Unit 1

1. Inheritance

Inheritance is an important pillar of OOP (Object Oriented Programming). It is the mechanism in java by which one class is allowed to inherit the features (fields and methods) of another class.

**Important terminology:**
- **Super Class:** The class whose features are inherited is known as super class (or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as sub class (or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of “reusability”, i.e., when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

2. Method binding

**Static Binding and Dynamic Binding**

Connecting a method call to the method body is known as binding.

There are two types of binding

1. static binding (also known as early binding).
2. dynamic binding (also known as late binding).

**Understanding Type**

Let's understand the type of instance.

1) **variables have a type**

Each variable has a type; it may be primitive and non-primitive.

1. int data = 30;

   Here data variable is a type of int.

2) **References have a type**

1. class Dog{
3) Objects have a type

An object is an instance of particular java class, but it is also an instance of its superclass.

```java
class Animal{
}

class Dog extends Animal{
    public static void main(String args[]){
        Dog d1=new Dog();
    }
}
```

Here d1 is an instance of Dog class, but it is also an instance of Animal.

---

**static binding**

When type of the object is determined at compiled time (by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

**Example of static binding**

```java
class Dog{
    private void eat(){System.out.println("dog is eating...");}
    public static void main(String args[]){
        Dog d1=new Dog();
        d1.eat();
    }
}
```

---

**Dynamic binding**

When type of the object is determined at run-time, it is known as dynamic binding.
Example of dynamic binding

```java
1. class Animal {
2.   void eat() { System.out.println("animal is eating..."); }
3. }
4.
5. class Dog extends Animal {
6.   void eat() { System.out.println("dog is eating..."); }
7.
8.   public static void main(String args[]) {
9.     Animal a = new Dog();
10.    a.eat();
11. }
12. }
```

Output: dog is eating...

In the above example object type cannot be determined by the compiler, because the instance of Dog is also an instance of Animal. So compiler doesn't know its type, only its base type.

3. Java buzzwords

Simple

- Java was designed to be easy for professional programmer to learn and use effectively.
- It's simple and easy to learn if you already know the basic concepts of Object Oriented Programming.
- C++ programmer can move to JAVA with very little effort to learn.
- In Java, there is small number of clearly defined ways to accomplish a given task.

Object Oriented

- Java is true object oriented language.
- Almost “Everything is an Object” paradigm. All program code and data reside within objects and classes.
- The object model in Java is simple and easy to extend.
- Java comes with an extensive set of classes, arranged in packages that can be used in our programs through inheritance.

Distributed
- Java is designed for distributed environment of the Internet. It's used for creating applications on networks.
- Java applications can access remote objects on Internet as easily as they can do in local system.
- Java enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

4. Data types,

5. String handling
In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

1. `char[] ch={'j','a','v','a','t','p','o','i','n','t'};`
2. `String s=new String(ch);`

is same as:

1. `String s="javatpoint";`

**Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The **java.lang.String** class implements **Serializable**, **Comparable** and **CharSequence** interfaces.

---

Unit II

1. Defining a Package,

- A **Java package** is a group of similar types of classes, interfaces and sub-packages.
• Package in java can be categorized in two form, built-in package and user-defined package.
• There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
• Here, we will have the detailed learning of creating and using user-defined packages.
• Advantage of Java Package
  • 1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
  • 2) Java package provides access protection.
  • 3) Java package removes naming collision.

2. importing packages

Package are used in Java, in-order to avoid name conflicts and to control access of class, interface and enumeration etc. A package can be defined as a group of similar types of classes, interface, enumeration or sub-package. Using package it becomes easier to locate the related classes and it also provides a good structure for projects with hundreds of classes and other files.

**Types of Packages: Built-in and User defined**
- **Built-in Package**: Existing Java package for example `java.lang`, `java.util` etc.
- **User-defined-package**: Java package created by user to categorize their project's classes and interface.

### Creating a package

Creating a package in java is quite easy. Simply include a package command followed by name of the package as the first statement in java source file.

```java
package mypack;
public class employee {
    statement;
}
```

The above statement will create a package with name `mypack` in the project directory.

Java uses file system directories to store packages. For example the `.java` file for any class you define to be part of `mypack` package must be stored in a directory called `mypack`.

### Additional points about package:

- A package is always defined as a separate folder having the same name as the package name.
- Store all the classes in that package folder.
- All classes of the package which we wish to access outside the package must be declared public.
- All classes within the package must have the package statement as its first line.
- All classes of the package must be compiled before use (So that they are error free)

### Example of Java packages

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

How to compile Java programs inside packages?

This is just like compiling a normal java program. If you are not using any IDE, you need to follow the steps given below to successfully compile your packages:

javac -d directory javafilename

Example:

javac -d . FirstProgram.java

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

How to run Java package program?

You need to use fully qualified name e.g. learnjava.FirstProgram etc to run the class.

To Compile:

javac -d . FirstProgram.java

To Run:

java learnjava.FirstProgram

Output: Welcome to package

import keyword

import keyword is used to import built-in and user-defined packages into your java source file so that your class can refer to a class that is in another package by directly using its name.

There are 3 different ways to refer to any class that is present in a different package:
1. **Using fully qualified name (But this is not a good practice.)**

If you use fully qualified name to import any class into your program, then only that particular class of the package will be accessible in your program, other classes in the same package will not be accessible. For this approach, there is no need to use the `import` statement. But you will have to use the fully qualified name every time you are accessing the class or the interface, which can look a little untidy if the package name is long.

This is generally used when two packages have classes with same names. For example: `java.util` and `java.sql` packages contain `Date` class.

**Example:**

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

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

**Output:**

Hello

2. **To import only the class/classes you want to use**
If you import `packagename.classname` then only the class with name `classname` in the package with name `packagename` will be available for use.

**Example:**

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

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

**Output:**

Hello

3. **To import all the classes from a particular package**

If you use `packagename.*`, then all the classes and interfaces of this package will be accessible but the classes and interface inside the subpackages will not be available for use.

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

**Example:**
3. implementing interfaces

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways –
An interface can contain any number of methods.

An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.

The byte code of an interface appears in a .class file.

Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including –

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaring Interfaces

The interface keyword is used to declare an interface. Here is a simple example to declare an interface –

Example

Following is an example of an interface –

```java
/* File name : NameOfInterface.java */

import java.lang.*;

// Any number of import statements

public interface NameOfInterface { 

    // Any number of final, static fields

    // Any number of abstract method declarations

}
```

Interfaces have the following properties –
An interface is implicitly abstract. You do not need to use the `abstract` keyword while declaring an interface.

Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.

Methods in an interface are implicitly public.

Example

```java
/* File name : Animal.java */

interface Animal {
    public void eat();
    public void travel();
}
```

Implementing Interfaces

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the `implements` keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example

```java
/* File name : MammalInt.java */

public class MammalInt implements Animal {

    public void eat() {
        System.out.println("Mammal eats");
    }

    public void travel() {
        System.out.println("Mammal travels");
    }
}
```
```java
public int noOg1egs() {
    return 0;
}

public static void main(String args[]) {
    MammalInt m = new MammalInt();
    m.eat();
    m.travel();
}
```

This will produce the following result –

Output

Mammal eats
Mammal travels

When overriding methods defined in interfaces, there are several rules to be followed –

- Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
- The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
- An implementation class itself can be abstract and if so, interface methods need not be implemented.

When implementation interfaces, there are several rules –

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, in a similar way as a class can extend another class.
Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The `extends` keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.

Example

```java
// Filename: Sports.java
public interface Sports {
    public void setHomeTeam(String name);
    public void setVisitingTeam(String name);
}

// Filename: Football.java
public interface Football extends Sports {
    public void homeTeamScored(int points);
    public void visitingTeamScored(int points);
    public void endOfQuarter(int quarter);
}

// Filename: Hockey.java
public interface Hockey extends Sports {
    public void homeGoalScored();
    public void visitingGoalScored();
    public void endOfPeriod(int period);
    public void overtimePeriod(int ot);
}
```
The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

Extending Multiple Interfaces

A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.

The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

For example, if the Hockey interface extended both Sports and Event, it would be declared as:

```java
public interface Hockey extends Sports, Event
```

Tagging Interfaces

The most common use of extending interfaces occurs when the parent interface does not contain any methods. For example, the MouseListener interface in the java.awt.event package extended java.util.EventListner, which is defined as:

```java
package java.util;

public interface EventListner
{
}
```

An interface with no methods in it is referred to as a tagging interface. There are two basic design purposes of tagging interfaces:

- **Creates a common parent** – As with the EventListner interface, which is extended by dozens of other interfaces in the Java API, you can use a tagging interface to create a common parent among a group of interfaces. For example, when an interface extends EventListner, the JVM knows that this particular interface is going to be used in an event delegation scenario.

4. Stream classes

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

**Stream**

A stream can be defined as a sequence of data. There are two kinds of Streams –

- **InPutStream** – The InputStream is used to read data from a source
- **OutPutStream** – The OutputStream is used for writing data to a destination

5. **Serialization**

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

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

The reverse operation of serialization is called **deserialization**.

**Advantage of Java Serialization**

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

---

**Unit III**

1. Fundamentals of exception handling

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

In this page, we will learn about java exception, its type and the difference between checked and unchecked exceptions.

---

**What is exception**

**Dictionary Meaning:** Exception is an abnormal condition.

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

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

Advantage of Exception Handling

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

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

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

2. Exception types
3. throw, throws and finally

**throw Keyword**

The `throw` keyword is used to throw an exception explicitly. Only objects of Throwable class or its subclasses can be thrown. Program execution stops on encountering `throw` statement, and the closest catch statement is checked for matching type of exception.

**Syntax:**

```java
throw ThrowableInstance
```

**Creating Instance of Throwable class**

There are two possible ways to create an instance of class Throwable,

1. Using a parameter in catch block.
2. Creating instance with `new` operator.

```java
new NullPointerException("test");
```

This constructs an instance of NullPointerException with name test.

**Example demonstrating throw Keyword**

```java
class Test
```
In the above example the avg() method throw an instance of ArithmeticException, which is successfully handled using the catch statement and thus, the program outputs "Exception caught".

**throws Keyword**

Any method that is capable of causing exceptions must list all the exceptions possible during its execution, so that anyone calling that method gets a prior knowledge about which exceptions are to be handled. A method can do so by using the `throws` keyword.

**Syntax :**
```java
type method_name(parameter_list) throws exception_list
{
    //definition of method
}
```
Example demonstrating throws Keyword

class Test
{
    static void check() throws ArithmeticException
    {
        System.out.println("Inside check function");
        throw new ArithmeticException("demo");
    }

    public static void main(String args[])
    {
        try
        {
            check();
        }
        catch(ArithmeticException e)
        {
            System.out.println("caught" + e);
        }
    }
}

Inside check function
caughtjava.lang.ArithmeticException: demo

Difference between throw and throws

<table>
<thead>
<tr>
<th>throw</th>
<th>throws</th>
</tr>
</thead>
<tbody>
<tr>
<td>throw keyword is used to throw an exception explicitly.</td>
<td>throws keyword is used to declare an exception possible during its execution.</td>
</tr>
<tr>
<td>throw keyword is followed by an instance of</td>
<td>throws keyword is followed by one or more</td>
</tr>
</tbody>
</table>
### finally clause

A finally keyword is used to create a block of code that follows a try block. A finally block of code is always executed whether an exception has occurred or not. Using a finally block, it lets you run any cleanup type statements that you want to execute, no matter what happens in the protected code. A finally block appears at the end of catch block.

### 4. Java thread model

Java is a multi-threaded programming language which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

### Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.
Following are the stages of the life cycle –

- **New** – A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

- **Runnable** – After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

- **Waiting** – Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

- **Timed Waiting** – A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

- **Terminated (Dead)** – A runnable thread enters the terminated state when it completes its task or otherwise terminates.

**Thread Priorities**

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).
Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

Create a Thread by Implementing a Runnable Interface

If your class is intended to be executed as a thread then you can achieve this by implementing a `Runnable` interface. You will need to follow three basic steps –

**Step 1**

As a first step, you need to implement a run() method provided by a `Runnable` interface. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of the run() method –

```java
public void run()
```

**Step 2**

As a second step, you will instantiate a `Thread` object using the following constructor –

```java
Thread(Runnable threadObj, String threadName);
```

Where, `threadObj` is an instance of a class that implements the `Runnable` interface and `threadName` is the name given to the new thread.

**Step 3**

Once a Thread object is created, you can start it by calling `start()` method, which executes a call to run() method. Following is a simple syntax of `start()` method –

```java
void start();
```

5. Inter thread communication.

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

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

- `wait()`
- `notify()`
- `notifyAll()`
1) wait() method

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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public final void wait() throws InterruptedException</td>
<td>waits until object is notified.</td>
</tr>
<tr>
<td>public final void wait(long timeout) throws InterruptedException</td>
<td>waits for the specified amount of time.</td>
</tr>
</tbody>
</table>

2) notify() method

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

public final void notify()

3) notifyAll() method

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

public final void notifyAll()

Unit IV

1. Collections overview

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.
All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

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

**What is Collection in java**

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

**What is framework in java**

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

**What is Collection framework**

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

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

Java collection class is used exclusively with static methods that operate on or return collections. It inherits Object class.

The important points about Java Collections class are:

- Java Collection class supports the **polymorphic algorithms** that operate on collections.
- Java Collection class throws a **NullPointerException** if the collections or class objects provided to them are null.
Let's see the declaration for Java.util.Collections class.

