="id",value="1"),@WebInitParam(name="name",value="pankaj")})
public class FirstServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;
        public static final String HTML_START="<html><body>";
        public static final String HTML_END="</body></html>";

    /**
     * @see HttpServlet#HttpServlet()
     */
    public FirstServlet() {
        super();
        // TODO Auto-generated constructor stub
```

```
   }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		PrintWriter out = response.getWriter();
		Date date = new Date();
		out.println(HTML_START + "<h2>Hi
There!</h2><br/><h3>Date="+date +"</h3>"+HTML_END);
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		// TODO Auto-generated method stub
	}

}
```

Before Servlet 3, we need to provide the url pattern information in web application deployment descriptor but servlet 3.0 uses **java annotations** that is easy to understand and chances of errors are less.

Now chose Run > Run on Server option from servlet editor window

After clicking finish, browser will open in Eclipse and we get following HTML page.
You can refresh it to check that Date is dynamic and keeps on changing, you can open it outside of Eclipse also in any other browser.

So servlet is used to generate HTML and send it in response, if you will look into the doGet() implementation, we are actually creating an HTML document as writing it in response PrintWriter object and we are adding dynamic information where we need it.

It's good for start but if the response is huge with lot of dynamic data, it's error prone and hard to read and maintain. This is the primary reason for introduction of JSPs.

JSP is also server side technology and it's like HTML with additional features to add dynamic content where we need it.
JSPs are good for presentation because it's easy to write because it's like HTML. Here is our first JSP program that does the same thing like above servlet.

If we run above JSP, we get output like below image.

7. Explain how the Web frameworks are used in JAVA application development.

Java Frameworks are nothing but large bodies of predefined Java code which you can apply to your own code to solve your problem in a specific domain. You can use a framework by calling its methods, inheritance, by providing "callbacks", listeners, or other implementations of the Observer pattern.

How Frameworks came into existence?

Late in 1990's the applications were widely developed using JEE standards. The premise of J2EE was multi-platform/ multi-vendor, if you can code according to the J2EE standards you can deploy your application on any J2EE application server irrespective of platform. Running your code on any application server provides you with many benefits like – transaction management, messaging, mailing, directory interface etc. But as nothing comes easy in this world, working with J2EE also had some difficulties.

- Very Complex: Enterprise Java Bean was developed for reducing the complexity in J2EE applications. But it didn't succeed in its aim in implementation. Reason behind it is that, while writing a component it is required to write a set of XML files, home interfaces, remote/ local interfaces, etc.
- 'look-up' problem: Whenever a component depended upon another component, it had to look up for the components it depended upon by itself. This component 'look-up' happens only by name, so the name of the dependency was hard-coded in the component.
- Heavy weight: As all features like clustering, remote controlling, etc., were supported, you have to configure them, regardless of the fact that you need them or not. This will make your applications bloated.

Advantages of using Java Frameworks

- **Efficiency**: Tasks that generally take you hours and hundreds of lines of code to compose, can now be done in minutes with pre-built functions. Development becomes a lot easier, so if it's much easier, it's much quicker, and subsequently more effective.
- **Security:** An extensively used framework will generally have large security applications. The big benefit is the neighborhood behind that framework, where users usually end up being long-lasting testers. If you find a vulnerability or a security hole, you can go to the framework's web site and let them know so that it can be fixed.
- **Expense**: Most popular structures are complimentary and so it helps the developer to code faster. If the coding is done faster the expense for the final client will certainly be smaller in every aspect, be it time or effort. Moreover the maintenance cost is also low.
- **Support**: As any other distributed tool, a framework generally includes documents, a support group or large-community online forums where you can acquire quick responses.

8. Describe about the role of spring frameworks in enterprise application development with an example.

Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, and reusable code.

Spring framework is an open source Java platform. It was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.

Spring is lightweight when it comes to size and transparency. The basic version of Spring framework is around 2MB.

The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promotes good programming practices by enabling a POJO-based programming model.

Benefits of Using the Spring Framework

Following is the list of few of the great benefits of using Spring Framework −

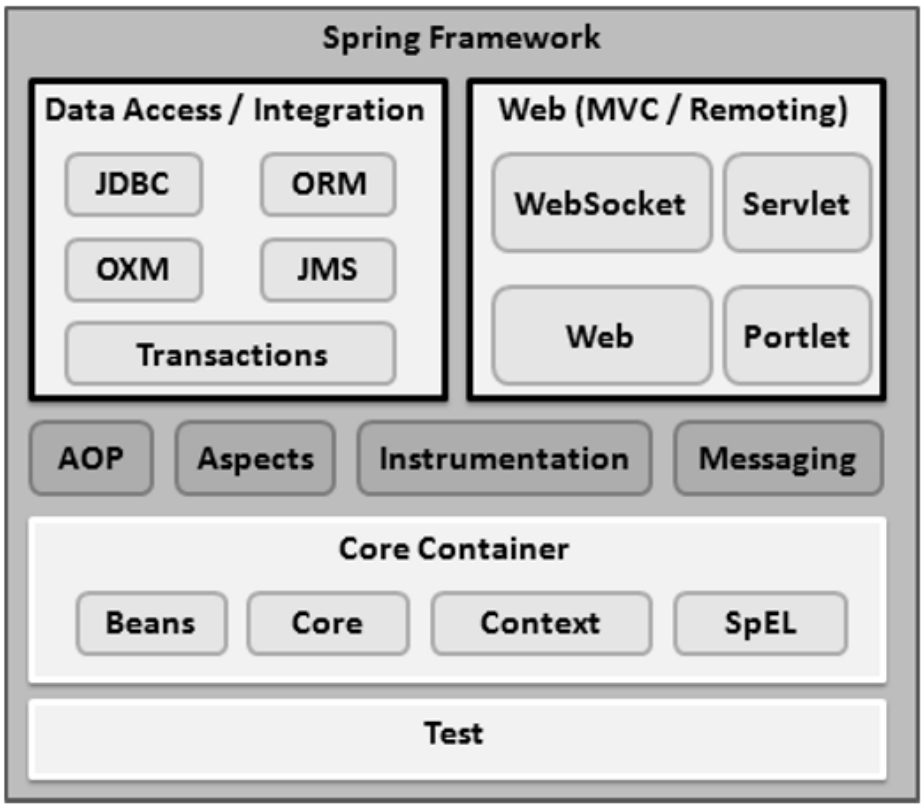
- Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.
- Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about the ones you need and ignore the rest.
- Spring does not reinvent the wheel, instead it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies.
- Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBeanstyle POJOs, it becomes easier to use dependency injection for injecting test data.
- Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.
- Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

- Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
- Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).
- Dependency Injection (DI)
- The technology that Spring is most identified with is the Dependency Injection (DI) flavor of Inversion of Control. The Inversion of Control (IoC) is a general concept, and it can be expressed in many different ways. Dependency Injection is merely one concrete example of Inversion of Control.
- 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 helps in gluing these classes together and at the same time keeping them independent.
- What is dependency injection exactly? Let's look at these two words separately. Here the dependency part translates into an association between two classes. For example, class A is dependent of class B. Now, let's look at the second part, injection. All this means is, class B will get injected into class A by the IoC.
- Dependency injection can happen in the way of passing parameters to the constructor or by post-construction using setter methods. As Dependency Injection is the heart of Spring Framework, we will explain this concept in a separate chapter with relevant example.
- Aspect Oriented Programming (AOP)
- One of the key components of Spring is the Aspect Oriented Programming (AOP) framework. 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 including logging, 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. DI helps you decouple your application objects from each other, while AOP helps you decouple cross-cutting concerns from the objects that they affect.

- The AOP module of Spring Framework provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.

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 implementation of the factory pattern.
- The Context module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.

- The SpEL module provides a powerful expression language for querying and manipulating an object graph at runtime.

Data Access/Integration

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows −

- The JDBC module provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
- The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
- The OXM module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
- The Java Messaging Service JMS module contains features for producing and consuming messages.
- The Transaction module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Web

The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules the details of which are as follows −

- The Web module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
- The Web-MVC module contains Spring's Model-View-Controller (MVC) implementation for web applications.
- The Web-Socket module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
- The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

Step 1 - Setup Java Development Kit (JDK)

You can download the latest version of SDK from Oracle's Java site − Java SE Downloads. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.

If you are running Windows and have installed the JDK in C:\jdk1.6.0_15, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.6.0_15\bin;%PATH%
set JAVA_HOME=C:\jdk1.6.0_15
```

Alternatively, on Windows NT/2000/XP, you will have to right-click on My Computer, select Properties → Advanced → Environment Variables. Then, you will have to update the PATH value and click the OK button.

