UNIT4	
1	Define Dynamic Method Dispatch
2	What is interface?
3	Write the syntax to create and import a package
4	Write a Java Program to explain super keyword.
5	Explain Super keyword in inheritance
6	Explain Abstract Keyword with example.
7	Differentiate Abstract class and Interface.
8	List out the steps to create a package
9	List types of Inheritance in Java.
10	Define Package in Java.(same as Q3)
11	Explain multilevel inheritance with example.
12	Explain interface in Java (same as Q2)
13	Write a Java program to demonstrate multilevel Inheritance
14	Write a Java program to demonstrate multiple Inheritances using Interface.
15	Difference: class and interface
16	Explain how abstract class differs from final class
17	Explain multilevel inheritance and hierarchical inheritance with examp
18	Write a program in Java to demonstrate implementation of multiple inheritance using interfaces.(same as 14)
19	Explain packages(same as Q3)
20	Describe abstract class.
21	Write the importance of super keyword in Java.(same as Q5)
22	State the Java does not support multiple inheritance.
23	Differentiate abstract class and final class(same as Q16)
24	What is Package ? Write the steps to create a Package with example.(same as Q3)
25	What is Interface ? Discuss the differences and similarities between Class and Interface.(same as Q15)
26	Explain the types of inheritance with example.
27	Define package and interface.

28	Explain how abstract class differs from final class.(same as Q16)
29	Write the steps to create user defined package.(same as Q8)
30	Write a program in Java in which a subclass constructor invokes the constructor of super class and instantiates the values.
31	List different types of inheritance and explain any one with example.(same as Q26)
32	Explain with example how multiple inheritance can be implemented by interface.(same as Q14)
33	Define interface in Java.(same as Q2)
34	Explain how to implement multiple inheritances in java through interface?(same as Q14)
35	Write a program in Java to demonstrate implementation of multiple inheritance using interfaces.(same as Q14)
36	State the name of any four inbuilt Java package.
37	List four types of Inheritance.(same as Q9)
38	Write an application that illustrates method overriding in the different packages.
39	ACCESS MODIFIER IN JAVA
40	Explain JAVA ENCAPSULATION

UNIT 5	
1	Explain throw and finally in Exception Handling of java language.
2	Write a java program to write a text file in java.
3	Write a Java Program to develop user defined exception.
4	Explain Synchronization in Thread.
5	Compare checked and Unchecked Exception.
6	Define Thread in Java.
7	Explain 'try' and 'catch' statements in exceptional handling of Java language
8	Explain 'Extending Thread class' in Java.
9	Explain life cycle of thread in Java.
10	List types of Errors in exceptional handling and explain any one of them.
11	Write a Java program that executes three threads. One thread displays "Thread – I" every 2500 millisecond, second thread displays "Thread – II" every 5000 millisecond and third thread displays "Thread – III" every 7500 millisecond.
12	List types of error in Java

Q1) Define Dynamic Method Dispatch

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

Example

```
class Bike{
 void run(){System.out.println("running");}
       }
      class Splendor extends Bike{
       void run(){System.out.println("running safely with 60km");}
        public static void main(String args[]){
        Bike b = new Splendor();//upcasting
        b.run();
       }
      }
```

Output:

```
running safely with 60km.
```

Q2) What is interface?

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.
The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract ethods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Why use Java interface?

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

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

Syntax:

```
interface <interface_name>{

 // declare constant fields
 // declare methods that abstract
 // by default.
}
```

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
}
}
```

OUTPUT: Hello

Q3) Write a Java Program to explain super keyword.

The super keyword in Java is a reference variable which is used to refer immediate parent class object.
Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

```
class Animal{
 String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

Output:

```
black
white
```

Q4) Explain Super keyword in inheritance

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

OUTPUT

```
eating...
barking...
```

Q5) Explain Abstract Keyword with example.

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body). It needs to be extended and its method implemented. It cannot be instantiated.

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

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

OUTPUT

```
running safely
```

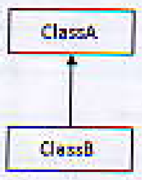
Q6) Differentiate Abstract class and Interface.

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract**methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance.**	Interface **supports multiple inheritance.**
3) Abstract class **can have final, non-final, static and non-static variables.**	Interface has **only static and** final variables
4) Abstract class **can provide the implementation of interface.**	Interface **can't provide the implementatio of abstract class.**
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

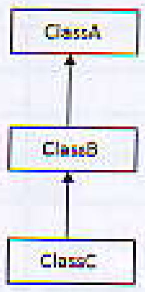
9)**Example:**
public abstract class Shape{
public abstract void draw();}
	Example:
	public interface Drawable{
	void draw();}

Q7) List types of Inheritance in Java.

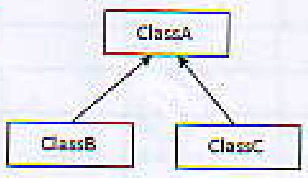
1) Single level
2) Multilevel
3) Hierarchical
4) Multiple
5) Hybrid



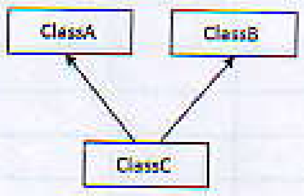
1) Single



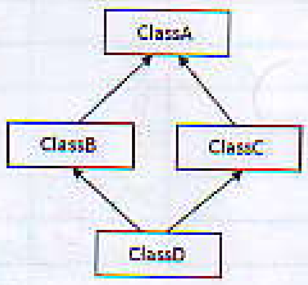
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

Q8) Explain multilevel inheritance with example.