**public class** Collections **extends** Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static <code>&lt;T&gt; boolean addAll(Collection&lt;? super T&gt; c, T... elements)</code></td>
<td>It is used to add all of the specified elements to the specified collection.</td>
</tr>
<tr>
<td>static <code>&lt;T&gt; Queue&lt;T&gt; asLifoQueue(Deque&lt;T&gt; deque)</code></td>
<td>It is used to return a view of a Deque as a Last-In-First-Out (LIFO) Queue.</td>
</tr>
<tr>
<td>static <code>&lt;T&gt; int binarySearch(List&lt;? extends T&gt; list, T key, Comparator&lt;? super T&lt; c)</code></td>
<td>It is used to search the specified list for the specified object using the binary search algorithm.</td>
</tr>
<tr>
<td>static <code>&lt;E&gt; List&lt;E&gt;</code> checkedList(List&lt;E&gt; list, Class&lt;E&gt; type)</td>
<td>It is used to return a dynamically typesafe view of the specified list.</td>
</tr>
<tr>
<td>static <code>&lt;E&gt; Set&lt;E&gt;</code> checkedSet(Set&lt;E&gt; s, Class&lt;E&gt; type)</td>
<td>It is used to return a dynamically typesafe view of the specified set.</td>
</tr>
<tr>
<td>static <code>&lt;E&gt;</code> SortedSet&lt;E&gt;checkedSortedSet(SortedSet &lt;E&gt; s, Class&lt;E&gt; type)</td>
<td>It is used to return a dynamically typesafe view of the specified sorted set.</td>
</tr>
<tr>
<td>static void reverse(List&lt;?&gt; list)</td>
<td>It is used to reverse the order of the elements in the specified list.</td>
</tr>
<tr>
<td>static <code>&lt;T&gt; T max(Collection&lt;? extends T&gt; coll, Comparator&lt;? super T&gt; comp)</code></td>
<td>It is used to return the maximum element of the given collection, according to the order induced by the specified comparator.</td>
</tr>
</tbody>
</table>
static <T extends Object & Comparable<? super T>> T min(Collection<? extends T> coll)

It is used to return the minimum element of the given collection, according to the natural ordering of its elements.

static boolean replaceAll(List list, T oldVal, T newVal)

It is used to replace all occurrences of one specified value in a list with another.

3. Accessing a Collection via an Iterator,

**Accessing a Collection**

To access, modify or remove any element from any collection we need to first find the element, for which we have to cycle through the elements of the collection. There are three possible ways to cycle through the elements of any collection.

1. Using Iterator interface
2. Using ListIterator interface
3. Using for-each loop

**Steps to use an Iterator**

1. Obtain an iterator to the start of the collection by calling the collection's iterator() method.
2. Set up a loop that makes a call to hasNext() method. Make the loop iterate as long as hasNext() method returns true.
3. Within the loop, obtain each element by calling next() method.

**Accessing elements using Iterator**

Iterator Interface is used to traverse a list in forward direction, enabling you to remove or modify the elements of the collection. Each collection classes provide iterator() method to return an iterator.

Methods of Iterator:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean hasNext()</td>
<td>Returns true if there are more elements in the collection. Otherwise, returns false.</td>
</tr>
</tbody>
</table>
**E next()**

Returns the next element present in the collection. Throws 
NoSuchElementException if there is not a next element.

**void remove()**

Removes the current element. Throws IllegalStateException if an attempt is made 
to call remove() method that is not preceded by a call to next() method.

---

```java
import java.util.*;
class Test_Iterator {

    public static void main(String[] args) {
        ArrayList<String> ar = new ArrayList<String>();
        ar.add("ab");
        ar.add("bc");
        ar.add("cd");
        ar.add("de");
        Iterator it = ar.iterator();  // Declaring Iterator
        while(it.hasNext()) {
            System.out.print(it.next()+" ");
        }
    }
}
```

ab bc cd de

---

**Accessing elements using ListIterator**

**ListIterator** Interface is used to traverse a list in both **forward** and **backward** direction. It is available to only those collections that implements the **List** Interface.

**Methods of ListIterator:**
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void add(E obj)</td>
<td>Inserts obj into the list in front of the element that will be returned by the next call to next() method.</td>
</tr>
<tr>
<td>boolean hasNext()</td>
<td>Returns true if there is a next element. Otherwise, returns false.</td>
</tr>
<tr>
<td>boolean hasPrevious()</td>
<td>Returns true if there is a previous element. Otherwise, returns false.</td>
</tr>
<tr>
<td>E next()</td>
<td>Returns the next element. A NoSuchElementException is thrown if there is not a next element.</td>
</tr>
<tr>
<td>int nextIndex()</td>
<td>Returns the index of the next element. If there is not a next element, returns the size of the list.</td>
</tr>
<tr>
<td>E previous()</td>
<td>Returns the previous element. A NoSuchElementException is thrown if there is not a previous element.</td>
</tr>
<tr>
<td>int previousIndex()</td>
<td>Returns the index of the previous element. If there is not a previous element, returns -1.</td>
</tr>
<tr>
<td>void remove()</td>
<td>Removes the current element from the list. An IllegalStateException is thrown if remove() method is called before next() or previous() method is invoked.</td>
</tr>
<tr>
<td>void set(E obj)</td>
<td>Assigns obj to the current element. This is the element last returned by a call to either next() or previous() method.</td>
</tr>
</tbody>
</table>

```java
import java.util.*;
class Test_Iterator {
   public static void main(String[] args)
```
Using for-each loop

for-each version of for loop can also be used for traversing the elements of a collection. But this can only be used if we don't want to modify the contents of a collection and we don't want any reverse access. for-each loop can cycle through any collection of object that implements Iterable interface.

```java
import java.util.*;
class ForEachDemo {
    public static void main(String[] args) {
        LinkedList<String> ls = new LinkedList<String>();
        ls.add("a");
    }
}
```
4. For-Each alternative

If you are working with arrays and collections, you can use alternative syntax of for loop (enhanced form of for loop) to iterate through items of arrays/collections. It is also referred as for-each loop because the loop iterates through each element of array/collection.

To learn about standard syntax of for loop, arrays and collections, visit:

- Java for Loop
- Java arrays
- Java collections

Here's an example to iterate through elements of an array using standard for loop:

```java
class ForLoop {
    public static void main(String[] args) {
        char[] vowels = {'a', 'e', 'i', 'o', 'u'};
        for (int i = 0; i < vowels.length; ++i) {
            System.out.println(vowels[i]);
        }
    }
}
```
You can perform the same task using for-each loop as follows:

```java
class AssignmentOperator {
    public static void main(String[] args) {
        char[] vowels = {'a', 'e', 'i', 'o', 'u'};
        // foreach loop
        for (char item : vowels) {
            System.out.println(item);
        }
    }
}
```

The output of both programs will be same:

```
a
e
i
o
u```

The use of enhanced for loop is easier to write and makes your code more readable. Hence, it's recommended over standard form whenever possible.

---

**Syntax of for-each loop**

Let's first look at the syntax of for each loop:

```java
for(data_type item : collection) {
```
In the above syntax,

- *collection* is a collection or array variable which you have to loop through.
- *item* is a single item from the collection.

---

**How for-each loop works?**

Here's how the enhanced for loop works. For each iteration, for-each loop

- iterates through each item in the given collection or array (*collection*).
- stores each item in a variable (*item*)
- and executes the body of the loop.

Let's make it clear through an example.

---

**Example: for-each loop**

The program below calculates the sum of all elements of an integer array.

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

    int[] numbers = {3, 4, 5, -5, 0, 12};
    int sum = 0;

    for (int number: numbers) {
      sum += number;
    }

    // The sum of all elements is calculated and stored in the variable sum.
  }
}
```

M. Rajkumar, Assistant Professor, Computer Science and Engineering
When you run the program, the output will be:

Sum = 19

In the above program, the execution of foreach loop looks as:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Value of number</th>
<th>Value of sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>-5</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>19</td>
</tr>
</tbody>
</table>

You can see during each iteration, the foreach loop

- iterates through each element in the `numbers` variable
- stores it in the `number` variable
- and executes the body, i.e. adds `number` to `sum`
5. Legacy Classes and Interfaces

Early version of Java did not include the Collections framework. It only defined several classes and interfaces that provide methods for storing objects. When Collections framework were added in J2SE 1.2, the original classes were reengineered to support the collection interface. These classes are also known as Legacy classes. All legacy classes and interface were redesigned by JDK 5 to support Generics. In general, the legacy classes are supported because there is still some code that uses them.

The following are the legacy classes defined by `java.util` package:

1. Dictionary
2. HashTable
3. Properties
4. Stack
5. Vector

There is only one legacy interface called `Enumeration`

**NOTE:** All the legacy classes are synchronized

---

**Enumeration interface**

1. **Enumeration** interface defines method to enumerate (obtain one at a time) through collection of objects.
2. This interface is superseded (replaced) by `Iterator` interface.
3. However, some legacy classes such as `Vector` and `Properties` define several methods in which `Enumeration` interface is used.
4. It specifies the following two methods:

```java
boolean hasMoreElements() // It returns true when there are still more elements to extract,
and returns false when all the elements have been enumerated.

Object nextElement() // It returns the next object in the enumeration i.e. each call to
nextElement() method
```
obtains the next object in the enumeration. It throws NoSuchElementException when the enumeration is complete.

Vector class

1. Vector is similar to ArrayList which represents a dynamic array.
2. There are two differences between Vector and ArrayList. First, Vector is synchronized while ArrayList is not, and Second, it contains many legacy methods that are not part of the Collections Framework.
3. With the release of JDK 5, Vector also implements Iterable. This means that Vector is fully compatible with collections, and a Vector can have its contents iterated by the for-each loop.
4. Vector class has following four constructor
5. Vector() //This creates a default vector, which has an initial size of 10.
6. Vector(int size) //This creates a vector whose initial capacity is specified by size.
7. Vector(int size, int incr) //This creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time.
8. Vector(Collection c) //This creates a vector that contains the elements of collection c.

Vector defines several legacy methods. Lets see some important legacy methods defined by Vector class.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void addElement(E element)</td>
<td>adds element to the Vector</td>
</tr>
<tr>
<td>E elementAt(int index)</td>
<td>returns the element at specified index</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Enumeration elements()</td>
<td>returns an enumeration of element in vector</td>
</tr>
<tr>
<td>E firstElement()</td>
<td>returns first element in the Vector</td>
</tr>
<tr>
<td>E lastElement()</td>
<td>returns last element in the Vector</td>
</tr>
<tr>
<td>void removeAllElements()</td>
<td>removes all elements of the Vector</td>
</tr>
</tbody>
</table>

**Example of Vector**

```java
import java.util.*;
public class Test
{
public static void main(String[] args)
{
    Vector ve = new Vector();
    ve.add(10);
    ve.add(20);
    ve.add(30);
    ve.add(40);
    ve.add(50);
    ve.add(60);

    Enumeration en = ve.elements();

    while(en.hasMoreElements())
    {
        System.out.println(en.nextElement());
    }
}
}
```
Hashtable class

1. Like HashMap, Hashtable also stores key/value pair. However neither keys nor values can be null.

2. There is one more difference between HashMap and Hashtable that is Hashtable is synchronized while HashMap is not.

3. Hashtable has following four constructor

4. `Hashtable()` //This is the default constructor. The default size is 11.

5. 

6. `Hashtable(int size)` //This creates a hash table that has an initial size specified by size.

7. 

8. `Hashtable(int size, float fillratio)` //This creates a hash table that has an initial size specified by size and a fill ratio specified by fillRatio. This ratio must be between 0.0 and 1.0, and it determines how full the hash table can be before it is resized upward. Specifically, when the number of elements is greater than the capacity of the hash table multiplied by its fill ratio, the hash table is expanded.

10. If you do not specify a fill ratio, then 0.75 is used.

11. 

12. `Hashtable(Map< ? extends K, ? extends V> m)` //This creates a hash table that is initialized with the elements in m. The capacity of the hash table is set to twice the number of elements in m. The default load factor of 0.75 is used.
Example of HasTable

```java
import java.util.*;

class HashTableDemo {
    public static void main(String args[]) {
        Hashtable< String,Integer> ht = new Hashtable< String,Integer>();
        ht.put("a",new Integer(100));
        ht.put("b",new Integer(200));
        ht.put("c",new Integer(300));
        ht.put("d",new Integer(400));

        Set st = ht.entrySet();
        Iterator itr=st.iterator();
        while(itr.hasNext()) {
            Map.Entry m=(Map.Entry)itr.next();
            System.out.println(itr.getKey()+ " "+itr.getValue());
        }
    }
}
```

a 100
b 200
c 300
d 400

Difference between HashMap and Hashtable
### Properties class

1. **Properties** class extends **Hashtable** class.
2. It is used to maintain list of value in which both key and value are **String**
3. **Properties** class define two constructor
4. `Properties() //This creates a Properties object that has no default values`
5. `Properties(Properties propdefault) //This creates an object that uses propdefault for its default values.`
6. One advantage of **Properties** over **Hashtable** is that we can specify a default property that will be useful when no value is associated with a certain key.

   **Note:** In both cases, the property list is empty
7. In Properties class, you can specify a default property that will be returned if no value is associated with a certain key.

---

**Example of Properties class**
import java.util.*;

public class Test {

    public static void main(String[] args) {
        Properties pr = new Properties();
        pr.put("Java", "James Ghosling");
        pr.put("C++", "Bjarne Stroustrup");
        pr.put("C", "Dennis Ritchie");
        pr.put("C#", "Microsoft Inc.");
        Set< ?> creator = pr.keySet();

        for(Object ob: creator) {
            System.out.println(ob + " was created by " + pr.getProperty((String)ob) );
        }
    }
}

Java was created by James Ghosling
C++ was created by Bjarne Stroustrup
C was created by Dennis Ritchie
C# was created by Microsoft Inc

**Stack class**

1. Stack class extends Vector.
2. It follows last-in, first-out principle for the stack elements.
3. It defines only one default constructor

```java
Stack() //This creates an empty stack
```

4. If you want to put an object on the top of the stack, call `push()` method. If you want to remove and return the top element, call `pop()` method. An `EmptyStackException` is thrown if you call `pop()` method when the invoking stack is empty.

You can use `peek()` method to return, but not remove, the top object. The `empty()` method returns true if nothing is on the stack. The `search()` method determines whether an object exists on the stack and returns the number of pops that are required to bring it to the top of the stack.

---

**Example of Stack**

```java
import java.util.*;

class StackDemo {
    public static void main(String args[]) {
        Stack st = new Stack();
        st.push(11);
        st.push(22);
        st.push(33);
        st.push(44);
        st.push(55);
        Enumeration e1 = st.elements();
        while (e1.hasMoreElements())
            System.out.println(e1.nextElement() + " ");
        st.pop();
        st.pop();
        System.out.println("\nAfter popping out two elements");
    }
}
```
Enumeration e2 = st.elements();

while(e2.hasMoreElements())
System.out.print(e2.nextElement()+" ");

}
}

11 22 33 44 55
After popping out two elements
11 22 33

Dictionary class

1. Dictionary is an abstract class.
2. It represents a key/value pair and operates much like Map.
3. Although it is not currently deprecated, Dictionary is classified as obsolete, because it is fully superseded by Map class.