On Unix (Solaris, Linux, etc.), if the SDK is installed in /usr/local/jdk1.6.0_15 and you use the C shell, you will have to put the following into your .cshrc file.

```
setenv PATH /usr/local/jdk1.6.0_15/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.6.0_15
```

Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, you will have to compile and run a simple program to confirm that the IDE knows where you have installed Java. Otherwise, you will have to carry out a proper setup as given in the document of the IDE.

Step 2 - Install Apache Common Logging API

You can download the latest version of Apache Commons Logging API from https://commons.apache.org/logging/. Once you download the installation, unpack the binary distribution into a convenient location. For example, in C:\commons-logging-1.1.1 on Windows, or /usr/local/commons-logging-1.1.1 on Linux/Unix. This directory will have the following jar files and other supporting documents, etc.

Step 3 - Setup Eclipse IDE

All the examples in this tutorial have been written using Eclipse IDE. So we would suggest you should have the latest version of Eclipse installed on your machine.

To install Eclipse IDE, download the latest Eclipse binaries from https://www.eclipse.org/downloads/. Once you download the installation, unpack the binary distribution into a convenient location. For example, in C:\eclipse on Windows, or /usr/local/eclipse on Linux/Unix and finally set PATH variable appropriately.

Eclipse can be started by executing the following commands on Windows machine, or you can simply double-click on eclipse.exe

```
%C:\eclipse\eclipse.exe
```

Eclipse can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine −

```
$/usr/local/eclipse/eclipse
```

Step 4 - Setup Spring Framework Libraries

Now if everything is fine, then you can proceed to set up your Spring framework. Following are the simple steps to download and install the framework on your machine.

- Make a choice whether you want to install Spring on Windows or Unix, and then proceed to the next step to download .zip file for Windows and .tz file for Unix.
- Download the latest version of Spring framework binaries from https://repo.spring.io/release/org/springframework/spring

9.Explain about ORM layer and it's usage in application development.

Object-relational mapping, in the purest sense, is a programming technique that supports the conversion of incompatible types in object-oriented programming languages, specifically between a data store and programming objects. You can use an ORM framework to persist model objects to a relational database and retrieve them, and the ORM framework will take care of converting the data between the two otherwise incompatible states. Most ORM tools rely heavily on metadata about both the database and objects, so that the objects need to know nothing about the database and the database doesn't need to know anything about how the data is structured in the application. ORM provides a clean separation of concerns in a well-designed data application, and the database and application can each work with data in its native form.

TIP: Nicknames and acronyms used for "object-relational mapping" include ORM, OR/M, and O/R mapping. Although ORM seems to be the term most commonly used in the .NET world, you'll often see the others in books and articles. We'll stick with ORM, mostly because it is the easiest to type!

The key feature of ORM is the mapping it uses to bind an object to its data in the database. Mapping expresses how an object and its properties and behaviors are related to one or more tables and their fields in the database. An ORM uses this mapping information to manage the process of converting data between its database and object forms, and generating the SQL for a relational database to insert, update, and delete data in response to changes the application makes to data objects.

ORM performs the rather amazing task of managing the application's interactions with the database. Once you've used an ORM's tools to create mappings and objects for use in an application, those objects completely manage the application's data access needs. You won't have to write any other low-level data access code. Strictly speaking, you could still write low-level data access code to supplement the ORM data objects, but this adds a significant layer of complexity to an application that we've rarely found necessary when using a robust ORM tool. It is better to stick to one or the other and keep the application simpler and more maintainable.

There are a number of benefits to using an ORM for development of databased applications and here's four:

1. **Productivity:** The data access code is usually a significant portion of a typical application, and the time needed to write that code can be a significant portion of the overall development schedule. When using an ORM tool, the amount of code is unlikely to be reduced—in fact, it might even go up—but the ORM tool generates 100% of the data access code automatically based on the data model you define, in mere moments.
2. **Application design:** A good ORM tool designed by very experienced software architects will implement effective design patterns that almost force you to use good programming practices in an application. This can help support a clean separation of concerns and independent development that allows parallel, simultaneous development of application layers.
3. **Code Reuse**: If you create a class library to generate a separate DLL for the ORM-generated data access code, you can easily reuse the data objects in a variety of applications. This way, each of the applications that use the class library need have no data access code at all.
4. **Application Maintainability**: All of the code generated by the ORM is presumably well-tested, so you usually don't need to worry about testing it extensively. Obviously you need to make sure that the code does what you need, but a widely used ORM is likely to have code banged on by many developers at all skill levels. Over the long term, you can refactor the database schema or the model definition without affecting how the application uses the data objects.

One potential downside to using an ORM is performance. It is very likely that the data access code generated by the ORM is more complex than you'd typically write for an application. This is because most ORMs are designed to handle a wide variety of data-use scenarios, far more than any single application is ever likely to use. Complex code generally means slower performance, but a well-designed ORM is likely to generate well-tuned code that minimizes the performance impact. Besides, in all but the most data-intensive applications the time spent interacting with the database is a relatively small portion of the time the user spends using the application. Nevertheless, we've never found a case where the small performance hit wasn't worth the other benefits of using an ORM. You should certainly test it for your data and applications to make sure that the performance is acceptable.

ORMs are being hyped for being the solution to Data Access problems. Personally, after having used them in an Enterprise Project, they are far from being the solution for Enterprise Application Development. Maybe they work in small projects. Here are the problems we have experienced with them specifically nHibernate:

1. **Configuration:** ORM technologies require configuration files to map table schemas into object structures. In large enterprise systems the configuration grows very quickly and becomes extremely difficult to create and manage. Maintaining the configuration also gets tedious and unmaintainable as business requirements and models constantly change and evolve in an agile environment.
2. **Custom Queries:** The ability to map custom queries that do not fit into any defined object is either not supported or not recommended by the framework providers. Developers are forced to find work-arounds by writing adhoc objects and queries, or writing custom code to get the data they need. They may have to use Stored Procedures on a regular basis for anything more complex than a simple Select.
3. **Proprietery binding**: These frameworks require the use of proprietary libraries and proprietary object query languages that are not standardized in the computer science

industry. These proprietary libraries and query languages bind the application to the specific implementation of the provider with little or no flexibility to change if required and no interoperability to collaborate with each other.

4. **Object Query Languages:** New query languages called Object Query Languages are provided to perform queries on the object model. They automatically generate SQL queries against the databse and the user is abstracted from the process. To Object Oriented developers this may seem like a benefit since they feel the problem of writing SQL is solved. The problem in practicality is that these query languages cannot support some of the intermediate to advanced SQL constructs required by most real world applications. They also prevent developers from tweaking the SQL queries if necessary.
5. **Performance:** The ORM layers use reflection and introspection to instantiate and populate the objects with data from the database. These are costly operations in terms of processing and add to the performance degradation of the mapping operations. The Object Queries that are translated to produce unoptimized queries without the option of tuning them causing significant performance losses and overloading of the database management systems. Performance tuning the SQL is almost impossible since the frameworks provide little flexiblity over controlling the SQL that gets autogenerated.
6. **Tight coupling**: This approach creates a tight dependancy between model objects and database schemas. Developers don't want a one-to-one correlation between database fields and class fields. Changing the database schema has rippling affects in the object model and mapping configuration and vice versa.
7. **Caches:** This approach also requires the use of object caches and contexts that are necessary to maintian and track the state of the object and reduce database roundtrips for the cached data. These caches if not maintained and synchrnonized in a multi-tiered implementation can have significant ramifications in terms of data-accuracy and concurrency. Often third party caches or external caches have to be plugged in to solve this problem, adding extensive burden to the data-access layer.