Unit IV

1. MVC Architecture

Before developing the web applications, we need to have idea about design models. There are two types of programming models (design models)

Model 1 Architecture

Model 2 (MVC) Architecture

Model 1 Architecture

Servlet and JSP are the main technologies to develop the web applications.
Servlet was considered superior to CGI. Servlet technology doesn't create process, rather it creates thread to handle request. The advantage of creating thread over process is that it doesn't allocate separate memory area. Thus many subsequent requests can be easily handled by servlet.

Problem in Servlet technology Servlet needs to recompile if any designing code is modified. It doesn't provide separation of concern. Presentation and Business logic are mixed up.

JSP overcomes almost all the problems of Servlet. It provides better separation of concern, now presentation and business logic can be easily separated. You don't need to redeploy the application if JSP page is modified. JSP provides support to develop web application using JavaBean, custom tags and JSTL so that we can put the business logic separate from our JSP that will be easier to test and debug.

As you can see in the above figure, there is picture which show the flow of the model1 architecture.

1. Browser sends request for the JSP page
2. JSP accesses Java Bean and invokes business logic
3. Java Bean connects to the database and get/save data
4. Response is sent to the browser which is generated by JSP

Advantage of Model 1 Architecture

- Easy and Quick to develop web application

Disadvantage of Model 1 Architecture

- Navigation control is decentralized since every page contains the logic to determine the next page. If JSP page name is changed that is referred by other pages, we need to change it in all the pages that leads to the maintenance problem.
- Time consuming You need to spend more time to develop custom tags in JSP. So that we don't need to use scriptlet tag.
- Hard to extend It is better for small applications but not for large applications.

Model 2 (MVC) Architecture
Model 2 is based on the MVC (Model View Controller) design pattern. The MVC design pattern consists of three modules model, view and controller.

**Model** The model represents the state (data) and business logic of the application.

**View** The view module is responsible to display data i.e. it represents the presentation.

**Controller** The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.

**Advantage of Model 2 (MVC) Architecture**

- **Navigation control is centralized** Now only controller contains the logic to determine the next page.
- **Easy to maintain**
- **Easy to extend**
- **Easy to test**
- **Better separation of concerns**

**Disadvantage of Model 2 (MVC) Architecture**

- We need to write the controller code self. If we change the controller code, we need to recompile the class and redeploy the application.

Visit here to get the [Example of MVC using Servlet and JSP](#).

**Solution of Model 2 Architecture: Configurable MVC Components**

It uses the declarative approach for defining view components, request mapping etc. It resolves the problem of Model 2 architecture. The Struts framework provides the configurable MVC support. In struts 2, we define all the action classes and view components in struts.xml file.

2. **Understanding Layout Managers**
The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. public static final int NORTH
2. public static final int SOUTH
3. public static final int EAST
4. public static final int WEST
5. public static final int CENTER

Constructors of BorderLayout class:

- BorderLayout(): creates a border layout but with no gaps between the components.
- JBorderLayout(int hgap, int vgap): creates a border layout with the given horizontal and vertical gaps between the components.

3. Event Handling

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener
interfaces for event handling.

Java Event classes and Listener interfaces

<table>
<thead>
<tr>
<th>Event Classes</th>
<th>Listener Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionEvent</td>
<td>ActionListener</td>
</tr>
<tr>
<td>MouseEvent</td>
<td>MouseListener and MouseMotionListener</td>
</tr>
<tr>
<td>MouseWheelEvent</td>
<td>MouseWheelListener</td>
</tr>
<tr>
<td>KeyEvent</td>
<td>KeyListener</td>
</tr>
<tr>
<td>ItemEvent</td>
<td>ItemListener</td>
</tr>
<tr>
<td>TextEvent</td>
<td>TextListener</td>
</tr>
<tr>
<td>AdjustmentEvent</td>
<td>AdjustmentListener</td>
</tr>
<tr>
<td>WindowEvent</td>
<td>WindowListener</td>
</tr>
<tr>
<td>ComponentEvent</td>
<td>ComponentListener</td>
</tr>
<tr>
<td>ContainerEvent</td>
<td>ContainerListener</td>
</tr>
<tr>
<td>FocusEvent</td>
<td>FocusListener</td>
</tr>
</tbody>
</table>

4. Handling mouse and keyboard events

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.
Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

```java
public abstract void mouseClicked(MouseEvent e);
public abstract void mouseEntered(MouseEvent e);
public abstract void mouseExited(MouseEvent e);
public abstract void mousePressed(MouseEvent e);
public abstract void mouseReleased(MouseEvent e);
```

5. Applets

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.
15. TUTORIAL TOPICS AND QUESTIONS
Unit I

Two marks Questions:

1. Define OOPS?

Ans: Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

The first step in OOP is to identify all the objects the programmer wants to manipulate and how they relate to each other, an exercise often known as data modeling. Once an object has been identified, it is generalized as a class of objects (think of Plato's concept of the "ideal" chair that stands for all chairs) which defines the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. Objects communicate with well-defined interfaces called messages.

2. Define encapsulation?

Ans: encapsulation is the inclusion of one thing within another thing so that the included thing is not apparent. Decapsulation is the removal or the making apparent a thing previously encapsulated.

In object-oriented programming, encapsulation is the inclusion within a program object of all the resources need for the object to function - basically, the methods and the data. The object is said to "publish its interfaces." Other objects adhere to these interfaces to use the object without having to be concerned with how the object accomplishes it. The idea is "don't tell me how you do it; just do it." An object can be thought of as a self-contained atom. The object interface consists of public methods and instantiated data.

In telecommunication, encapsulation is the inclusion of one data structure within another structure so that the first data structure is hidden for the time being. For example, a TCP/IP-
formatted data packet can be encapsulated within an ATM frame (another kind of transmitted data unit). Within the context of transmitting and receiving the ATM frame, the encapsulated packet is simply a stream of bits between the ATM data that describes the transfer.

3. Define inheritance?

**Ans:** In object-oriented programming, inheritance is the concept that when a class of objects is defined, any subclass that is defined can inherit the definitions of one or more general classes. This means for the programmer that an object in a subclass need not carry its own definition of data and methods that are generic to the class (or classes) of which it is a part. This not only speeds up program development; it also ensures an inherent validity to the defined subclass object (what works and is consistent about the class will also work for the subclass).

4. Define polymorphism?

**Ans:** Polymorphism (from the Greek meaning "having multiple forms") is the characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form. There are several different kinds of polymorphism.

A variable with a given name may be allowed to have different forms and the program can determine which form of the variable to use at the time of execution. For example, a variable named USERID may be capable of being either an integer (whole number) or a string of characters (perhaps because the programmer wants to allow a user to enter a user ID as either an employee number - an integer - or with a name - a string of characters). By giving the program a way to distinguish which form is being handled in each case, either kind can be recognized and handled.

A named function can also vary depending on the parameters it is given. For example, if given a variable that is an integer, the function chosen would be to seek a match against a list of employee numbers; if the variable were a string, it would seek a match against a list of
names. In either case, both functions would be known in the program by the same name. This type of polymorphism is sometimes known as overloading.

Polymorphism can mean, as in the ML language, a data type of "any," such that when specified for a list, a list containing any data types can be processed by a function. (For example, if a function simply determines the length of a list, it doesn't matter what data types are in the list.)

In PHP, polymorphism means that if B is a descendant of A and a function can accept A as a parameter, it can also accept B.

5. Define constructor?

Ans: A constructor is a special method of a class or structure in object-oriented programming that initializes an object of that type. A constructor is an instance method that usually has the same name as the class, and can be used to set the values of the members of an object, either to default or to user-defined values.

Three marks questions:

1. What are the concepts and rules used in object-oriented programming provide these important benefits?

Ans:

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.

- Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of data hiding provides greater system security and avoids unintended data corruption.
OBJECT ORIENTED PROGRAMMING THROUGH JAVA

- The definition of a class is reusable not only by the program for which it is initially created but also by other object-oriented programs (and, for this reason, can be more easily distributed for use in networks).

- The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.

2. Define string handling?

Ans: The basic aim of **String Handling** concept is storing the string data in the main memory (RAM), manipulating the data of the String, retrieving the part of the String etc. **String Handling** provides a lot of concepts that can be performed on a string such as concatenation of string, comparison of string, find sub string etc.

**Character**

It is an identifier enclosed within single quotes (').

Example: 'A', '$', 'p'

**String:**

String is a sequence of characters enclosed within double quotes (" ") is known as **String**.

Example: "Java Programming".

In java programming to store the character data we have a fundamental datatype called **char**. Similarly to store the string data and to perform various operation on String data, we have three predefined classes they are:

- String
- StringBuffer
- StringBuilder

3. Describe scope and life time of variables?

Ans: The scope of a variable defines the section of the code in which the variable is visible. As a general rule, variables that are defined within a block are not accessible outside that block. The lifetime of a variable refers to how long the variable exists before
it is destroyed. Destroying variables refers to deallocating the memory that was allotted to the variables when declaring it. We have written a few classes till now. You might have observed that not all variables are the same. The ones declared in the body of a method were different from those that were declared in the class itself. There are there types of variables: instance variables, formal parameters or local variables and local variables.

**Instance variables**

Instance variables are those that are defined within a class itself and not in any method or constructor of the class. They are known as instance variables because every instance of the class (object) contains a copy of these variables. The scope of instance variables is determined by the access specifier that is applied to these variables. We have already seen about it earlier. The lifetime of these variables is the same as the lifetime of the object to which it belongs. Object once created do not exist for ever. They are destroyed by the garbage collector of Java when there are no more reference to that object. We shall see about Java's automatic garbage collector later on.

**Argument variables**

These are the variables that are defined in the header oaf constructor or a method. The scope of these variables is the method or constructor in which they are defined. The lifetime is limited to the time for which the method keeps executing. Once the method finishes execution, these variables are destroyed.

**Local variables**

A local variable is the one that is declared within a method or a constructor (not in the header). The scope and lifetime are limited to the method itself.

One important distinction between these three types of variables is that access specifiers can be applied to instance variables only and not to argument or local variables.

In addition to the local variables defined in a method, we also have variables that are defined in blocks life an if block and an else block. The scope and is the same as that of the block itself.
4. List types of operators?

**Ans:** Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

5. Define type conversion?

**Ans:** When you assign value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not then they need to be casted or converted explicitly. For example, assigning an int value to a long variable.

**Widening or Automatic Type Conversion**

Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- When we assign value of a smaller data type to a bigger data type.
For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.

**Byte -> Short -> Int -> Long -> Float -> Double**

**Widening or Automatic Conversion**

**Five marks questions:**

1. What are the characteristics of Object Oriented programming language?

Ans:

**Encapsulation** – Encapsulation is capturing data and keeping it safely and securely from outside interfaces.

**Inheritance**- This is the process by which a class can be derived from a base class with all features of base class and some of its own. This increases code reusability.

**Polymorphism**- This is the ability to exist in various forms. For example an operator can be overloaded so as to add two integer numbers and two floats.

**Abstraction**- The ability to represent data at a very conceptual level without any details.

Some key features of the Object Oriented programming are:

- Emphasis on data rather than procedure
- Programs are divided into entities known as objects
- Data Structures are designed such that they characterize objects
- Functions that operate on data of an object are tied together in data structures
- Data is hidden and cannot be accessed by external functions
- Objects communicate with each other through functions
- New data and functions can be easily added whenever necessary
- Follows bottom up design in program design

The characteristics of OOP are:
OBJECT ORIENTED PROGRAMMING THROUGH JAVA

Class definitions – Basic building blocks OOP and a single entity which has data and operations on data together

Objects – The instances of a class which are used in real functionality – its variables and operations

Abstraction – Specifying what to do but not how to do; a flexible feature for having a overall view of an object’s functionality.

Encapsulation – Binding data and operations of data together in a single unit – A class adhere this feature

Inheritance and class hierarchy – Reusability and extension of existing classes

Polymorphism – Multiple definitions for a single name - functions with same name with different functionality; saves time in investing many function names Operator and Function overloading

Generic classes – Class definitions for unspecified data. They are known as container classes. They are flexible and reusable.

Class libraries – Built-in language specific classes

Message passing – Objects communicates through invoking methods and sending data to them. This feature of sending and receiving information among objects through function parameters is known as Message Passing.

2. Describe the features(buzzwords) of java programming language?

Ans: The main objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some awesome features which play important role in the popularity of this language. The features of Java are also known as java buzzwords. Following is a list of most important features of Java language. The Java Features given below are simple and easy to understand.

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

Simple
Java is very easy to learn and its syntax is simple, clean and easy to understand. According to Sun, Java language is simple because:

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

Object-oriented
Java is Object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming(OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

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

Platform Independent

Java is platform independent because it is different from other languages like C, C++ etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

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

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

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

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

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside virtual machine sandbox**
Classloader: Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to dynamically load Java classes into the Java Virtual Machine. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.

Bytecode Verifier: It checks the code fragments for illegal code that can violate access right to objects.

Security Manager: It determines what resources a class can access such as reading and writing to the local disk.

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

Robust

Robust simply means strong. Java is robust because:

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

Architecture-neutral

Java is architecture neutral because there is no implementation dependent features e.g. size of primitive types is fixed.

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

Portable

Java is portable because it facilitates you to carry the java bytecode to any platform.
High-performance

Java is faster than traditional interpretation since bytecode is "close" to native code still somewhat slower than a compiled language (e.g., C++). Java is an interpreted language, so it is also a reason that why it is slower than compiled language C, C++.

Distributed

Java is distributed because it facilitates us to create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Multi-threaded

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

3. Explain ad hoc polymorphism?

Ans: Ability of an organism to take different shapes is polymorphism in bio world. A simplest definition in computer terms would be, handling different data types using the same interface.

In this tutorial, we will learn about what is polymorphism in computer science and how polymorphism can be used in Java.

I wish this tutorial will help address the following,

- is overloading polymorphism?
- is overriding polymorphism?
- Ad hoc Polymorphism
- Parametric Polymorphism
- Coercion Polymorphism
- Inclusion or subtype Polymorphism
- what is static-binding?
- what is dynamic binding?
Polymorphism in computer science was introduced in 1967 by Christopher Strachey. Please let me know with reference if it is not a fact and the tutorial can be updated. Following are the two major types of polymorphism as defined by Strachey.

1. Ad hoc Polymorphism
2. Parametric Polymorphism

Later these were further categorized as below:

Ad hoc Polymorphism

"Ad-hoc polymorphism is obtained when a function works, or appears to work, on several different types (which may not exhibit a common structure) and may behave in unrelated ways for each type. Parametric polymorphism is obtained when a function works uniformly on a range of types; these types normally exhibit some common structure." – Strachey 1967
If we want to say the above paragraph in two words, they are operator overloading and function overloading. Determining the operation of a function based on the arguments passed.

Ad hoc Polymorphism in Java

In Java we have function overloading and we do not have operator overloading. Yes we have “+” operator implemented in a polymorphic way.

```
String fruits = "Apple" + "Orange";
int a = b + c;
```

The definition is when the type is different, the internal function adjusts itself accordingly. int and float are different types and so even the following can be included in polymorphism operator overloading.

```
int i = 10 - 3;
float f = 10.5 - 3.5;
```

Similarly even * and / can be considered as overloaded for int and float types.

Having said all the above, these are all language implemented features. Developers cannot custom overload an operator. So answer for the question, “does Java supports operator overloading?” is “yes and no”.

Java wholeheartedly supports function overloading. We can have same function name with different argument type list. For function overloading in Java I have already written a super-hit tutorial and I am sure you will enjoy reading it.
In inheritance, the ability to replace an inherited method in the subclass by providing a different implementation is overriding. Function overriding is discussed in the same tutorial as overloading.

Polymorphism is a larger concept which consists of all these different types. So it is not right to say that overloading or overriding alone is polymorphism. It is more than that.

Coercion Polymorphism

Implicit type conversion is called coercion polymorphism. Assume that we have a function with argument int. If we call that function by passing a float value and if the run-time is able to convert the type and use it accordingly then it is coercion polymorphism.

Now with this definition, let us see if Java has coercion polymorphism. The answer is half yes. Java supports widening type conversion and not narrowing conversions.

Narrowing Conversion

```java
class FToC {

    public static float fToC (int fahrenheit) {

        return (fahrenheit - 32)*5/9;
    }

    public static void main(String args[]) {

        System.out.println(fToC(98.4));
    }
}
```

Java does not support narrowing conversion and we will get error as "FToC.java:7: fToC(int) in FToC cannot be applied to (double)"
Widening Conversion

class FToC {
    public static float fToC (float fahrenheit) {
        return (fahrenheit - 32)*5/9;
    }
    public static void main(String args[]) {
        System.out.println(fToC(98));
    }
}

The above code will work without an error in Java. We are passing an int value ‘98’ wherein the expected value type is a float. Java implicitly converts int value to float and it supports widening conversion.

Universal Polymorphism

Universal polymorphism is the ability to handle types universally. There will be a common template structure available for operations definition irrespective of the types. Universal polymorphism is categorized into inclusion polymorphism and parametric polymorphism.

Inclusion polymorphism (subtype polymorphism)

Substitutability was introduced by eminent Barbara Liskov and Jeannette Wing. It is also called as Liskov substitution principle.

“Let T be a super type and S be its subtype (parent and child class). Then, instances (objects) of T can be substituted with instances of S.”

Replacing the supertype’s instance with a subtype’s instance. This is called inclusion polymorphism or subtype polymorphism. This is covariant type and the reverse of it is contra
variant. We have discussed the substitution principle and covariant types, contra variant and invariant earlier in the linked tutorial. This is demonstrated with a code example. Java supports subtype polymorphism from Java / JDK version 1.5.

Parametric Polymorphism

Here we go, we have come to ‘Generics’. This is a nice topic and requires a full detailed tutorial with respect to Java. For now, parametric polymorphism is the ability to define functions and types in a generic way so that it works based on the parameter passed at runtime. All this is done without compromising type-safety.

The following source code demonstrates a generics feature of Java. It gives the ability to define a class and parameterize the type involved. The behavior of the class is based on the parameter type passed when it is instantiated.

```java
package com.javapapers.java;

import java.util.ArrayList;

import java.util.List;

public class PapersJar {
    private List itemList = new ArrayList();

    public void add(T item) {
        itemList.add(item);
    }

    public T get(int index) {
        return itemList.get(index);
    }
}
```
public static void main(String args[]) {

    PapersJar papersStr = new PapersJar();
    papersStr.add("Lion");

    String str = papersStr.get(0);
    System.out.println(str);

    PapersJar papersInt = new PapersJar();
    papersInt.add(new Integer(100));

    Integer integerObj = papersInt.get(0);
    System.out.println(integerObj);
}

Static Binding vs Dynamic Binding

Give all the above polymorphism types, we can classify these under different two broad groups static binding and dynamic binding. It is based on when the binding is done with the corresponding values. If the references are resolved at compile time, then it is static binding and if the references are resolved at runtime then it is dynamic binding. Static binding and dynamic binding also called as early binding and late binding. Sometimes they are also referred as static polymorphism and dynamic polymorphism.

Let us take overloading and overriding for example to understand static and dynamic binding. In the below code, first call is dynamic binding. Whether to call the obey method of DomesticAnimal or Animal is resolve at runtime and so it is dynamic binding. In the second call, whether the method obey() or obey(String i) should be called is decided at compile time and so this is static binding.
package com.javapapers.java;

public class Binding {

    public static void main(String args[]) {

        Animal animal = new DomesticAnimal();

        System.out.println(animal.obey());

        DomesticAnimal domesticAnimal = new DomesticAnimal();

        System.out.println(domesticAnimal.obey("Ok!")));

    }
}

class Animal {

    public String obey() {

        return "No!";

    }
}

class DomesticAnimal extends Animal {

    public String obey() {

        return "Yes!";

    }

    public String obey(String i) {

        return i;

    }
}
Output:

Yes!

Ok!

Advantages of Polymorphism

- Generics: Enables generic programming.
- Extensibility: Extending an already existing system is made simple.
- De-clutters the object interface and simplifies the class blueprint.

4. Explain different types of inheritance in java

**Ans:** Below are various types of inheritance in Java. We will see each one of them one by one with the help of examples and flow diagrams.

1) Single Inheritance

**Single inheritance** is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a parent class of B and B would be a child class of A.

![Diagram](a) Single Inheritance
2) Multiple Inheritance

“Multiple Inheritance” refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with “multiple inheritance” is that the derived class will have to manage the dependency on two base classes.

3) Multilevel Inheritance

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A. For more details and example refer – Multilevel inheritance in Java.

4) Hierarchical Inheritance

In such kind of inheritance one class is inherited by many sub classes. In below example class B, C and D inherits the same class A. A is parent class (or base class) of B, C & D. Read More at – Hierarchical Inheritance in java with example program.

5) Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of Single and Multiple inheritance. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!! Using interfaces. Yes you heard it right. By using interfaces you can have multiple as well as hybrid inheritance in Java.

5. Benefits of inheritance

Ans: Not just in Java, but in General Inheritance in Object Oriented Programming has lot of advantages…

Advantages:-
One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses. Where equivalent code exists in two related classes, the hierarchy can usually be refactored to move the common code up to a mutual superclass. This also tends to result in a better organization of code and smaller, simpler compilation units.

Inheritance can also make application code more flexible to change because classes that inherit from a common superclass can be used interchangeably. If the return type of a method is superclass

**Reusability** - facility to use public methods of base class without rewriting the same.

**Extensibility** - extending the base class logic as per business logic of the derived class.

**Data hiding** - base class can decide to keep some data private so that it cannot be altered by the derived class.

**Overriding** - With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

**Disadvantages:**

- One of the main disadvantages of inheritance in Java (the same in other object-oriented languages) is the increased time/effort it takes the program to jump through all the levels of overloaded classes. If a given class has ten levels of abstraction above it, then it will essentially take ten jumps to run through a function defined in each of those classes.

- Main disadvantage of using inheritance is that the two classes (base and inherited class) get tightly coupled. This means one cannot be used independent of each other.

- Also with time, during maintenance adding new features both base as well as derived classes are required to be changed. If a method signature is changed then we will be affected in both cases (inheritance & composition).

- If a method is deleted in the "base class" or aggregate, then we will have to re-factor in case of using that method. Here things can get a bit complicated in case of inheritance because our programs will still compile, but the methods of the subclass
OBJECT ORIENTED PROGRAMMING THROUGH JAVA

will no longer be overriding base class methods. These methods will become
independent methods in their own right.

Objective question with answers

UNIT I

1. Which of the following are legal lines of Java code?
   1. int w = (int)888.8;
   2. byte x = (byte)100L;
   3. long y = (byte)100;
   4. byte z = (byte)100L;
   a) 1 and 2 b) 2 and 3 c) 3 and 4 d) All statements are correct.

2. An expression involving byte, int, and literal numbers is promoted to which of these?
   a) int b) long c) byte d) float

3. Which of these is necessary condition for automatic type conversion in Java?
   a) The destination type is smaller than source type.
   b) The destination type is larger than source type.
   c) The destination type can be larger or smaller than source type.
   d) None of the mentioned

4. What is the prototype of the default constructor of this class?
   public class prototype { }
   a) prototype( ) b) prototype(void) c) public prototype(void) d) public prototype( )
5. What is Truncation in Java?  ( )
a) Floating-point value assigned to an integer type.
b) Integer value assigned to floating type.
c) Floating-point value assigned to an Floating type.
d) Integer value assigned to floating type.

6. Which of these can be returned by the operator &?  ( )
a) Integer b) Boolean c) Character d) Integer or Boolean

7. Literals in Java must be preceded by which of these?  ( )
a) L  b) l  c) D  d) L and I

8. Which of these operators is used to allocate memory to array variable in Java?  ( )
a) malloc b) alloc c) new  d) new malloc

9. What will this code print?  ( )
int arr[] = new int [5];  
System.out.print(arr);
a) 0  b) value stored in arr[0]. c) 00000  d) Garbage value

10. Which of these is necessary to specify at time of array initialization?  ( )
a) Row  b) Column c) Both Row and Column d) None of the mentioned

Answers

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D</td>
<td>6</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>7</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>8</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>9</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

Fill in the blanks questions with answers

1. _______ and character can be operands of arithmetic operators
2. _______. Modulus operator, %, can be applied.
3. _______ can be used only with numeric and character data type
4. _______ have highest precedence
5. _______ should be expression1 evaluate to in using ternary operator as in this line?
6. _______ has higher precedence than multiplication operator.
7. ________ is the output of relational operators
8. ________ selection statements test only for equality?
9. ________ selection statements in Java.

10. ________ will execute the body of loop even when condition controlling the loop is initially false

Answers

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Numeric</td>
<td>6</td>
<td>Division operator</td>
</tr>
<tr>
<td>2</td>
<td>Integers and floating point</td>
<td>7</td>
<td>Boolean</td>
</tr>
<tr>
<td>3</td>
<td>Assignment operator</td>
<td>8</td>
<td>Switch</td>
</tr>
<tr>
<td>4</td>
<td>&gt;&gt;</td>
<td>9</td>
<td>If</td>
</tr>
<tr>
<td>5</td>
<td>Boolean</td>
<td>10</td>
<td>While loop</td>
</tr>
</tbody>
</table>

Unit II

TWO marks

1. Define package

Ans: A package is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer. You might keep HTML pages in one folder, images in another, and scripts or applications in yet another. Because software written in the Java programming language can be composed of hundreds or thousands of individual classes, it makes sense to keep things organized by placing related classes and interfaces into packages.

The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications. This library is known as the "Application Programming Interface", or "API" for short. Its packages represent the tasks most commonly associated with general-purpose programming. For example, a String object contains state and behavior for
OBJECT ORIENTED PROGRAMMING THROUGH JAVA

character strings; a `File` object allows a programmer to easily create, delete, inspect, compare, or modify a file on the filesystem; a `Socket` object allows for the creation and use of network sockets; various GUI objects control buttons and checkboxes and anything else related to graphical user interfaces. There are literally thousands of classes to choose from. This allows you, the programmer, to focus on the design of your particular application, rather than the infrastructure required to make it work.

The Java Platform API Specification contains the complete listing for all packages, interfaces, classes, fields, and methods supplied by the Java SE platform. Load the page in your browser and bookmark it. As a programmer, it will become your single most important piece of reference documentation.

2. Difference between public, private and protected?
Ans: There is four access modifier in Java namely public, private, protected and the default access modifier, also known as package level modifier. The difference between these access modifier comes in their ability to restrict access to a class, method or variables, public is the least restrictive access modifier while private is the most restrictive access modifier, package and protected lies in between. Another key difference between public, protected, package and private modifier come from the point where you can apply them, for example, you cannot use private or protected modifier with a top level class but you can use public modifier there.

The default or package modifier is little bit special, when you do not specify any of the public, protected and private modifier, which is also the keywords, then Java automatically apply a default modifier (no it doesn't use default keyword), which means the said class, method or member will only be accessible inside the package it has declared.

3. Define interface?
Ans: An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

4. Define generics?
Ans: Java `Generic` methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.
5. Difference between byte stream and character stream

Ans: A stream is a way of sequentially accessing a file. In Streams you can process the data one at a time as bulk operations are unavailable with them. But, streams supports a huge range of source and destinations including disk file, arrays, other devices, other programs etc. In Java, a byte is not the same thing as a char. Therefore a byte stream is different from a character stream. So, Java defines two types of streams: Byte Streams and Character Streams.

Byte Streams

A byte stream access the file byte by byte. Java programs use byte streams to perform input and output of 8-bit bytes. It is suitable for any kind of file, however not quite appropriate for text files. For example, if the file is using a unicode encoding and a character is represented with two bytes, the byte stream will treat these separately and you will need to do the conversion yourself. Byte oriented streams do not use any encoding scheme while Character oriented streams use character encoding scheme (UNICODE). All byte stream classes are descended from InputStream and OutputStream.

Three marks Questions:

1. Difference between Autoboxing and Unboxing in Java

Ans: Autoboxing: Converting a primitive value into an object of the corresponding wrapper class is called autoboxing. For example, converting int to Integer class. The Java compiler applies autoboxing when a primitive value is:

- Passed as a parameter to a method that expects an object of the corresponding wrapper class.
- Assigned to a variable of the corresponding wrapper class.

Unboxing: Converting an object of a wrapper type to its corresponding primitive value is called unboxing. For example conversion of Integer to int. The Java compiler applies unboxing when an object of a wrapper class is:

- Passed as a parameter to a method that expects a value of the corresponding primitive type.
- Assigned to a variable of the corresponding primitive type.