10. Discuss about Web containers used in JAVA application.

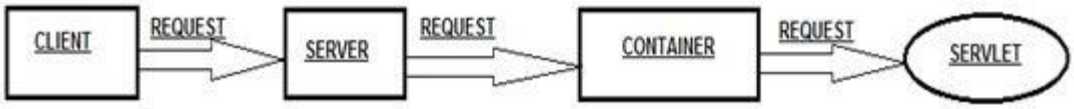
Web Container is an java application that controls servlet. Servlet do not have a main() method, So they require a container to load them. Container is a place where servlet gets deployed.

When a client sends a request to web server that contain a servlet, server sends that request to container rather than to servlet directly. Container then finds out the requested servlet and pass the Http Request and response to servlet and loads the servlet methods i.e. doGet() or do Post().

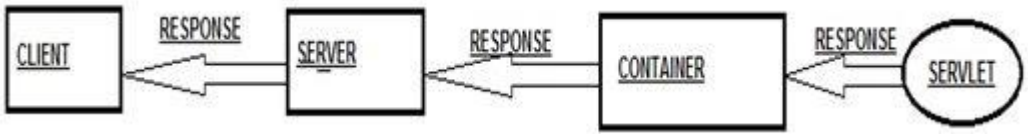
Example of a web container is Tomcat.

Diagrams to show the request made by the client to the server and response received by the client.

1. Request made by client to server



2. Response received by client



A web container like apache tomcat or jetty just implements the JSP and Servlet specification of Java EE. However there are more specifications within Java EE, mainly the EJB specification. A servlet container implements only the JSP and Servlet specification, and doesn't implement EJB specification. Hence we cannot deploy and execute EJB applications in a servlet container like Apache Tomcat or Jetty. Instead you need a full application server. An application server contains an EJB container as well as servlet container and implements all the required specifications of Java EE. Examples of application servers are Glassfish Application Server, which is a Java EE reference implementation from Oracle, Websphere Application Server by IBM, JBoss Application Server by Red Hat inc etc.

A Web application runs within a Web container of a Web server. The Web container provides the runtime environment through components that provide naming context and life cycle management. Some Web servers may also provide additional services such as security and concurrency control. A Web server may work with an EJB server to provide some of those services. A Web server, however, does not need to be located on the same machine as an EJB server.

Web applications are composed of web components and other data such as HTML pages. Web components can be servlets, JSP pages created with the JavaServer Pages™ technology, web filters, and web event listeners. These components typically execute in a web server and may respond to HTTP requests from web clients. Servlets, JSP pages, and filters may be used to generate HTML pages that are an application's user interface. They may also be used to generate XML or other format data that is consumed by other application components.

UNIT V

1. What is Java beans? Enumerate its advantages. What are the steps involved to create a Java Bean?

- A *Java Bean* is a software component that has been designed to be reusable in a variety of different environments.
- It may perform a simple function, such as checking the spelling of a document, or a complex function, such as forecasting the performance of a stock portfolio.

Advantages of Java Beans

- A Bean obtains all the benefits of Java's "write-once, run-anywhere" paradigm.

- The properties, events, and methods of a Bean that are exposed to an application builder tool can be controlled.
- A Bean may be designed to operate correctly in different locales, which makes it useful in global markets.

Steps for creating of a new bean:

(i) Start--new project. A new project dialog window appear.
(ii) In the new project dialog window select general in categories and java application in project box.
(iii) Then click Next button and specify the project name and project path too.
(iv) Then click Finish button.

2. Discuss JAR files in detail with suitable example.

JAR Files:
The JAR Utility
A utility is used to generate a JAR file. Its syntax is shown here:
jar options files
Creating a JAR File
The following command creates a JAR file named Xyz.jar that contains all of the .class and .gif files in the current directory:
jar cf Xyz.jar *.class *.gif
If a manifest file such as Yxz.mf is available, it can be used with the following command:
jar cfm Xyz.jar Yxz.mf *.class *.gif
Tabulating the Contents of a JAR File
The following command lists the contents of Xyz.jar:
jar tf Xyz.jar
Extracting Files from a JAR File
The following command extracts the contents of Xyz.jar and places those files in the current directory:
jar xf Xyz.jar
Updating an Existing JAR File
The following command adds the file file1.class to Xyz.jar:
jar -uf Xyz.jar file1.class
Recursing Directories
The following command adds all files below directoryX to Xyz.jar:
jar -uf Xyz.jar -C directoryX *

3. Explain Design patterns and properties in detail.

A property is a subset of a Bean's state. The values assigned to the properties determine the behavior and appearance of that component. There are three types of properties: simple, Boolean, and indexed.

Simple Properties: A simple property has a single value. It can be identified by the following design patterns, where N is the name of the property and T is its type.

```
public T getN( );
public void setN(T arg);
```

A read/write property has both of these methods to access its values. A read-only property has only a get method. A write-only property has only a set method.

Example for Simple Properties:

```
public class Box {
private double depth, height, width;
public double getDepth( ) {
return depth;
}
public void setDepth(double d) {
depth = d;
}
public double getHeight( ) {
return height;
}
public void setHeight(double h) {
height = h;
}
public double getWidth( ) {
return width;
}
public void setWidth(double w) {
width = w;
}
}
```

Boolean Properties: A Boolean property has a value of true or false. It can be identified by the following design patterns, where N is the name of the property:

```
public boolean isN( );
public boolean getN( );
public void setN(boolean value);
```

Either the first or second pattern can be used to retrieve the value of a Boolean property. However, if a class has both of these methods, the first pattern is used.

Example:

```
public class Line {
private boolean dotted = false;
public boolean isDotted( ) {
return dotted;
}
public void setDotted(boolean dotted) {
this.dotted = dotted;
}
}
```

Indexed Properties: An indexed property consists of multiple values. It can be identified by the following design patterns, where N is the name of the property and T is its type:

```
public T getN(int index);
public void setN(int index, T value);
public T[ ] getN( );
public void setN(T values[ ]);
```

Example:

```
public class PieChart {
private double data[ ];
public double getData(int index) {
return data[index];
}
```

```
public void setData(int index, double value) {
data[index] = value;
}
public double[ ] getData( ) {
return data;
}
public void setData(double[ ] values) {
data = new double[values.length];
System.arraycopy(values, 0, data, 0, values.length);
}
}
```

4. Explain BeanInfo interface in detail.

Design patterns were used to determine the information that was provided to a Bean user. This section describes how a developer can use the **BeanInfo** interface to explicitly control this process.

This interface defines several methods, including these:

PropertyDescriptor[ ] getPropertyDescriptors( )
EventSetDescriptor[ ] getEventSetDescriptors( )
MethodDescriptor[ ] getMethodDescriptors( )

They return arrays of objects that provide information about the properties, events, and methods of a Bean. By implementing these methods, a developer can designate exactly what is presented to a user.