2. Give Example for Java Serialization

Ans:

```java
import java.io.*;
```

112

M. Rajkumar, Assistant Professor,
Computer Science and Engineering
class Persist{
    public static void main(String args[]) throws Exception{
        Student s1 = new Student(211, "ravi");

        FileOutputStream fout = new FileOutputStream("f.txt");
        ObjectOutputStream out = new ObjectOutputStream(fout);

        out.writeObject(s1);
        out.flush();
        System.out.println("success");
    }
}

3. Define byte stream?
Ans: Byte Streams
Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, FileInputStream and FileOutputStream.

4. Explain reading-and-writing-files-in
Ans: FileReader for text files in your system's default encoding (for example, files containing Western European characters on a Western European computer).
FileStream for binary files and text files that contain 'weird' characters.
FileReader (for text files) should usually be wrapped in a BufferedReader. This saves up data so you can deal with it a line at a time or whatever instead of character by character (which usually isn't much use).

If you want to write files, basically all the same stuff applies, except you'll deal with classes named FileWriter with BufferedWriter for text files, or FileOutputStream for binary files.

5. Explain Reading Binary Files in Java
Ans: If you want to read a binary file, or a text file containing 'weird' characters (ones that your system doesn't deal with by default), you need to use FileInputStream instead of FileReader. Instead of wrapping FileInputStream in a buffer, FileInputStream defines a
method called `read()` that lets you fill a buffer with data, automatically reading just enough bytes to fill the buffer (or less if there aren't that many bytes left to read).

**Five marks Questions:**

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

   **Stream**

   A stream can be defined as a sequence of data. There are two kinds of Streams –
   - **InputStream** – The InputStream is used to read data from a source.
   - **OutputStream** – The OutputStream is used for writing data to a destination.

   Java provides strong but flexible support for I/O related to files and networks but this tutorial covers very basic functionality related to streams and I/O. We will see the most commonly used examples one by one –

   **Byte Streams**

   Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**.

   **Character Streams**

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

2. **Explain java access protection?**
Ans: Packages adds another dimension to the access control. As you will see, Java provides many levels of protection to allow fine-grained control over visibility of the variables and methods within classes, subclasses, and packages.

Classes and packages are means of encapsulating and containing the name space and scope of the variables and methods.

Packages behaves as containers for classes and other subordinate packages.

Classes act as containers for data and code.

**Class Members Visibility**

The Java's smallest unit of abstraction is class. Because of the interplay between the classes and packages, Java addresses the following four categories of visibility for class members:

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

The three access modifiers are:

- public
- private
- protected

provides a variety of ways to produce many levels of access required by these categories. The upcoming table sums up the interaction:

While the access control mechanism of Java may seem complicated, we can simplify it as follows.

Anything declared as **public** can be accessed from anywhere.

Anything declared as **private** can't be seen outside of its class.

**Class Member Access**

When a member doesn't have an explicit access specification, then it is visible to the subclasses as well as to the other classes in the same package. This is the default access. And if you want to allow an element to be seen outside your current package, but only to the classes that subclass your class directly, then declare that element protected.

<table>
<thead>
<tr>
<th>Access Modifier</th>
<th>Private</th>
<th>Protected</th>
<th>Public</th>
<th>No Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>private</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>protected</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access Control</td>
<td>Same class</td>
<td>Same package subclass</td>
<td>Same package non-subclass</td>
<td>Different package subclass</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>-----------------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

This table applies only to the members of classes. A non-nested class has only two possible access levels i.e., default and public.

When a class is declared as public, then it is accessible by any other code. If a class has default access, then it can only be accessed by the other code within its same package. When a class is public, it must be only the public class declared in the file that must have the same name as the class.

**Java Access Protection Example**

Here the upcoming example allows all the combinations of access control modifiers. This example has two packages and five classes.

Always remember that the classes for the two different packages need to be stored in directories after their respective packages (in this case pkg1 and pkg2).

The source for the first package defines the three classes i.e., Protection, Derived, and SamePackage. The first class defines the four variables of type int in each of the legal protection modes. The variable n declared with the default protection, the variables n_priv, n_prot, and n_publ is private, protected, and public respectively.

Each subsequent class in the following example will try to access the variables in an instance of this class. The lines that will not compile due to the access restrictions are commented out. Before each of these lines is a comment that listing the places from which this level of protection would allow access.

The second class named Derived, is a subclass of Protection in the same package, pkg1. This grants Derived access to every variable in the class Protection except for n_priv, the private one. The third class named SamePackage, is not a subclass of the class Protection, but is in the same package and also has access to all but not n_priv.

This is Protection.java file:

```java
package pkg1;
```
public class Protection {
    int n = 1;
    private int n_priv = 2;
    protected int n_prot = 3;
    public int n_publ = 4;

    public Protection() {
        System.out.println("base constructor");
        System.out.println("n = " + n);
        System.out.println("n_priv = " + n_priv);
        System.out.println("n_prot = " + n_prot);
        System.out.println("n_publ = " + n_publ);
    }
}

This is Derived.java file:

package pkg1;

class Derived extends Protection {
    Derived() {
        System.out.println("derived constructor");
        System.out.println("n = " + n);

        /* class only
         * System.out.println("n_priv = " + n_priv);
         */
        System.out.println("n_prot = " + n_prot);
        System.out.println("n_publ = " + n_publ);
    }
}
This is \texttt{SamePackage.java} file:

```java
package pkg1;

class SamePackage {
    SamePackage() {
        Protection pro = new Protection();
        System.out.println("same package constructor");
        System.out.println("n = " + pro.n);

        /* class only
         * System.out.println("n\_priv = " + pro.n\_priv); */

        System.out.println("n\_prot = " + pro.n\_prot);
        System.out.println("n\_publ = " + pro.n\_publ);
    }
}
```

Following is the source code for the other package named \texttt{pkg2}. The two classes defined in the package \texttt{pkg2} cover the outer two conditions that are affected by the access control. The first class named \texttt{Protection2}, is a subclass of \texttt{pkg1.Protection}. This grants access to all of \texttt{pkg1}. Variables of the class \texttt{Protection} except for \texttt{n\_priv} (because it is private) and \texttt{n}, the variable declared with the default protection.

\textbf{Always remember} that the default only allows access from within the class or the package, not extra-package subclasses. Finally, the class \texttt{OtherPackage} has access to \texttt{n\_publ} only which was declared as \texttt{public}.

This is \texttt{Protection2.java} file:

```java
package pkg2;

class Protection2 extends pkg1.Protection {
    Protection2() {
    }
}
This is OtherPackage.java file:

package pkg2;

class OtherPackage
{
    OtherPackage()
    {
        pkg1.Protection pro = new pkg1.Protection();
        System.out.println("other package constructor");

        /* class or package only
        *  System.out.println("n = "+ pro.n); */

        /* class only
        *  System.out.println("n_priv = "+ pro.n_priv); */

        System.out.println("n_prot = "+ pro.n_prot);
        System.out.println("n_publ = "+ pro.n_publ);
    }
}
If you want to try these two packages, here are two test files you can use.
The one for package **pkg1** is shown here:

```
/* demo package pkg1 */

package pkg1;

/* instantiate the various classes in pkg1 */
public class Demo {
    public static void main(String args[]) {
        Protection obj1 = new Protection();
        Derived obj2 = new Derived();
        SamePackage obj3 = new SamePackage();
    }
}
```

The test file for **pkg2** is shown below:

```
/* demo package pkg2 */

package pkg2;

/* instantiate the various classes in pkg2 */
public class Demo {
    public static void main(String args[]) {
        Protection2 obj1 = new Protection2();
        OtherPackage obj2 = new OtherPackage();
    }
}
```
3. Write different Ways to read input from console in Java

Ans: In Java, there are three different ways for reading input from the user in the command line environment (console).

1. Using BufferedReader Class
This is the Java classical method to take input, Introduced in JDK 1.0. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

Advantages
- The input is buffered for efficient reading.

Drawback:
- The wrapping code is hard to remember.

2. Using Scanner Class
This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however it is also can be used to read input from the user in the command line.

Advantages:
- Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
- Regular expressions can be used to find tokens.

Drawback:
- The reading methods are not synchronized

To see more differences, please see this article.

3. Using Console Class
It has been becoming a preferred way for reading user’s input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).

Advantages:
- Reading password without echoing the entered characters.
- Reading methods are synchronized.
- Format string syntax can be used.

Drawback:
- Does not work in non-interactive environment (such as in an IDE).
4. Explain interface?

Ans: An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways –

- An interface can contain any number of methods.
- An interface is written in a file with a `.java` extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a `.class` file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including –

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaring Interfaces

The `interface` keyword is used to declare an interface. Here is a simple example to declare an interface –

Example

Following is an example of an interface –
/* File name : NameOfInterface.java */
import java.lang.*;
// Any number of import statements

public interface NameOfInterface {
    // Any number of final, static fields
    // Any number of abstract method declarations
}

Interfaces have the following properties –
  • An interface is implicitly abstract. You do not need to use the abstract keyword while
    declaring an interface.
  • Each method in an interface is also implicitly abstract, so the abstract keyword is not
    needed.
  • Methods in an interface are implicitly public.

Example

/* File name : Animal.java */
interface Animal {
    public void eat();
    public void travel();
}

Implementing Interfaces

When a class implements an interface, you can think of the class as signing a contract,
agreeing to perform the specific behaviors of the interface. If a class does not perform all the
behaviors of the interface, the class must declare itself as abstract.

A class uses the implements keyword to implement an interface. The implements keyword
appears in the class declaration following the extends portion of the declaration.

Example

/* File name : MammalInt.java */
public class MammalInt implements Animal {
    public void eat() {
        System.out.println("Mammal eats");
    }
}
public void travel() {
    System.out.println("Mammal travels");
}
public int noOfLegs() {
    return 0;
}
public static void main(String args[]) {
    MammalInt m = new MammalInt();
    m.eat();
    m.travel();
}

This will produce the following result –
Output
Mammal eats
Mammal travels

When overriding methods defined in interfaces, there are several rules to be followed –

- Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
- The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
- An implementation class itself can be abstract and if so, interface methods need not be implemented.

When implementing interfaces, there are several rules –

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, in a similar way as a class can extend another class.

Extending Interfaces
An interface can extend another interface in the same way that a class can extend another class. The `extends` keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.

Example

```java
// Filename: Sports.java
public interface Sports {
    public void setHomeTeam(String name);
    public void setVisitingTeam(String name);
}

// Filename: Football.java
public interface Football extends Sports {
    public void homeTeamScored(int points);
    public void visitingTeamScored(int points);
    public void endOfQuarter(int quarter);
}

// Filename: Hockey.java
public interface Hockey extends Sports {
    public void homeGoalScored();
    public void visitingGoalScored();
    public void endOfPeriod(int period);
    public void overtimePeriod(int ot);
}
```

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

Extending Multiple Interfaces

A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface. The `extends` keyword is used once, and the parent interfaces are declared in a comma-separated list.
For example, if the Hockey interface extended both Sports and Event, it would be declared as:

Example

```java
public interface Hockey extends Sports, Event
```

### Tagging Interfaces

The most common use of extending interfaces occurs when the parent interface does not contain any methods. For example, the MouseListener interface in the java.awt.event package extended java.util.EventListener, which is defined as:

Example

```java
package java.util;
public interface EventListener {}
```

An interface with no methods in it is referred to as a **tagging** interface. There are two basic design purposes of tagging interfaces:

- **Creates a common parent** – As with the EventListener interface, which is extended by dozens of other interfaces in the Java API, you can use a tagging interface to create a common parent among a group of interfaces. For example, when an interface extends EventListener, the JVM knows that this particular interface is going to be used in an event delegation scenario.

- **Adds a data type to a class** – This situation is where the term, tagging comes from. A class that implements a tagging interface does not need to define any methods (since the interface does not have any), but the class becomes an interface type through polymorphism.

5. Explain Generics?

Ans: It would be nice if we could write a single sort method that could sort the elements in an Integer array, a String array, or an array of any type that supports ordering.

Java **Generic** methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.

Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.
Object Oriented Programming Through Java

Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements.

Generic Methods

You can write a single generic method declaration that can be called with arguments of different types. Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately. Following are the rules to define Generic Methods –

- All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type ( < E > in the next example).
- Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
- The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
- A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char).

Example

Following example illustrates how we can print an array of different type using a single Generic method –

Live Demo

```java
public class GenericMethodTest {
    // generic method printArray
    public static < E > void printArray( E[] inputArray ) {
        // Display array elements
        for(E element : inputArray) {
            System.out.printf("%s ", element);
        }
        System.out.println();
    }
}
```
public static void main(String args[]) {

    // Create arrays of Integer, Double and Character
    Integer[] intArray = { 1, 2, 3, 4, 5 };  
    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };  
    Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };  
    System.out.println("Array integerArray contains:");
    printArray(intArray);  // pass an Integer array
    System.out.println("nArray doubleArray contains:");
    printArray(doubleArray);  // pass a Double array
    System.out.println("nArray characterArray contains:");
    printArray(charArray);  // pass a Character array

    System.out.println("Array integerArray contains:");
    System.out.println(1 2 3 4 5);
    System.out.println("Array doubleArray contains:");
    System.out.println(1.1 2.2 3.3 4.4);
    System.out.println("Array characterArray contains:");
    System.out.println(H E L L O);
}

This will produce the following result –

Output

Array integerArray contains:
1 2 3 4 5

Array doubleArray contains:
1.1 2.2 3.3 4.4

Array characterArray contains:
H E L L O

Bounded Type Parameters

There may be times when you'll want to restrict the kinds of types that are allowed to be passed to a type parameter. For example, a method that operates on numbers might only want to accept instances of Number or its subclasses. This is what bounded type parameters are for.

To declare a bounded type parameter, list the type parameter's name, followed by the extends keyword, followed by its upper bound.

Example
Following example illustrates how extends is used in a general sense to mean either "extends" (as in classes) or "implements" (as in interfaces). This example is Generic method to return the largest of three Comparable objects –

Live Demo