SimpleBeanInfo is a class that provides default implementations of the **BeanInfo** interface, including the three methods just shown. You may extend this class and override one or more of them. The following listing shows how this is done for the **Colors** Bean that was developed earlier. **ColorsBeanInfo** is a subclass of **SimpleBeanInfo**. It overrides **getPropertyDescriptors( )** in order to designate which properties are presented to a Bean user. This method creates a **PropertyDescriptor** object for the **rectangular** property. The **PropertyDescriptor** constructor that is used is shown here:

PropertyDescriptor(String *property*, Class *beanCls*)
throws IntrospectionException

Here, the first argument is the name of the property, and the second argument is the class of the Bean.

```
// A Bean information class.
package sunw.demo.colors;
import java.beans.*;
public class ColorsBeanInfo extends SimpleBeanInfo {
public PropertyDescriptor[] getPropertyDescriptors() {
try {
PropertyDescriptor rectangular = new
PropertyDescriptor("rectangular", Colors.class);
PropertyDescriptor pd[] = {rectangular};
return pd;
}
catch(Exception e) {
}
return null;
}
```

}
You must compile this file from the **BDK\demo** directory or set **CLASSPATH** so that it includes **c:\bdk\demo**. If you don't, the compiler won't find the **Colors.class** file properly. After this file is successfully compiled, the **colors.mft** file can be updated, as shown here:
Name: sunw/demo/colors/ColorsBeanInfo.class
Name: sunw/demo/colors/Colors.class
Java-Bean: True
Use the JAR tool to create a new **colors.jar** file. Restart the BDK and create an instance of the **Colors** Bean in the BeanBox. The introspection facilities are designed to look for a **BeanInfo** class. If it exists, its behavior explicitly determines the information that is presented to a Bean user. Otherwise, design patterns are used to infer this information.

5. Explain Java Bean API in detail.

Interface	Description
AppletInitializer	This interface is designed to work in collusion with java.beans.Beans.instantiate.
BeanInfo	A bean implementor who wishes to provide explicit information about their bean may provide a BeanInfo class that implements this BeanInfo interface and provides explicit information about the methods, properties, events, etc, of their bean.
Customizer	A customizer class provides a complete custom GUI for customizing a target Java Bean.
DesignMode	This interface is intended to be implemented by, or delegated from, instances of java.beans.beancontext.BeanContext, in order to propagate to its nested hierarchy of java.beans.beancontext.BeanContextChild instances, the current "designTime" property.
ExceptionListener	An ExceptionListener is notified of internal exceptions.
PropertyChangeListener	A "PropertyChange" event gets fired whenever a bean changes a "bound" property.
PropertyEditor	A PropertyEditor class provides support for GUIs that want to allow users to edit a property value of a given type.
VetoableChangeListener	A VetoableChange event gets fired whenever a bean changes a "constrained" property.
Visibility	Under some circumstances a bean may be run on servers where a GUI is not availabl

Class	Description
BeanDescriptor	A BeanDescriptor provides global information about a "bean", including its Java class, its displayName, etc.
Beans	This class provides some general purpose beans control methods.
DefaultPersistence Delegate	The `DefaultPersistenceDelegate` is a concrete implementation of the abstract `PersistenceDelegate` class

	and is the delegate used by default for classes about which no information is available.
Encoder	An `Encoder` is a class which can be used to create files or streams that encode the state of a collection of JavaBeans in terms of their public APIs.
EventHandler	The `EventHandler` class provides support for dynamically generating event listeners whose methods execute a simple statement involving an incoming event object and a target object.
EventSetDescriptor	An EventSetDescriptor describes a group of events that a given Java bean fires.
Expression	An `Expression` object represents a primitive expression in which a single method is applied to a target and a set of arguments to return a result - as in "`a.getFoo()`".
FeatureDescriptor	The FeatureDescriptor class is the common baseclass for PropertyDescriptor, EventSetDescriptor, and MethodDescriptor, etc.
IndexedPropertyChangeEvent	An "IndexedPropertyChange" event gets delivered whenever a component that conforms to the JavaBeans specification (a "bean") changes a bound indexed property.
IndexedPropertyDescriptor	An IndexedPropertyDescriptor describes a property that acts like an array and has an indexed read and/or indexed write method to access specific elements of the array.
Introspector	The Introspector class provides a standard way for tools to learn about the properties, events, and methods supported by a target Java Bean.
MethodDescriptor	A MethodDescriptor describes a particular method that a Java Bean supports for external access from other components.
ParameterDescriptor	The ParameterDescriptor class allows bean implementors to provide additional information on each of their parameters, beyond the low level type information provided by the java.lang.reflect.Method class.
PersistenceDelegate	The PersistenceDelegate class takes the responsibility for expressing the state of an instance of a given class in terms of the methods in the class's public API.
PropertyChangeEvent	A "PropertyChange" event gets delivered whenever a bean changes a "bound" or "constrained" property.
PropertyChangeListenerProxy	A class which extends the `EventListenerProxy` specifically for adding a named `PropertyChangeListener`.
PropertyChangeSupport	This is a utility class that can be used by beans that support bound properties.
PropertyDescriptor	A PropertyDescriptor describes one property that a Java Bean exports via a pair of accessor methods.

PropertyEditorManager	The PropertyEditorManager can be used to locate a property editor for any given type name.
PropertyEditorSupport	This is a support class to help build property editors.
SimpleBeanInfo	This is a support class to make it easier for people to provide BeanInfo classes.
Statement	A `Statement` object represents a primitive statement in which a single method is applied to a target and a set of arguments - as in "`a.setFoo(b)`".

6. Explain reserved sockets and proxy servers in detail.

Reserved Sockets: Once connected, a higher-level protocol ensues, which is dependent on which port you are using. TCP/IP reserves the lower 1,024 ports for specific protocols. Many of these will seem familiar to you if you have spent any time surfing the Internet. Port number 21 is for FTP, 23 is for Telnet, 25 is for e-mail, 79 is for finger, 80 is for HTTP, 119 is for netnews—and the list goes on. It is up to each protocol to determine how a client should interact with the port. For example, HTTP is the protocol that web browsers and servers use to transfer hypertext pages and images. It is quite a simple protocol for a basic page-browsing web server. Here's how it works. When a client requests a file from an HTTP server, an action known as a *hit*, it simply prints the name of the file in a special format to a predefined port and reads back the contents of the file. The server also responds with a status code number to tell the client whether the request can be fulfilled and why.
Proxy Servers: A *proxy server* speaks the client side of a protocol to another server. This is often required when clients have certain restrictions on which servers they can connect to. Thus, a client would connect to a proxy server, which did not have such restrictions, and the proxy server would in turn communicate for the client. A proxy server has the additional ability to filter certain requests or cache the results of those requests for future use. A caching proxy HTTP server can help reduce the bandwidth demands on a local network's connection to the Internet. When a popular web site is being hit by hundreds of users, a proxy server can get the contents of the web server's popular pages once, saving expensive internetwork transfers while providing faster access to those pages to the clients.

7. Explain InetAddress in detail.

Whether you are making a phone call, sending mail, or establishing a connection across the Internet, addresses are fundamental. The **InetAddress** class is used to encapsulate both the numerical IP address we discussed earlier and the domain name for that address. You interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address. The **InetAddress** class hides the number inside. As of Java 2, version 1.4, **InetAddress** can handle both IPv4 and IPv6 addresses.
The **InetAddress** class has no visible constructors. To create an **InetAddress** object, you have to use one of the available factory methods. *Factory methods* are merely a convention whereby static methods in a class return an instance of that class. This is done in lieu of overloading a constructor with various parameter lists when having unique method names makes the results much clearer. Three commonly used **InetAddress** factory methods are shown here.
static InetAddress getLocalHost( )

throws UnknownHostException
static InetAddress getByName(String *hostName*)
throws UnknownHostException
static InetAddress[ ] getAllByName(String *hostName*)
throws UnknownHostException
The **getLocalHost( )** method simply returns the **InetAddress** object that represents the local host. The **getByName( )** method returns an **InetAddress** for a host name passed to it. If these methods are unable to resolve the host name, they throw an **UnknownHostException**.

```
// Demonstrate InetAddress.
import java.net.*;
class InetAddressTest
{
public static void main(String args[]) throws UnknownHostException {
InetAddress Address = InetAddress.getLocalHost();
System.out.println(Address);
Address = InetAddress.getByName("osborne.com");
System.out.println(Address);
InetAddress SW[] = InetAddress.getAllByName("www.nba.com");
for (int i=0; i<SW.length; i++)
System.out.println(SW[i]);
}
}
```

8. Explain TCP/IP Client Sockets in detail.(DEC2015)

TCP/IP sockets are used to implement reliable, bidirectional, persistent, point-to- point, stream-based connections between hosts on the Internet. A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet.

There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients. The **ServerSocket** class is designed to be a "listener," which waits for clients to connect before doing anything. The **Socket** class is designed to connect to server sockets and initiate protocol exchanges.

The creation of a **Socket** object implicitly establishes a connection between the client and server. There are no methods or constructors that explicitly expose the details of establishing that connection. Here are two constructors used to create client sockets:

Socket(String *hostName*, int *port*) Creates a socket connecting the local host to the named host and port; can throw an **UnknownHostException** or an **IOException**.

Socket(InetAddress *ipAddress*, int *port*) Creates a socket using a preexisting **InetAddress** object and a port; can throw an **IOException**.

A socket can be examined at any time for the address and port information associated with it, by use of the following methods:

InetAddress getInetAddress( ) Returns the **InetAddress** associated with the **Socket** object.

int getPort( ) Returns the remote port to which this **Socket** object is connected.

int getLocalPort( ) Returns the local port to which this **Socket** object is connected.

Once the **Socket** object has been created, it can also be examined to gain access to the input and output streams associated with it. Each of these methods can throw an **IOException** if the sockets have been invalidated by a loss of connection on the Net.

InputStream getInputStream( ) Returns the **InputStream** associated with the invoking socket.

OutputStream getOutputStream( ) Returns the **OutputStream** associated with the invoking socket.

9. Explain TCP/IP Server Sockets in detail.

Java has a different socket class that must be used for creating server applications. The **ServerSocket** class is used to create servers that listen for either local or remote client programs to connect to them on published ports. Since the Web is driving most of the activity on the Internet, this section develops an operational web (http) server.
ServerSockets are quite different from normal **Socket**s. When you create a **ServerSocket**, it will register itself with the system as having an interest in client connections. The constructors for **ServerSocket** reflect the port number that you wish to accept connections on and, optionally, how long you want the queue for said port to be. The queue length tells the system how many client connections it can leave pending before it should simply refuse connections. The default is 50. The constructors might throw an **IOException** under adverse conditions.
Here are the constructors:
ServerSocket(int *port*) - Creates server socket on the specified port with a queue length of 50.
ServerSocket(int *port*, int *maxQueue*) - Creates a server socket on the specified port with a maximum queue length of *maxQueue.*
ServerSocket(int *port*, int *maxQueue*, InetAddress *localAddress*) - Creates a server socket on the specified port with a maximum queue length of *maxQueue.* On a multihomed host, *localAddress* specifies the IP address to which this socket binds.

10. Explain Datagram in detail with an example.

Datagrams are bundles of information passed between machines. They are somewhat like a hard throw from a well-trained but blindfolded catcher to the third baseman. Once the datagram has been released to its intended target, there is no assurance that it will arrive or even that someone will be there to catch it. Likewise, when the datagram is received, there is no assurance that it hasn't been damaged in transit or that whoever sent it is still there to receive a response. Java implements datagrams on top of the UDP protocol by using two classes:
The **DatagramPacket** object is the data container, while the **DatagramSocket** is the mechanism used to send or receive the **DatagramPacket**s.
DatagramPacket: DatagramPacket defines several constructors. Four are described here. The first constructor specifies a buffer that will receive data, and the size of a packet. It is used for receiving data over a **DatagramSocket**. The second form allows you to specify an offset into the buffer at which data will be stored. The third form specifies a target address and port, which are used by a **DatagramSocket** to determine where the data in the packet will be sent. The fourth form transmits packets beginning at the specified offset into the data. Think of the first two forms as building an "in box," and the second two forms as stuffing and addressing an envelope. Here are the four constructors:
DatagramPacket(byte *data*[ ], int *size*)
DatagramPacket(byte *data*[ ], int *offset*, int *size*)
DatagramPacket(byte *data*[ ], int *size*, InetAddress *ipAddress*, int *port*)
DatagramPacket(byte *data*[ ], int *offset*, int *size*, InetAddress *ipAddress*, int *port*)
There are several methods for accessing the internal state of a **DatagramPacket**. They give complete access to the destination address and port number of a packet, as well as the raw data and its length. Here are some of the most commonly used:

InetAddress getAddress( ) Returns the destination **InetAddress**, typically used for sending.
int getPort( ) Returns the port number.
byte[ ] getData( ) Returns the byte array of data contained in the datagram. Mostly used to retrieve data from the datagram after it has been received.
int getLength( ) Returns the length of the valid data contained in the byte array that would be returned from the **getData( )** method. This typically does not equal the length of the whole byte array.

```
import java.net.*;
class WriteServer {
public static int serverPort = 998;public static int clientPort = 999;
public static int buffer_size = 1024; public static DatagramSocket ds;
public static byte buffer[] = new byte[buffer_size];
public static void TheServer() throws Exception {
int pos=0;
while (true) {
int c = System.in.read();
switch (c) {
case -1:
System.out.println("Server Quits.");return;
case '\r':
break;
case '\n':
ds.send(new DatagramPacket(buffer,pos,
InetAddress.getLocalHost(),clientPort));
pos=0;
break;
default:
buffer[pos++] = (byte) c;
}}}
public static void TheClient() throws Exception {
while(true) {
DatagramPacket p = new DatagramPacket(buffer, buffer.length);
ds.receive(p);
System.out.println(new String(p.getData(), 0, p.getLength()));
}}
public static void main(String args[]) throws Exception {
if(args.length == 1) {
ds = new DatagramSocket(serverPort);
TheServer();
} else {
ds = new DatagramSocket(clientPort);
TheClient();}}}
```

11.Write a client application that executes an infinite loop to perform the following : Prompts the user for a number (ii) Sends it to :the server (iii) Receives a number sent by the server (iv) Displays the received number Also write a server that executes an infinite loop to read a number from a client, compute the square value and send the result to the client. (DEC2014)

PROGRAM for FTPSERVER

```
import java.io.*;
import java.net.*;
class Server
{
 public static void main(String args[]) throws IOException
 {
  ServerSocket ss = new ServerSocket(8010);
  Socket s = ss.accept();
  BufferedReader br = new BufferedReader(new InputStreamReader(s.getInputStream()));
  PrintWriter p = new PrintWriter(s.getOutputStream(), true);
  DataInputStream in = new DataInputStream(System.in);
  boolean bo=true;
  while(bo)
  {
   int i=Integer.parseInt(br.readLine());
   String st;
   switch(i)
   {
    case 1:
          {
           st=br.readLine();
           System.out.println("Input File name from client is:" +st);
           DataInputStream inf = new DataInputStream(new FileInputStream(new File(st)));
           st=br.readLine();
           System.out.println("Output File name from client is:" +st);
              DataOutputStream outf = new DataOutputStream(new FileOutputStream(new
File(st)));
           st=inf.readLine();
           while(st!=null)
           {
             outf.writeBytes(st);
             st=inf.readLine();
           }
           System.out.println("File content has been sent to client");
           break;
          }
    case 2:
          {
                System.out.println("Enter the Input File name from server to client");
           st=in.readLine();
           p.println(st);
                System.out.println("Enter the Output File name from server to client");
           st=in.readLine();
           p.println(st);
          }
    case 3:
          {
           bo=false; break;
          }
```

```
      }
    }} }
```