```java
public class MaximumTest {
    // determines the largest of three Comparable objects
    public static <T extends Comparable<T>> T maximum(T x, T y, T z) {
        T max = x;   // assume x is initially the largest
        if(y.compareTo(max) > 0) {
            max = y;   // y is the largest so far
        }
        if(z.compareTo(max) > 0) {
            max = z;   // z is the largest now
        }
        return max;   // returns the largest object
    }
    public static void main(String args[]) {
        System.out.printf("Max of %d, %d and %d is %d\n\n", 3, 4, 5, maximum( 3, 4, 5 ));
        System.out.printf("Max of %.1f,%.1f and %.1f is %.1f\n\n", 6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ));
        System.out.printf("Max of %s, %s and %s is %s\n\n", "pear", "apple", "orange", maximum("pear", "apple", "orange");
    }
}
```

This will produce the following result –

Output

Max of 3, 4 and 5 is 5
Max of 6.6, 8.8 and 7.7 is 8.8
Max of pear, apple and orange is pear

Generic Classes

A generic class declaration looks like a non-generic class declaration, except that the class name is followed by a type parameter section.

As with generic methods, the type parameter section of a generic class can have one or more type parameters separated by commas. These classes are known as parameterized classes or parameterized types because they accept one or more parameters.

Example

Following example illustrates how we can define a generic class –

```java
public class Box<T> {
    private T t;
    public void add(T t) {
        this.t = t;
    }
    public T get() {
        return t;
    }
    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        Box<String> stringBox = new Box<String>();
        integerBox.add(new Integer(10));
        stringBox.add(new String("Hello World"));
        System.out.printf("Integer Value :%d\n", integerBox.get());
        System.out.printf("String Value :%s\n", stringBox.get());
    }
}
```

This will produce the following result –

Output

```
Integer Value :10
String Value :Hello World
```
### Objective question with answers

1. Which of these keyword must be used to inherit a class?
   - a) super
   - b) this
   - c) extent
   - d) extends

2. Which of these keywords is used to refer to member of base class from a sub class?
   - a) upper
   - b) super
   - c) this
   - d) None of the mentioned

3. A class member declared protected becomes member of subclass of which type?
   - a) public member
   - b) private member
   - c) protected member
   - d) static member

4. Which of these is correct way of inheriting class A by class B?
   - a) class B + class A {}
   - b) class B inherits class A {}
   - c) class B extends A {}
   - d) class B extends class A {}

5. Which of the following statements are incorrect?
   - a) public members of class can be accessed by any code in the program.
   - b) private members of class can only be accessed by other members of the class.
   - c) private members of class can be inherited by a sub class, and become protected members in sub class.
   - d) protected members of a class can be inherited by a sub class, and become private members of the sub class.

6. Which of these keywords is used to define packages in Java?
   - a) pkg
   - b) Pkg
   - c) package
   - d) Package

7. Which of these is a mechanism for naming and visibility control of a class and its content?
   - a) Object
   - b) Packages
   - c) Interfaces
   - d) None of the Mentioned.

8. Which of this access specifies can be used for a class so that its members can be accessed by a different class in the same package?
   - a) Public
   - b) Protected
   - c) No Modifier
   - d) All of the mentioned

9. Which of these access specifiers can be used for a class so that it’s members can be accessed by a different class in the different package?
   - a) Public
   - b) Protected
   - c) Private
   - d) No Modifier
10. Which of the following is correct way of importing package ‘pkg’? ( )
a) import pkg.  b) Import pkg.  c) import pkg.*  d) Import pkg.*

Answers:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D</td>
<td>6</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>7</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>8</td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>9</td>
<td>a</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>10</td>
<td>C</td>
</tr>
</tbody>
</table>

**Fill in the blanks question with answers**

1. A package can be renamed without renaming ______ in which the classes are stored.
2. ________ package stores all the standard java classes
3. ________ keywords is used to define interfaces in Java
4. ________ can be used to fully abstract a class from its implementation
5. ________ access specifiers can be used for an interface?
6. ________ keywords is used by a class to use an interface defined previously
7. ________ is correct way of implementing an interface salary by class
8 Interfaces are specified ________ if they are to be accessed by any code in the program.
9. ________ package stores all the standard java classes?
10. ________ keyword can be used in subclass to call the constructor of superclass

Answers:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Directory</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Interface</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Implementation</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Public</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>Implements</td>
<td>10</td>
</tr>
</tbody>
</table>

**Unit III**

**Two marks:**

M. Rajkumar, Assistant Professor,
Computer Science and Engineering
1. Types of Exception

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

1. Checked Exception
2. Unchecked Exception
3. Error

2. Difference between checked and unchecked exceptions

Ans: Checked Exception

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

Unchecked Exception

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

Error

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

3. Write Java Exception Handling Keywords

Ans: There are 5 keywords used in java exception handling.

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

4. Define Thread Priorities

Ans: Each thread has its own priority in Java. Thread priority is an absolute integer value. Thread priority decides only when a thread switches from one running thread to next, called context switching. Priority does increase the running time of the thread or gives faster execution.

5. Define Messaging

Ans: A program is a collection of more than one thread. Threads can communicate with each other. Java supports messaging between the threads with lost-cost. It provides methods to all
objects for inter-thread communication. As a thread exits from synchronization state, it notifies all the waiting threads.

Three marks:

1. Define Uncaught exceptions?
   Ans: The java.lang.ThreadGroup.uncaughtException() method is called by the Java Virtual Machine when a thread in this thread group stops because of an uncaught exception, and the thread does not have a specific Thread.UncaughtExceptionHandler installed.

2. Write the use of try & catch?
   Ans: The try block contains set of statements where an exception can occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block or both.

3. Write multiple catch clauses?
   Ans: Java Multi catch block
   If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.
   Let's see a simple example of java multi-catch block.

```java
public class TestMultipleCatchBlock{
    public static void main(String args[]){
        try{
            int a[]=new int[5];
            a[5]=30/0;
        }
        catch(ArithmeticException e){System.out.println("task1 is completed");}
        catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
        catch(Exception e){System.out.println("common task completed");}