PROGRAM for FTPCLIENT

```
import java.io.*;
import java.net.*;
class Client
{
  public static void main(String args[]) throws IOException
  {
    Socket s = new Socket("localhost", 8010);
    BufferedReader br = new BufferedReader(new InputStreamReader(s.getInputStream()));
    PrintWriter p = new PrintWriter(s.getOutputStream(), true);
    DataInputStream in = new DataInputStream(System.in);
    boolean bo=true;
    while(bo)
    {
System.out.println("Enter the choice\n 1. Getting the file content\n 2.Sending the file content\n 3.Exit");
      int i=Integer.parseInt(in.readLine());
      p.println(i);
      String st;
      switch(i)
      {
        case 1:
              {

              System.out.println("Enter the Input File name from client to server");
              st=in.readLine(); p.println(st);

              System.out.println("Enter the Output File name from client to server");
              st=in.readLine(); p.println(st);   break;
             }
       case 2:
              {
                    st=br.readLine();
                    System.out.println("Input File name from server is:" +st);
             DataInputStream inf = new DataInputStream(new FileInputStream(new File(st)));
              st=br.readLine();
              System.out.println("Output File name from server is:" +st);
              DataOutputStream outf = new DataOutputStream(new FileOutputStream(new File(st)));
              st=inf.readLine();
              while(st!=null)
              {
                outf.writeBytes(st);  st=inf.readLine();
              }
              System.out.println("File content has been sent to server");
              break;
             }
       case 3:
```

```
            {
              bo=false; break;
            }
      }
    }
 }
}
```

12.Write about Java Bean builder tool with neat example.

Building Your Application

The Bean Builder can also be used for building custom UI applications which are to be integrated and invoked from the Web NMS client. This is easily achievable by making your applications implement either the NmsPanel Interface or the NmsFrame Interface which form the building blocks of the Web NMS client. For more details on building NmsPanels and NmsFrames using Bean Builder, please refer Bean Builder documentation.

Packaging your Application

Having built the application, our next responsibility is packaging the same so to be invoked from the Web NMS client. This process is made easy in Bean Builder by using the **NAR** mechanism . (**NAR** stands for **Nms ARc**hive). The NAR proprietary archiving mechanism used for packaging Web NMS applications. It is as same as a **J**ava **AR**chive with some additional Web NMS-specific information. A separate wizard named as the "**Package Wizard**" (provide link to package wizard document in Bean Builder) will help you in generating a NAR file for your application. This tool can be invoked from the following menu item in Bean Builder : "**Project -> Package For WebNMS -> Device Specific/Not Device Specific**".

Integrating/Installing Your Application

After packaging your application using the package wizard, a NAR file is generated for your application. This NAR file can be integrated into the Web NMS client using a tool named "**Deployment Wizard**" which can be invoked from the "*<Web NMS Home>/bin/developertools*" directory. The way in which your application has to be invoked from the client (i.e., whether it has to be invoked as a Frame on a menu action or from the Tree) can be configured using this tool based your application build type (whether an NmsPanel or an NmsFrame). For more details on installing a NAR file using the Deployment Wizard. You can refer the document: Deployment tool-->Installing a Client NAR.

Implementation Example

Some of the applications/tools bundled in Web NMS client has been built using Bean Builder and packaged using the NAR mechanism. The bean builder projects as a whole (Screens, Source files, etc.) are also bundled with the product and can be accessed from under the *<Web NMS Home>/projects* directory. The tools such as "**Runtime Administration**" and "**Security Administration"** which can be accessed from the "Tools" menu of the Java client and "**Batch Configuration"** displayed as a node in the client tree are some project examples built using the Bean Builder tool. One among these builder project is taken as an example and

explained for your reference in the following chapter: "Run-Time Administration: A Bean Builder Project for Web NMS".

13. Write about Java Bean builder tool with neat example and bean developer Kit (DEC 2015)

"A Javabean is a *reusable software component* that can be manipulated visually in an application builder tool."
"A Javabean is an independent, reusable software component. Beans may be visual object, like Swing components (e.g. `JButton`, `JTextField`) that you can drag and drop using a GUI builder tool to assemble your GUI application. Beans may also be invisible object, like queues or stacks. Again, you can use these components to assemble your application using a builder tool."

avabeans expose their features (such as properties, methods, events) to the application builder tools for visual manipulation. These feature names must adhere to a strict naming convention in order for them to be examined automatically. In other words, an application builder tool relies on these naming conventions to discover the exposed features, in a process known as introspection. For examples,

1. A property called propertyName of type PropertyType has the following convention:

```
PropertyType propertyName                // declaration
public PropertyType getPropertyName()    // getter
public void setPropertyName(PropertyType p)  // setter
```

Bean Development Kit (BDK)

Bean Development Kit (BDK) is a tool for testing whether your Javabeans meets the JavaBean specification. Follow the instruction provided to install the BDK. Read the documentation and tutorial provided (in particular, "The Java Tutorial, specialized trial on JavaBeans"). BDK comes with a set of sample demo beans. You should try them out and closely study these demo beans before writing our own beans.

Let's try to assemble (or compose) an application using the BDK demo beans.

1. Start the "beanbox" by running "$bdk\beanbox\run.bat".
2. From the "Toolbox" window, select "Juggler" (a demo bean) and place it inside the "beanbox" (by clicking the desired location in the "beanbox" window). Observe the "Property" window of the Juggler bean.
3. Create a button by selecting "OurButton" demo bean from the "Toolbox" and place it inside the "Beanbox". In the "Proprety" window, change the "label" from "press" to "start".
4. Focus on "OurButton", choose "Edit”" from menu ⇒ "Events" ⇒ "mouse" ⇒ "mouseClicked" and place it onto the "Juggler" (i.e., "Juggler" is the target of this event). In the "EventTargetDialog", select method "startJuggling" as the event handler.
5. Create another button by selecting "OurButton" bean from "Toolbox" and place it inside the "Beanbox" again. In the "Proprety" window, change the "label" from "press" to "stop".

6. Focus on the stop button, choose "Edit" from menu ⇒ "Events" ⇒ "mouse" ⇒ "mouseClicked" and place it onto the "Juggler". In the "EventTargetDialog", select method "stopJuggling" as the event handler.
7. Click on the buttons, and observe the result.

```
1. package elect;
2. import java.awt.*;
3. import java.io.Serializable;
4.
5. public class LightBulb extends Canvas implements Serializable {
6.
7.    public LightBulb() {     // constructor
8.       setSize(50,50);
9.       setBackground(Color.GRAY);
10.   }
11.
12.   // Properties
13.   private static final Color COLOR_OFF = Color.BLACK;
14.   private Color color = Color.ORANGE;       // property with a default value
15.   public Color getColor() { return color; }  // getter
16.   public void setColor(Color color) { this.color = color; } // setter
17.
18.   boolean on = false;                    // property with a default value
19.   public boolean isOn() { return on; }       // getter for boolean
20.   public void setOn(boolean on) { this.on = on; } // setter
21.
22.   // Override the paint() method to draw the LightBulb
23.   public void paint(Graphics g) {
24.      if (on) g.setColor(color);
25.      else g.setColor(COLOR_OFF);
26.      g.fillOval(10, 10, 30, 30);
27.   }
28.
29.   public void switchOn() {   // switch on the Light
30.      on = true;
31.      repaint();
32.   }
33.
34.   public void switchOff() {  // switch off the Light
35.      on = false;
36.      repaint();
37.   }
38.
39.   public void toggle() {   // If on turns off; else turns on
40.      on = !on;
41.      repaint();
42.   }
```

```
43. }
```

14. Discuss about Classes and Interface used by Java Bean (DEC 2016)

The Java Beans API: The Java Beans functionality is provided by a set of classes and interfaces in the java.beans package. Set of classes and interfaces in the java.beans package Package java.beans Contains classes related to developing beans -- components based on the JavaBeansTM architecture

BeanDescriptor: A BeanDescriptor provides global information about a "bean", including its Java class, its displayName, etc Beans This class provides some general purpose beans control methods.