        System.out.println("rest of the code...");
    }
}
```

4. What are built-in exceptions
   Ans: Java defines several exception classes inside the standard package java.lang.
The most general of these exceptions are subclasses of the standard type RuntimeException. Since java.lang is implicitly imported into all Java programs, most exceptions derived from RuntimeException are automatically available.

5. What do you know about Inter-thread communication in Java

Ans: Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other. Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of Object class:

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

Five marks:

1. Explain Hierarchy of Java Exception classes
2. Differences between thread-based multitasking and process-based multitasking?

**Ans:** Prerequisite – Thread, Difference between multitasking, multithreading and multiprocessing

A **multitasking operating system** is an operating system that gives you the perception of 2 or more tasks/jobs/processes running at the same time. It does this by dividing system resources amongst these tasks/jobs/processes and switching between the tasks/jobs/processes while they are executing over and over again. Usually CPU processes only one task at a time but the switching is so fast that it looks like CPU is executing multiple processes at a time. They can support either **preemptive multitasking**, where the OS doles out time to applications.
OBJECT ORIENTED PROGRAMMING THROUGH JAVA

(virtually all modern OSes) or cooperative multitasking, where the OS waits for the program
to give back control (Windows 3.x, Mac OS 9 and earlier), leading to hangs and crashes. Also
known as **Timesharing**, multitasking is a logical extension of multiprogramming.

**Multitasking programming is of two types –**

1. Process-based Multitasking
2. Thread-based Multitasking.

**Process Based Multitasking Programming –**

- In process based multitasking two or more processes and programs can be run concurrently.
- In process based multitasking a process or a program is the smallest unit.
- Program is a bigger unit.
- Process based multitasking requires more overhead.
- Process requires its own address space.
- Process to Process communication is expensive.
- Here, it is unable to gain access over idle time of CPU.
- It is comparatively heavy weight.
- It has slower data rate multi-tasking.

**Example** – We can listen to music and browse internet at the same time. The processes in
this example are the music player and browser.

**Thread Based Multitasking Programming –**

- In thread based multitasking two or more threads can be run concurrently.
- In thread based multitasking a thread is the smallest unit.
- Thread is a smaller unit.
- Thread based multitasking requires less overhead.
- Threads share same address space.
- Thread to Thread communication is not expensive.
- It allows taking gain access over idle time taken by CPU.
- It is comparatively light weight.
- It has faster data rate multi-tasking.

**Examples** – Using a browser we can navigate through the webpage and at the same time
download a file. In this example, navigation is one thread and downloading is another thread.
Also in a word-processing application like MS Word, we can type text in one thread and spell
checker checks for mistakes in another thread.
3. Explain Java thread model

**Ans:** The Java language and its run-time system was designed keeping in mind about multithreading. The run-time system depend upon multithreading. Java provides asynchronous thread environment, this helps to increase the utilization of CPU.

Multithreading is best in all cases in contrast with single-thread model. Single-thread system uses an approach of event loop with polling. According to this approach a single thread in the system runs in an infinite loop. Polling the mechanism, that selects a single event from the event queue to choose what to do next. As the event is selected, then event loop forwards the control to the corresponding required event handler. Nothing else can be happened, until the event handler returns. Because of this CPU time is wasted. Here, only one part of the complete program is dominating the whole system, and preventing the system to execute or start any other process. In single-thread model one thread blocks all other threads until its execution completes. On other waiting or idle thread can start and acquire the resource which is not in use by the current thread. This causes the wastage of resources.

Java's multithreading provides benefit in this area by eliminating the loop and polling mechanism, one thread can be paused without stopping the other parts of the program. If any thread is paused or blocked, still other threads continue to run.

As the process has several states, similarly a thread exists in several states. A thread can be in the following states:

*Ready to run (New):* First time as soon as it gets CPU time.

*Running:* Under execution.

*Suspended:* Temporarily not active or under execution.

*Blocked:* Waiting for resources.

*Resumed:* Suspended thread resumed, and start from where it left off.

*Terminated:* Halts the execution immediately and never resumes.

Java thread model can be defined in the following three sections:

**Thread Priorities**

Each thread has its own priority in Java. Thread priority is an absolute integer value. Thread priority decides only when a thread switches from one running thread to next, called *context switching*. Priority does increase the running time of the thread or gives faster execution.
Synchronization

Java supports an asynchronous multithreading, any number of thread can run simultaneously without disturbing other to access individual resources at different instant of time or shareable resources. But some time it may be possible that shareable resources are used by at least two threads or more than two threads, one has to write at the same time, or one has to write and other thread is in the middle of reading it. For such type of situations and circumstances Java implements synchronization model called monitor. The monitor was first defined by C.A.R. Hoare. You can consider the monitor as a box, in which only one thread can reside. As a thread enter in monitor, all other threads have to wait until that thread exits from the monitor. In such a way, a monitor protects the shareable resources used by it being manipulated by other waiting threads at the same instant of time. Java provides a simple methodology to implement synchronization.

Messaging

A program is a collection of more than one thread. Threads can communicate with each other. Java supports messaging between the threads with lost-cost. It provides methods to all objects for inter-thread communication. As a thread exits from synchronization state, it notifies all the waiting threads.

4. Difference between wait and sleep?

Ans: Let’s see the important differences between wait and sleep methods.

<table>
<thead>
<tr>
<th>wait()</th>
<th>sleep()</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait() method releases the lock</td>
<td>sleep() method doesn't release the lock.</td>
</tr>
<tr>
<td>is the method of Object class</td>
<td>is the method of Thread class</td>
</tr>
<tr>
<td>is the non-static method</td>
<td>is the static method</td>
</tr>
<tr>
<td>is the non-static method</td>
<td>is the static method</td>
</tr>
<tr>
<td>should be notified by notify() or notifyAll() methods</td>
<td>after the specified amount of time, sleep is completed.</td>
</tr>
</tbody>
</table>
5. Explain Inter-thread communication in Java

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

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

- `wait()`:
  - Causes current thread to release the lock and wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.
  - The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public final void wait() throws InterruptedException</code></td>
<td>waits until object is notified.</td>
</tr>
<tr>
<td><code>public final void wait(long timeout) throws InterruptedException</code></td>
<td>waits for the specified amount of time.</td>
</tr>
</tbody>
</table>

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

- `notifyAll()`:
  - Wakes up all threads that are waiting on this object's monitor.
  - Syntax:
    ```java
    public final void notifyAll()
    ```
Objective question with answers

1. When does Exceptions in Java arises in code sequence?
   a) Run Time   b) Compilation Time
   c) Can Occur Any Time d) None of the mentioned

2. Which of these keywords is not a part of exception handling?
   a) try  b) finally  c) thrown  d) catch

3. Which of these keywords must be used to monitor for exceptions?
   a) try  b) finally  c) throw  d) catch

4. Which of these keywords must be used to handle the exception thrown by try block in some rational manner?
   a) try  b) finally  c) throw  d) catch

5. Which of these keywords is used to manually throw an exception?
   a) try  b) finally  c) throw  d) catch

6. Which of these is a super class of all exceptional type classes?
   a) String b) RuntimeExceptions c) Throwable d) Cachable

7. Which of these class is related to all the exceptions that can be caught by using catch?
   a) Error   b) Exception c) RuntimeException d) All of the mentioned

8. Which of these class is related to all the exceptions that cannot be caught?
   a) Error b) Exception c) RuntimeException d) All of the mentioned

9. Which of these handles the exception when no catch is used?
   a) Default handler  b) finally c) throw handler  d) Java run time system

10. Which of these keywords is used to manually throw an exception?
    a) try  b) finally  c) throw  d) catch

Answers

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>
Fill in the blanks question with answers

11. ________these keywords is used to generate an exception explicitly
12. ________ class is related to all the exceptions that are explicitly thrown
13. New operator is used to generate an ________ of an exception than can be thrown by using throw?
14._______ handler handles the exception when no catch is used?
15. ________ keywords is used to by the calling function to guard against the exception that is thrown by called function?
16. ________ is a process in which two or more parts of same process run simultaneously.
17. ________ are types of multitasking?
18. ________ these packages contain all the Java’s built in exceptions?
19. Thread priority in Java is ____________
20. If two thread of same priority are called to be processed simultaneously It is - __ will happen.

Answers

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>throw</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>Throwable</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>instance</td>
<td>18</td>
</tr>
<tr>
<td>14</td>
<td>Default</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>throws</td>
<td>10</td>
</tr>
</tbody>
</table>

Unit IV

2 marks

1. Identify Collection Interfaces?
Ans: The collection interfaces are divided into two groups. The most basic interface, java.util.Collection, has the following descendants:

- java.util.Set
- java.util.SortedSet
- java.utilNavigableSet
- java.util.Queue
- java.util.concurrent.BlockingQueue
- java.util.concurrent.TransferQueue
- java.util.Deque
- java.util.concurrent.BlockingDeque

2. Define Collection Interfaces

Ans: The Java collections framework (JCF) is a set of classes and interfaces that implement commonly reusable collection data structures. The JCF provides both interfaces that define various collections and classes that implement them.

3. What is framework in java

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

4. What is Collection framework

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

   Interfaces and its implementations i.e. classes
   Algorithm

5. Write the Methods of Iterator interface

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

<table>
<thead>
<tr>
<th>No.</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>public boolean hasNext()</td>
<td>It returns true if iterator has more elements.</td>
</tr>
<tr>
<td>2</td>
<td>public Object next()</td>
<td>It returns the element and moves the cursor pointer to the next element.</td>
</tr>
</tbody>
</table>
3. public void remove()
   It removes the last elements returned by the iterator.
   It is rarely used.

Three marks

1. What are the primary advantages of a collections framework?
   Ans: Reduces programming effort by providing data structures and algorithms so you don’t have to write them yourself.
   Increases performance by providing high-performance implementations of data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be tuned by switching implementations.
   Provides interoperability between unrelated APIs by establishing a common language to pass collections back and forth.
   Reduces the effort required to learn APIs by requiring you to learn multiple ad hoc collection APIs.
   Reduces the effort required to design and implement APIs by not requiring you to produce ad hoc collections APIs.
   Fosters software reuse by providing a standard interface for collections and algorithms with which to manipulate them.

2. What are the collections framework consists of?
   Ans: Collection interfaces. Represent different types of collections, such as sets, lists, and maps. These interfaces form the basis of the framework.
   General-purpose implementations. Primary implementations of the collection interfaces.
   Legacy implementations. The collection classes from earlier releases, Vector and Hashtable, were retrofitted to implement the collection interfaces.
   Special-purpose implementations. Implementations designed for use in special situations. These implementations display nonstandard performance characteristics, usage restrictions, or behavior.
   Concurrent implementations. Implementations designed for highly concurrent use.
   Wrapper implementations. Add functionality, such as synchronization, to other implementations.
   Convenience implementations. High-performance "mini-implmentations" of the collection interfaces.
Abstract implementations. Partial implementations of the collection interfaces to facilitate custom implementations.

Algorithms. Static methods that perform useful functions on collections, such as sorting a list.

Infrastructure. Interfaces that provide essential support for the collection interfaces.

Array Utilities. Utility functions for arrays of primitive types and reference objects. Not, strictly speaking, a part of the collections framework, this feature was added to the Java platform at the same time as the collections framework and relies on some of the same infrastructure.

3. What is the difference between ArrayList and LinkedList classes in collection framework?

Ans: ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes. But there are many differences between ArrayList and LinkedList classes that are given below.

<table>
<thead>
<tr>
<th>ArrayList</th>
<th>LinkedList</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) ArrayList internally uses <strong>dynamic array</strong> to store the elements.</td>
<td>LinkedList internally uses <strong>doubly linked list</strong> to store the elements.</td>
</tr>
<tr>
<td>2) Manipulation with ArrayList is <strong>slow</strong> because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.</td>
<td>Manipulation with LinkedList is <strong>faster</strong> than ArrayList because it uses doubly linked list so no bit shifting is required in memory.</td>
</tr>
<tr>
<td>3) ArrayList class can <strong>act as a list</strong> only because it implements List only.</td>
<td>LinkedList class can <strong>act as a list and queue</strong> both because it implements List and Deque interfaces.</td>
</tr>
<tr>
<td>4) ArrayList is <strong>better for storing and accessing</strong> data.</td>
<td>LinkedList is <strong>better for manipulating</strong> data.</td>
</tr>
</tbody>
</table>
4. What is the difference between ArrayList and Vector classes in collection framework?

Ans: ArrayList and Vector both implements List interface and maintains insertion order. But there are many differences between ArrayList and Vector classes that are given below.

<table>
<thead>
<tr>
<th>ArrayList</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) ArrayList is <strong>not synchronized</strong>.</td>
<td>Vector is <strong>synchronized</strong>.</td>
</tr>
<tr>
<td>2) ArrayList <strong>increments 50%</strong> of current array size if number of element exceeds from its capacity.</td>
<td>Vector <strong>increments 100%</strong> means doubles the array size if total number of element exceeds than its capacity.</td>
</tr>
<tr>
<td>3) ArrayList is <strong>not a legacy class</strong>, it is introduced in JDK 1.2.</td>
<td>Vector is a <strong>legacy class</strong>.</td>
</tr>
<tr>
<td>4) ArrayList is <strong>fast</strong> because it is non-synchronized.</td>
<td>Vector is <strong>slow</strong> because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object.</td>
</tr>
<tr>
<td>5) ArrayList uses <strong>Iterator</strong> interface to traverse the elements.</td>
<td>Vector uses <strong>Enumeration</strong> interface to traverse the elements. But it can use Iterator also.</td>
</tr>
</tbody>
</table>

5. What is the difference between HashSet and HashMap classes in collection framework?

Ans:

<table>
<thead>
<tr>
<th>HASHSET</th>
<th>HASHMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>HashSet class implements the Set interface</td>
<td>HashMap class implements the Map interface</td>
</tr>
<tr>
<td>In HashSet we store objects(elements or objects)</td>
<td>HashMap is used for storing key &amp; value pairs. In short</td>
</tr>
</tbody>
</table>

M.Rajkumar, Assistant Professor, Computer Science and Engineering
values) e.g. If we have a HashSet of string elements then it could depict a set of HashSet elements: \{“Hello”, “Hi”, “Bye”, “Run”\} it maintains the mapping of key & value (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This is how you could represent HashMap elements if it has integer key and value of String type: e.g. \{1->”Hello”, 2->”Hi”, 3->”Bye”, 4->”Run”\}

<table>
<thead>
<tr>
<th>HashSet does not allow duplicate elements that means you can not store duplicate values in HashSet.</th>
<th>HashMap does not allow duplicate keys however it allows to have duplicate values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HashSet permits to have a single null value.</td>
<td>HashMap permits single null key and any number of null values.</td>
</tr>
</tbody>
</table>

**Five marks**

1. Hierarchy of Collection Framework
2. What is the difference between Iterator and Enumeration interface in collection framework?

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Iterator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using <em>Enumeration</em>, you can only traverse the collection. You can’t do any modifications to collection while traversing it.</td>
<td>Using <em>Iterator</em>, you can remove an element of the collection while traversing it.</td>
</tr>
<tr>
<td><em>Enumeration</em> is introduced in JDK 1.0</td>
<td><em>Iterator</em> is introduced from JDK 1.2</td>
</tr>
<tr>
<td><em>Enumeration</em> is used to traverse the legacy classes like Vector, Stack and HashTable.</td>
<td><em>Iterator</em> is used to iterate most of the classes in the collection framework like ArrayList, HashSet, HashMap, LinkedList etc.</td>
</tr>
</tbody>
</table>
Methods:
- `hasMoreElements()` and `nextElement()`

Methods:
- `hasNext()`, `next()` and `remove()`

*Enumeration* is fail-safe in nature.
*Enumeration* is not safe and secured due to it’s fail-safe nature.

*Iterator* is fail-fast in nature.
*Iterator* is safer and secured than *Enumeration*.

3. What is the difference between Comparable and Comparator interfaces?

<table>
<thead>
<tr>
<th>Comparable</th>
<th>Comparator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Comparable provides <strong>single sorting sequence</strong>. In other words, we can sort the collection on the basis of single element such as id or name or price etc.</td>
<td>Comparator provides <strong>multiple sorting sequence</strong>. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc.</td>
</tr>
<tr>
<td>2) Comparable <strong>affects the original class</strong> i.e. actual class is modified.</td>
<td>Comparator <strong>doesn’t affect the original class</strong> i.e. actual class is not modified.</td>
</tr>
<tr>
<td>3) Comparable provides <code>compareTo()</code> method to sort elements.</td>
<td>Comparator provides <code>compare()</code> method to sort elements.</td>
</tr>
<tr>
<td>4) Comparable is found in <code>java.lang</code> package.</td>
<td>Comparator is found in <code>java.util</code> package.</td>
</tr>
<tr>
<td>5) We can sort the list elements of Comparable type by <code>Collections.sort(List)</code> method.</td>
<td>We can sort the list elements of Comparator type by <code>Collections.sort(List,Comparator)</code> method.</td>
</tr>
</tbody>
</table>

4. What is the difference between java collection and java collections?
- Major difference between Collection and Collections is Collection is an interface and Collections is a class.
- Both are belongs to java.util package.
• Collection is base interface for list set and queue.
• Collections is a class and it is called utility class.
• Collections utility class contains some predefined methods so that we can use while working with Collection type of classes (treeset, arraylist, linkedlist etc.)
• Collection is base interface for List, Set and Queue.
• Collection is a root level interface of the Java Collection Framework. Most of the classes in Java Collection Framework inherit from this interface. List, Set and Queue are main sub interfaces of this interface.
• Collections is an utility class in java.util package. It consists of only static methods which are used to operate on objects of type Collection. For example, it has the method to find the maximum element in a collection, it has the method to sort the collection, it has the method to search for a particular element in a collection.
• JDK doesn’t provide any direct implementations of this interface. But, JDK provides direct implementations of it’s sub interfaces. ArrayList, Vector, HashSet, LinkedHashSet, PriorityQueue are some indirect implementations of Collection interface. Map interface, which is also a part of java collection framework, doesn’t inherit from Collection interface. Collection interface is a member of java.util package.

5. **Explain Legacy Classes**

**Ans:** Early version of java did not include the Collections framework. It only defined several classes and interfaces that provide methods for storing objects. When Collections framework were added in J2SE 1.2, the original classes were reengineered to support the collection interface. These classes are also known as Legacy classes. All legacy classes and interface were redesign by JDK 5 to support Generics. In general, the legacy classes are supported because there is still some code that uses them.

The following are the legacy classes defined by java.util package

1. Dictionary
2. HashTable
3. Properties
4. Stack
5. Vector

There is only one legacy interface called **Enumeration**
Objective Type Question with answers

1. Which of these packages contain all the collection classes? ( )
   a) java.lang b) java.util c) java.net d) java.awt

2. Which of these classes is not part of Java’s collection framework? ( )
   a) Maps b) Array c) Stack d) Queue

3. Which of these interface is not a part of Java’s collection framework? ( )
   a) List b) Set c) SortedMap d) SortedList

4. Which of these methods deletes all the elements from invoking collection? ( )
   a) clear() b) reset() c) delete() d) refresh()

5. What is Collection in Java? ( )
   a) A group of objects   b) A group of classes
   c) A group of interfaces d) None of the mentioned

6. Why are generics used? ( )
   a) Generics make code more fast. b) Generics make code more optimised and readable.
   c) Generics add stability to your code by making more of your bugs detectable at compile
time.
   d) Generics add stability to your code by making more of your bugs detectable at run time.

7. Which of these type parameters is used for a generic class to return and accept any type of
   object? ( )
   a) K b) N c) T d) V

8. Which of these type parameters is used for a generic class to return and accept a number? ( )
   a) K b) N c) T d) V

9. Which of these is an correct way of defining generic class? ( )
   a) class name(T1, T2, ..., Tn) { /* ... */ } b) class name { /* ... */
   c) class name(T1, T2, ..., Tn) { /* ... */ } d) class name(T1, T2, ..., Tn) { /* ... */

10. Which of the following is incorrect statement regarding the use of generics and
    parameterized types in Java? ( )
    a) Generics provide type safety by shifting more type checking responsibilities to the
b) Generics and parameterized types eliminate the need for down casts when using Java Collections.

c) When designing your own collections class (say, a linked list), generics and parameterized types allow you to achieve type safety with just a single class definition as opposed to defining multiple classes.

d) All of the mentioned

Answers

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>6</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>7</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>8</td>
<td>b</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>9</td>
<td>b</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>10</td>
<td>c</td>
</tr>
</tbody>
</table>

**Fill in the blanks question with answers**

11. _______class reference types cannot be generic

12. _______class contains the methods used to write in a file?

13. _______ exception is thrown in cases when the file specified for writing it not found.

14. _______methods are used to read in from file.

15. _______values is returned by read() method is end of file (EOF) is encountered.

16. _______is thrown by close() and read() methods.

17. _______methods is used to write into a file.

18. _______ is used to access the database server at time of executing the program and get the data from the server accordingly

19. _______header must be included in java program to establish database connectivity using JDBC ?

20. DriverManager.getConnection ________________

Answers

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Anonymous innerclass</td>
<td>16</td>
<td>IO exception</td>
</tr>
</tbody>
</table>
Unit IV

Two marks:

1. Explain limitations of AWT?
Ans: Limitations of AWT: The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents or peers.

2. Advantage of Model 2 (MVC) Architecture
Ans:
- **Navigation control is centralized** Now only controller contains the logic to determine the next page.
- **Easy to maintain**
- **Easy to extend**
- **Easy to test**
- **Better separation of concerns**

3. Disadvantage of Model 2 (MVC) Architecture
Ans:
- We need to write the controller code self. If we change the controller code, we need to recompile the class and redeploy the application.

4. Disadvantage of Model 1 Architecture
Ans:
**Navigation control is decentralized** since every page contains the logic to determine the next page. If JSP page name is changed that is referred by other pages, we need to change it in all the pages that leads to the maintenance problem.
Time consuming You need to spend more time to develop custom tags in JSP. So that we don't need to use scriptlet tag.

Hard to extend It is better for small applications but not for large applications.

5. Write Java Event Handling Code

Ans: We can put the event handling code into one of the following places:
- Within class
- Other class
- Anonymous class

Three marks:

1. Write Java BorderLayout in brief?

Ans: The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

   public static final int NORTH
   public static final int SOUTH
   public static final int EAST
   public static final int WEST
   public static final int CENTER

2. Write different Constructors of BorderLayout class:

Ans:
- BorderLayout(): creates a border layout but with no gaps between the components.
- JBorderLayout(int hgap, int vgap): creates a border layout with the given horizontal and vertical gaps between the components.

3. Write different Java LayoutManagers

Ans: The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

   java.awt.BorderLayout
   java.awt.FlowLayout
   java.awt.GridLayout
   java.awt.CardLayout
4. Write Java Anonymous inner class?

**Ans:** A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

- Class (may be abstract or concrete).
- Interface

Java anonymous inner class example using class