DefaultPersistenceDelegate: The DefaultPersistenceDelegate is a concrete implementation of the abstract PersistenceDelegate class and is the delegate used by default for classes about which no information is available.

Encoder: An Encoder is a class which can be used to create files or streams that encode the state of a collection of JavaBeans in terms of their public APIs.

EventHandler :The EventHandler class provides support for dynamically generating event listeners whose methods execute a simple statement involving an incoming event object and a target object. **EventSetDescriptor:** An EventSetDescriptor describes a group of events that a given Java bean fires. Expression An Expression object represents a primitive expression in which a single method is applied to a target and a set of arguments to return a result - as in "a.getFoo()". AppletInitializer This interface is designed to work in collusion with java.beans.Beans.instantiate.

BeanInfo A bean implementor who wishes to provide explicit information about their bean may provide a BeanInfo class that implements this BeanInfo interface and provides explicit information about the methods, properties, events, etc, of their bean.

Customizer A customizer class provides a complete custom GUI for customizing a target Java Bean DesignMode This interface is intended to be implemented by, or delegated from, instances of java.beans.beancontext.BeanContext, in order to propagate to its nested hierarchy of java.beans.beancontext.BeanContextChild instances, the current "designTime" property. **ExceptionListener**: An ExceptionListener is notified of internal exceptions.

PropertyChangeListener: A "PropertyChange" event gets fired whenever a bean changes a "bound" property.

PropertyEditor A PropertyEditor class provides support for GUIs that want to allow users to edit a property value of a given type.

VetoableChangeListener: A VetoableChange event gets fired whenever a bean changes a "constrained" property Visibility Under some circumstances a bean

The BeanInfoInterface : By default an Introspector uses the Reflection API to determine the features of a JavaBean.However, a JavaBean can provide its own BeanInfo which will be used instead by the Introspector to determine the discussed information. This allows a developer hiding specific properties, events and methods from a builder tool or from any other tool which uses the Introspector class. Moreover it allows supplying further details about events/properties/methods as you are in charge of creating the descriptor objects. Hence you can, for example, call the setShortDescription() method to set a descriptive description.

A BeanInfo class has to be derived from the SimpleBeanInfo class and its name has to start with the name of the associated JavaBean. At this point it has to be underlined that the name of the BeanInfo class is the only relation between a JavaBean and its BeanInfo class. The BeanInfo interface provides the methods that enable you to specify and retrieve the information about a JavaBean.

15. i)Methods Supported by Datagramsocket(DEC 2016)

ad.getHostName();
ad.getHostAddress();
Handles Internet addresses both as host names and as IP addresses. Static Method getByName returns the IP address of a specified host name as an InetAddress object. Methods for address/name conversion:
public static InetAddress getByName(String host) throws UnknownHostException
public static InetAddress[] getAllByName(String host) throws UnknownHostException
public static InetAddress getLocalHost() throws UnknownHostException
public boolean isMulticastAddress()
public String getHostName()
public byte[] getAddress()
public String getHostAddress()
public int hashCode()
public boolean equals(Object obj)
public String toString()
The UDP classes : 2 classes:
java.net.DatagramSocket class: is a connection to a port that does the sending and receiving. A DatagramSocket can send to multiple, different addresses.The address to which data goes is stored in the packet, not in the socket.
public DatagramSocket() throws SocketException public DatagramSocket(int port) throws SocketException
public DatagramSocket(int port, InetAddress laddr) throws SocketException
java.net.DatagramPacket class : is a wrapper for an array of bytes from which data will be sent or into which data will be received. It also contains the address and port to which the packet will be sent.
public DatagramPacket(byte[] data, int length) public DatagramPacket(byte[] data, int length, InetAddress host, int port)
SERVER:
1. Create a DatagramSocket object DatagramSocket dgramSocket = new DatagramSocket(1234);
2. Create a buffer for incoming datagrams byte[] buffer = new byte[256];
3. Create a DatagramPacket object for the incoming datagram DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);
4. Accept an incoming datagram dgramSocket.receive(inPacket)
5. Accept the sender's address and port from the packet InetAddress clientAddress = inPacket.getAddress(); int clientPort = inPacket.getPort();
6. Retrieve the data from the buffer string message = new String(inPacket.getData(), 0, inPacket.getLength());
7. Create the respons e datagram DatagramPacket outPacket = new DatagramPacket(response.getBytes(), response.length(), clientAddress, clientPort);
8. Send the response datagram dgramSocket.send(outPacket)
9. Close the DatagramSocket: dgram.close();
CLIENT:
1. Create a DatagramSocket object DatagramSocket dgramSocket = new DatagramSocket;
2. Create the outgoing datagram DatagramPacket outPacket = new DatagramPacket(message.getBytes(), message.length(), host, port);
3. Send the datagram message dgramSocket.send(outPacket)
4. Create a buffer for incoming datagrams byte[] buffer = new byte[256];

5. Create a DatagramPacket object for the incoming datagram DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);
6. Accept an incoming datagram dgramSocket.receive(inPacket)
7. Retrieve the data from the buffer string response = new String(inPacket.getData(), 0, inPacket.getLength());
8. Close the DatagramSocket: dgram.close();

When you receive a packet, the IP and port number of the sender are set in the DatagramPacket use the same packet to reply, by overwriting the data, using the method: packet.setData(newbuffer);

Non-blocking I/O receiving UDP packets: set a time-out in milliseconds to determine how long a read operation blocks, before throwing an exception.
…socket.setSoTimeout(duration);

15. ii) Explain the Fundamentals of Java Bean Ad Bdk(DEC 2016)

INTRODUCTION TO JAVA BEANS

Software components are self-contained software units developed according to the motto "Developed them once, run and reused them everywhere". Or in other words, reusability is the main concern behind the component model. A software component is a reusable object that can be plugged into any target software application. You can develop software components using various programming languages, such as C, C++, Java, and Visual Basic.

- A "Bean" is a reusable software component model based on sun's java bean specification that can be manipulated visually in a builder tool.
- The term software component model describe how to create and use reusable software components to build an application
- Builder tool is nothing but an application development tool which lets you both to create new beans or use existing beans to create an application.
- To enrich the software systems by adopting component technology JAVA came up with the concept called Java Beans.
- Java provides the facility of creating some user defined components by means of Bean programming.
- We create simple components using java beans.
- We can directly embed these beans into the software. Advantages of Java Beans:
- The java beans possess the property of "Write once and run anywhere".
- Beans can work in different local platforms.
- Beans have the capability of capturing the events sent by other objects and vice versa enabling object communication.
- The properties, events and methods of the bean can be controlled by the application developer.(ex. Add new properties)
- Beans can be configured with the help of auxiliary software during design time.(no hassle at runtime)
- The configuration setting can be made persistent.(reused)
- Configuration setting of a bean can be saved in persistent storage and restored later.

What can we do/create by using JavaBean: There is no restriction on the capability of a Bean.• It may perform a simple function, such as checking the spelling of a document, or a complex function, such as forecasting the performance of a stock portfolio. A Bean may be visible to an end user. One example of this is a button on a graphical user interface. • Software to generate a pie chart from a set of data points is an example of a Bean that can

execute locally. • Bean that provides real-time price information from a stock or commodities exchange.

JavaBeans Basic rules: A JavaBean should: ¬ be public ¬ implement the Serializable interface ¬ have a no-arg constructor ¬ be derived from javax.swing.JComponent or java.awt.Component if it is visual The classes and interfaces defined in the java.beans package enable you to create JavaBeans.

The Java Bean components can exist in one of the following three phases of development

- Construction phase
- Build phase
- Execution phase

It supports the standard component architecture features of • Properties • Events • Methods • Persistence.

In addition Java Beans provides support for • Introspection (Allows Automatic Analysis of a java beans) • Customization (To make it easy to configure a java beans component)

Elements of a JavaBean:

- Properties Similar to instance variables. A bean property is a named attribute of a bean that can affect its behavior or appearance. Examples of bean properties include color, label, font, font size, and display size.

•**Methods**: Same as normal Java methods. Every property should have accessor (get) and mutator (set) method. All Public methods can be identified by the introspection mechanism. There is no specific naming standard for these methods

- Events Similar to Swing/AWT event handling.

The Java Bean Component Specification: Customization: Is the ability of JavaBean to allow its properties to be changed in build and execution phase. Persistence:-Is the ability of JavaBean to save its state to disk or storage device and restore the saved state when the JavaBean is reloaded Communication:- Is the ability of JavaBean to notify change in its properties to other JavaBeans or the container. Introspection:-Is the ability of a JavaBean to allow an external application to query the properties, methods, and events supported by it.

Beans Development Kit Is a development environment to create, configure, and test JavaBeans. The features of BDK environment are:

- Provides a GUI to create, configure, and test JavaBeans.
- Enables you to modify JavaBean properties and link multiple JavaBeans in an application using BDK.
- Provides a set of sample JavaBeans.
- Enables you to associate pre-defined events with sample JavaBeans. Identifying BDK Components
- Execute the run.bat file of BDK to start the BDK development environment.

The components of BDK development environment are:

1. ToolBox
2. BeanBox(Is a workspace for creating the layout of JavaBean application.)
3. Properties (Displays all the exposed properties of a JavaBean. You can modify JavaBean properties in the properties window. The following figure shows the Properties window)
4. Method Tracer(Displays the debugging messages and method calls for a JavaBean application.)

Steps to Develop a User-Defined JavaBean:

1. Create a directory for the new bean
2. Create the java bean source file(s)
3. Compile the source file(s)

4. Create a manifest file
5. Generate a JAR file
6. Start BDK
7. Load Jar file
8. Test.

16. Explain TCP/IP Client and the Various Classes and Methods Supported By Java for TCP/IP Client Sockets (DEC 2016)

A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet. There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients. The ServerSocket class is designed to be a "listener," which waits for clients to connect before doing anything. The Socket class is designed to connect to server sockets and initiate protocol exchanges.

TCP/IP Client Sockets Here are two constructors used to create client sockets: Socket(String hostName, int port) Creates a socket connecting the localhost to the named host and port; can throw an UnknownHostException oran IOException. Socket(InetAddress ipAddress, int port) Creates a socket using a preexisting InetAddress object and a port; can throw an IOException.

A socket can be examined at any time for the address and port information associated with it, by use of the following methods: InetAddress getInetAddress( ) Returns the InetAddress associated with the Socket object. int getPort( ) Returns the remote port to which this Socket object is connected. int getLocalPort( ) Returns the local port to which this Socket object is connected

Once the Socket object has been created, it can also be examined to gain access to the input and output streams associated with it.

InputStream getInputStream( )

Returns the InputStream associated with the invoking socket.

OutputStream getOutputStream( )

Returns the OutputStream associated with the invoking socket.

The very simple example that follows opens a connection to a whois port on the InterNIC server, sends the command -line argument down the socket, and then prints the data that is returned. InterNIC will try to lsookup the argument as a registered Internet domain name, then send back the IP address and contact information for that site.

```
import java.net.*;
import java.io.*;
class Whois
 {
public static void main(String args[]) throws Exception
 {
int c; Socket s = new Socket("internic.net", 43);
InputStream in = s.getInputStream(); OutputStream out = s.getOutputStream();
Stringstr=(args.length==0? "osborne.com":args[0])+"n";
byte buf[] = str.getBytes(); out.write(buf);
while ((c = in.read()) != -1) System.out.print((char) c); s.close();
 }
 }
```

URL (Uniform Resource Locator)

The URL (Uniform Resource Locator) provides a reasonably intelligible form to uniquely identify or address information on the Internet. URLs are ubiquitous; every browser uses them to identify information on the Web. In fact, the Web is really just that same old

Internet with all of its resources addressed as URLs plus HTML. Within Java's network class library, the URL class provides a simple, concise API to access information across the Internet using URLs. Two examples of URLs are

http://www.osborne.com/ and http://www.osborne.com:80/ index.htm.

A URL specification is based on four components:

The first is the protocol to use, separated from the rest of the locator by a colon (:). Common protocols are http, ftp, gopher, and file, although these days almost everything is being done via HTTP (in fact, most browsers will proceed correctly if you leave off the "http://" from your URL specification).

The second component is the host name or IP address of the host to use; this is delimited on the left by double slashes (//) and on the right by a slash (/) or optionally a colon (:).

The third component, the port number, is an optional parameter, delimited on the left from the host name by a colon (:) and on the right by a slash (/). (It defaults to port 80, the predefined HTTP port; thus ":80" is redundant.)

The fourth part is the actual file path. Most HTTP servers will append a file named index.html or index.htm to URLs that refer directly to a directory resource.

TCP/IP Server Sockets

The ServerSocket class is used to create servers that listen for either local or remote client programs to connect to them on published ports. Since the Web is driving most of the activity on the Internet, this section develops an operational web (http) server. ServerSockets are quite different from normal Sockets. When you create a ServerSocket, it will register itself with the system as having an interest in client connections.

The constructors for ServerSocket reflect the port number that you wish to accept connections on and, optionally, how long you want the queue for said port to be. The queue length tells the system how many client connections it can leave pending before it should simply refuse connections. The default is 50. The constructors might throw an IOException under adverse conditions. Here are the constructors:

ServerSocket has a method called accept( ), which is a blocking call that will wait for a client to initiate communications, and then return with a normal Socket that is then used for communication with the client.

Serve- Client Program in Java

Server Side Program (Server.java)

```
import java.io.*;
import java.net.*;
public class Server
 {
private static Socket socket; public static void main(String[] args)
{
try
{
int port = 25000;
  ServerSocket serverSocket = new ServerSocket(port);
System.out.println("Server Started and listening to the port 25000");
while(true)
    {
socket = serverSocket.accept();
   InputStream is = socket.getInputStream();
 InputStreamReader isr = new InputStreamReader(is);
  BufferedReader br = new BufferedReader(isr);
   String number = br.readLine();
```

```
System.out.println("Message received from client is "+number);
  String returnMessage="You are Welcome....Mr. Client";
 OutputStream os = socket.getOutputStream();
OutputStreamWriter osw = new OutputStreamWriter(os);
 BufferedWriter bw = new BufferedWriter(osw);
bw.write(returnMessage);
System.out.println("Message sent to the client is "+returnMessage);
bw.flush();
}
  }
catch (Exception e) { e.printStackTrace();
}
 Finally
  {
try
{
socket.close();
 }
catch(Exception e){} } } }
```

Client Side Program (Client.java)

```
import java.io.*;
import java.net.*;
public class Client
{
private static Socket socket; public static void main(String args[])
{
try
 {
 String host = "localhost";
int port = 25000;
 InetAddress address = InetAddress.getByName(host);
socket = new Socket(address, port);
  OutputStream os = socket.getOutputStream();
 OutputStreamWriter osw = new OutputStreamWriter(os);
 BufferedWriter bw = new BufferedWriter(osw);
String number = "Thank You server. i am Connected !!!!!";
String sendMessage = number + "\n";            bw.write(sendMessage);
bw.flush();
System.out.println("Message sent to the server : "+sendMessage);
 InputStream is = socket.getInputStream();
  InputStreamReader isr = new InputStreamReader(is);
BufferedReader br = new BufferedReader(isr);
String message = br.readLine();
System.out.println("Message received from the server : " +message);
 }
catch (Exception exception)
{
exception.printStackTrace();
}
finally
```

```
  {
try
      {
socket.close();
    }
catch(Exception e)
        {
e.printStackTrace();    } } } }
```