```java
abstract class Person{
    abstract void eat();
}

class TestAnonymousInner{
    public static void main(String args[]){
        Person p=new Person(){
            void eat(){System.out.println("nice fruits");}
        };
        p.eat();
    }
}
```

Java anonymous inner class example using interface

```java
interface Eatable{
    void eat();
}
class TestAnonymousInner1{
    public static void main(String args[]){
        Eatable e=new Eatable(){
```
Five marks:

1. **Explain** Model 1 and Model 2 (MVC) Architecture

**Ans:**

Model 1 Architecture
Servlet and JSP are the main technologies to develop the web applications. **Servlet** was considered superior to CGI. Servlet technology doesn't create process, rather it creates thread to handle request. The advantage of creating thread over process is that it doesn't allocate separate memory area. Thus many subsequent requests can be easily handled by servlet.

**Problem in Servlet technology** Servlet needs to recompile if any designing code is modified. It doesn't provide separation of concern. Presentation and Business logic are mixed up.

**JSP** overcomes almost all the problems of Servlet. It provides better separation of concern, now presentation and business logic can be easily separated. You don't need to redeploy the application if JSP page is modified. JSP provides support to develop web application using JavaBean, custom tags and JSTL so that we can put the business logic separate from our JSP that will be easier to test and debug.

As you can see in the above figure, there is picture which show the flow of the model1 architecture.

1. Browser sends request for the JSP page
2. JSP accesses Java Bean and invokes business logic
3. Java Bean connects to the database and get/save data
4. Response is sent to the browser which is generated by JSP

Advantage of Model 1 Architecture
- Easy and Quick to develop web application

Disadvantage of Model 1 Architecture
- **Navigation control is decentralized** since every page contains the logic to determine the next page. If JSP page name is changed that is referred by other pages, we need to change it in all the pages that leads to the maintenance problem.
- **Time consuming** You need to spend more time to develop custom tags in JSP. So that we don't need to use scriptlet tag.
- **Hard to extend** It is better for small applications but not for large applications.

Model 2 (MVC) Architecture
Model 2 is based on the MVC (Model View Controller) design pattern. The MVC design pattern consists of three modules model, view and controller.

**Model** The model represents the state (data) and business logic of the application.

**View** The view module is responsible to display data i.e. it represents the presentation.

**Controller** The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.

2. Write Key Differences Between Applet and Application

**Ans:** Let's understand the difference between applet and application through the given points:
Applets are not entirely featured application programs and are usually written to achieve a small task or part of it. On the other hand, an Application is a program which runs on an underlying operating system. These are generic in a sense and designed to perform a specific task directly for the user.

An applet does not use the main() method. Instead, it automatically calls defined methods after loading, for Applet class to start and execute the applet code. On the contrary, Application uses main() method for initiating the execution of the code.

Unlike a stand-alone application, applets cannot be run independently. They are run from inside a web page using a special feature known as HTML tag. Applets cannot write to and read from the files in the local computer. While Application can perform such operation to the files in the local computer.

Prior installation is not required in an applet. As against, Prior explicit installation is necessarily needed while using an Application.

Restrictions are imposed on applets for using libraries from other languages and also local files. Whereas Application can access libraries as well as local files.

An application can run several programs from the local computer. In contrast, Applets can not do so.

3. Difference between AWT and Swing

Ans: There are many differences between java awt and swing that are given below.

<table>
<thead>
<tr>
<th>No.</th>
<th>Java AWT</th>
<th>Java Swing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>AWT components are platform-dependent.</td>
<td>Java swing components are platform-independent.</td>
</tr>
<tr>
<td>2)</td>
<td>AWT components are heavyweight.</td>
<td>Swing components are lightweight.</td>
</tr>
<tr>
<td>3)</td>
<td>AWT doesn't support pluggable look and feel.</td>
<td>Swing supports pluggable look and feel.</td>
</tr>
<tr>
<td>4)</td>
<td>AWT provides less components than Swing.</td>
<td>Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.</td>
</tr>
</tbody>
</table>
5) AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view. Swing follows MVC.

4. Hierarchy of Java Swing classes
   
   Ans:

   \[
   \text{import javax.swing.*;} \\
   \text{public class Simple} \{ \\
   \text{JFrame f;} \\
   \text{Simple()}{ \\
   \text{f=new JFrame();//creating instance of JFrame}
   \}
   \]

5. Write Example of Swing by Association inside constructor
   
   Ans:
OBJECT ORIENTED PROGRAMMING THROUGH JAVA

```java
JButton b = new JButton("click"); // creating instance of JButton
b.setBounds(130, 100, 100, 40);

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

public static void main(String[] args) {
    new Simple();
}
```

**Objective type question with answers**

1) Give the abbreviation of AWT?
   ( )
   A) Applet Windowing Toolkit      B) Abstract Windowing Toolkit
   C) Absolute Windowing Toolkit    D) None of the above

2) Which is the container that contain title bar and can have MenuBars. It can have other components like button, textfield etc.?
   ( )
   A) Panel  B) Frame  C) Window  D) Container

3) Which is a component in AWT that can contain another components like buttons, textfields, labels etc.?
   ( )
   A) Window  B) Container  C) Panel  D) Frame

4) AWT is used for GUI programming in java?
   ( )
   A) True  B) False

5) The following specifies the advantages of
It is lightweight.
It supports pluggable look and feel.
It follows MVC (Model View Controller) architecture.
   ( )
   A) Swing  B) AWT  C) Both A & B  D) None of the above
6) Swing is not a part of JFC (Java Foundation Classes) that is used to create GUI application?
   A) True    B) False

7) Which class provides many methods for graphics programming?
   A) java.awt    B) java.Graphics    C) java.awt.Graphics    D) None of the above

8) These two ways are used to create a Frame
   By creating the object of Frame class (association)
   By extending Frame class (inheritance)
   A) True    B) False

9) The Following steps are required to perform
   1) Implement the Listener interface and overrides its methods
   2) Register the component with the Listener
   A) Exception Handling    B) String Handling    C) Event Handling    D) None of the above

10) In which places can put the event handling code
    A) Same class    B) Other class    C) Anonymouse class    D) All mentioned above

Answers

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>10</td>
</tr>
</tbody>
</table>

Fill in the blanks question with answers

11) ____________package provides many event classes and Listener interfaces for event handling.

12) To use the ActionListener interface it must be implemented by a class ________.

13) The ActionListener interface is not used for ____________

14) The ________ ActionListener interface is used for handling action events.

15) ____________is the container that doesn't contain title bar and MenuBars.

16) Functions is called to display the output of an applet __________

17) ____________ methods can be used to output a sting in an applet.
18. __________ methods is a part of Abstract Window Toolkit (AWT)
19. Modifiers can be used for a variable so that it can be accessed from any thread or parts of a program __________
20. __________ operators can be used to get run time information about an object.

Answers:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Java.awt.event</td>
<td>16</td>
<td>Print(0</td>
</tr>
<tr>
<td>12</td>
<td>Listener interface</td>
<td>17</td>
<td>drawstring</td>
</tr>
<tr>
<td>13</td>
<td>Action listener</td>
<td>18</td>
<td>display</td>
</tr>
<tr>
<td>14</td>
<td>JButtonActionListener</td>
<td>19</td>
<td>volatile</td>
</tr>
<tr>
<td>15</td>
<td>panel</td>
<td>20</td>
<td>Instanceof</td>
</tr>
</tbody>
</table>
17. **Beyond syllabus Topics with material**

**Jdbc drivers & Types of jdbc drivers**

**What is JDBC Driver?**

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

The `java.sql` package that ships with JDK, contains various classes with their behaviours defined and their actual implementations are done in third-party drivers. Third party vendors implements the `java.sql.Driver` interface in their database driver.

**JDBC Drivers Types**

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below –

**Type 1: JDBC-ODBC Bridge Driver**

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.
The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Type 2: JDBC-Native API

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC’s overhead.

The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Type 3: JDBC-Net pure Java

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.
You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

Type 4: 100% Pure Java

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.
Which Driver should be Used?

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.

If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.

Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

---

**Connecting JDBC TO DATABASE**

After you've installed the appropriate driver, it is time to establish a database connection using JDBC.

The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps –

- **Import JDBC Packages**: Add `import` statements to your Java program to import required classes in your Java code.

- **Register JDBC Driver**: This step causes the JVM to load the desired driver implementation into memory so it can fulfill your JDBC requests.

- **Database URL Formulation**: This is to create a properly formatted address that points to the database to which you wish to connect.

- **Create Connection Object**: Finally, code a call to the `DriverManager` object's `getConnection()` method to establish actual database connection.

Import JDBC Packages

The `import` statements tell the Java compiler where to find the classes you reference in your code and are placed at the very beginning of your source code.

To use the standard JDBC package, which allows you to select, insert, update, and delete data in SQL tables, add the following `imports` to your source code –
Register JDBC Driver

You must register the driver in your program before you use it. Registering the driver is the process by which the Oracle driver's class file is loaded into the memory, so it can be utilized as an implementation of the JDBC interfaces.

You need to do this registration only once in your program. You can register a driver in one of two ways.

Approach I - Class.forName()

The most common approach to register a driver is to use Java's `Class.forName()` method, to dynamically load the driver's class file into memory, which automatically registers it. This method is preferable because it allows you to make the driver registration configurable and portable.

The following example uses `Class.forName()` to register the Oracle driver –

```java
try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
}
```

```java
} catch(ClassNotFoundException ex) {
    System.out.println("Error: unable to load driver class!");
    System.exit(1);
}
```

You can use `getInstance()` method to work around noncompliant JVMs, but then you'll have to code for two extra Exceptions as follows –

```java
try {
    Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
}
```

```java
} catch(ClassNotFoundException ex) {
    System.out.println("Error: unable to load driver class!");
}
```
System.exit(1);

try {
    Driver myDriver = new oracle.jdbc.driver.OracleDriver();
    DriverManager.registerDriver( myDriver );
} catch(ClassNotFoundException ex) {
    System.out.println("Error: unable to load driver class!");
    System.exit(1);
}

Approach II - DriverManager.registerDriver()

The second approach you can use to register a driver, is to use the static\texttt{DriverManager.registerDriver()} method.

You should use the \texttt{registerDriver()} method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.

The following example uses registerDriver() to register the Oracle driver –

Database URL Formulation

After you've loaded the driver, you can establish a connection using the\texttt{DriverManager.getConnection()} method. For easy reference, let me list the three overloaded \texttt{DriverManager.getConnection()} methods –

- \texttt{getConnection(String url)}
- \texttt{getConnection(String url, Properties prop)}
getConnection(String url, String user, String password)

Here each form requires a database **URL**. A database URL is an address that points to your database.

Formulating a database URL is where most of the problems associated with establishing a connection occurs.

Following table lists down the popular JDBC driver names and database URL:

<table>
<thead>
<tr>
<th>RDBMS</th>
<th>JDBC driver name</th>
<th>URL format</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>com.mysql.jdbc.Driver</td>
<td>jdbc:mysql://hostname/databaseName</td>
</tr>
<tr>
<td>ORACLE</td>
<td>oracle.jdbc.driver.OracleDriver</td>
<td>jdbc:oracle:thin:@hostname:port Number:databaseName</td>
</tr>
<tr>
<td>DB2</td>
<td>COM.ibm.db2.jdbc.net.DBDriver</td>
<td>jdbc:db2:hostname:port Number/databaseName</td>
</tr>
<tr>
<td>Sybase</td>
<td>com.sybase.jdbc.SybDriver</td>
<td>jdbc:sybase:tds:hostname: port Number/databaseName</td>
</tr>
</tbody>
</table>

All the highlighted part in URL format is static and you need to change only the remaining part as per your database setup.

Create Connection Object

We have listed down three forms of `DriverManager.getConnection()` method to create a connection object.

Using a Database URL with a username and password

The most commonly used form of `getConnection()` requires you to pass a database URL, a *username*, and a *password*:

Assuming you are using Oracle's **thin** driver, you'll specify a host:port:databaseName value for the database portion of the URL.

If you have a host at TCP/IP address 192.0.0.1 with a host name of amrood, and your Oracle listener is configured to listen on port 1521, and your database name is EMP, then complete database URL would be –
Now you have to call `getConnection()` method with appropriate username and password to get a `Connection` object as follows –

```java
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
String USER = "username";
String PASS = "password"
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

Using Only a Database URL

A second form of the `DriverManager.getConnection()` method requires only a database URL –

```java
DriverManager.getConnection(String url);
```

However, in this case, the database URL includes the username and password and has the following general form –

```java
jdbc:oracle:driver:username/password@database
```

So, the above connection can be created as follows –

```java
String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";
Connection conn = DriverManager.getConnection(URL);
```

Using a Database URL and a `Properties` Object

A third form of the `DriverManager.getConnection()` method requires a database URL and a `Properties` object –

```java
DriverManager.getConnection(String url, Properties info);
```

A `Properties` object holds a set of keyword-value pairs. It is used to pass driver properties to the driver during a call to the `getConnection()` method.

To make the same connection made by the previous examples, use the following code –

```java
import java.util.*;
```
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";

Properties info = new Properties();
info.put( "user", "username" );
info.put( "password", "password" );

Connection conn = DriverManager.getConnection(URL, info);

Closing JDBC Connections

At the end of your JDBC program, it is required explicitly to close all the connections to the database to end each database session. However, if you forget, Java's garbage collector will close the connection when it cleans up stale objects.

Relying on the garbage collection, especially in database programming, is a very poor programming practice. You should make a habit of always closing the connection with the close() method associated with connection object.

To ensure that a connection is closed, you could provide a 'finally' block in your code. A finally block always executes, regardless of an exception occurs or not.

To close the above opened connection, you should call close() method as follows –

conn.close();

Explicitly closing a connection conserves DBMS resources, which will make your database administrator happy.

Accesing online Query
Servlets:

Servlet technology is used to create web application (resides at server side and generates dynamic web page).

Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet,ServletRequest,ServletResponse etc.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.
A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

Disadvantages of CGI

There are many problems in CGI technology:

1. If number of clients increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantage of Servlet

There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

1. **Better performance**: because it creates a thread for each request not process.
2. **Portability**: because it uses java language.
3. **Robust**: Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
4. **Secure**: because it uses java language

JSP
JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc.

A JSP page consists of HTML tags and JSP tags. The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc.

Advantage of JSP over Servlet

There are many advantages of JSP over servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Life cycle of a JSP Page

The JSP pages follows these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (class file is loaded by the classloader)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Request processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).