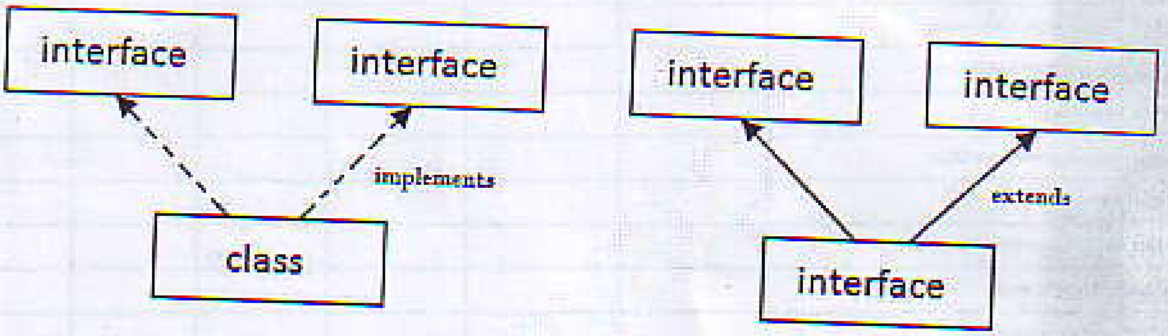
```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();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

OUTPUT:

```
weeping
barking...
eating...
```

Q9) Write a Java program to demonstrate multiple Inheritances using Interface.

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance



Multiple Inheritance in Java

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
}
}
```

OUTPUT

```
Hello
Welcome
```

Q10) Explain multilevel inheritance and hierarchical inheritance with example

Multilevel Inheritance Example

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

OUTPUT:

```
weeping
barking...
eating...
```

Hierarchical Inheritance Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
}}
```

Output:

```
meowing...
eating.
```

Q11) Describe abstract class.

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body). It needs to be extended and its method implemented. It cannot be instantiated.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)

2 Interface (100%)

Example of Abstract class that has an abstract method

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

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

<u>OUTPUT</u>

```
running safely
```

Q12) State the Java does not support multiple inheritance.

Multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class. For example:

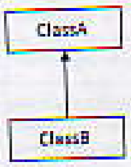
```
interface Printable{
void print();
}
interface Showable{
void print();
}

class TestInterface3 implements Printable, Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestInterface3 obj = new TestInterface3();
obj.print();
} }
```

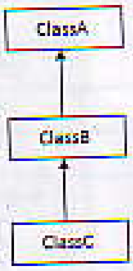
Output: Hello

Q13) Explain the types of inheritance with example.

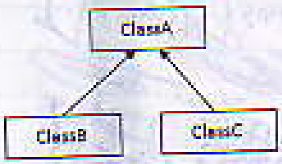
1) Single level
2) Multilevel
3) Hierarchical
4) Multiple
5) Hybrid



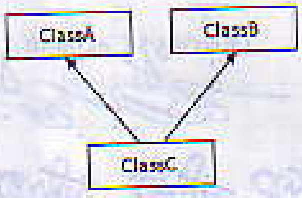
1) Single



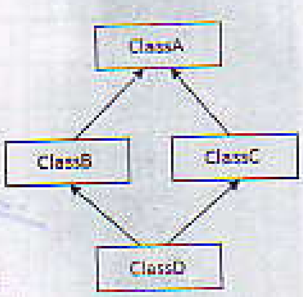
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

Single Inheritance Example

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

OUTPUT

```
barking...
eating...
```

[illegible]

Multilevel Inheritance Example

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

OUTPUT:

```
weeping
barking...
eating...
```

Hierarchical Inheritance Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating.
```

Q14) Write a program in Java in which a subclass constructor invokes the constructor of super class and instantiates the values.

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();  }}
```

Output:

```
animal is created
    dog is created
```

Q15) Write the syntax to create and import 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.

Advantage of Java Package
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.

Simple example of java package

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

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

Q16) List out the steps to create a package

1) Package package_name

Or

2) Package package_name.subpackage_name

Q17) Difference: class and interface

CLASS	INTERFACE
Supports only multilevel and hierarchical inheritances but not multiple inheritance	Supports all types of inheritance: multilevel, hierarchical and multiple
"extends" keyword should be used to inherit	"implements" keyword should be used to inherit
Should contain only concrete methods (methods with body)	Should contain only abstract methods (methods without body)
The methods can be of any access specifier (all the four types)	The access specifier must be public only
Methods can be final and static	Methods should not be final and static
Variables can be private	Variables should be public only
Can have constructors	Cannot have constructors
Can have main() method	Cannot have main() method as main() is a concrete method

Q18) Explain how abstract class differs from final class

PROPERTY	ABSTRACT CLASS	FINAL CLASS
SUBCLASSING	Should be subclassed to override the functionality of abstract methods	Can never be subclassed as final does not permit
METHOD ALTERATIONS	Abstract class methods functionality can be altered in subclass	Final class methods should be used as it is by other classes
INSTANTIATION	Cannot be instantiated	Can be instantiated
OVERRIDING CONCEPT	For later use, all the abstract methods should be overridden	Overriding concept does not arise as final class cannot be inherited
INHERITANCE	Can be inherited	Cannot be inherited
ABSTRACT METHODS	Can contain abstract methods	Cannot contain abstract methods
PARTIAL IMPLEMENTATION	A few methods can be implemented and a few cannot	All methods should have implementation
IMMUTABLE OBJECTS	Cannot create immutable objects (infact, no objects can be created)	Immutable objects can be created (eg. String class)
NATURE	It is an incomplete class (for this reason only, designers do not allow to create objects)	It is a complete class (in the sense, all methods have complete functionality or meaning)
ADDING EXTRA FUNCTIONALITY	Extra functionality to the methods can be added in subclass	No extra functionality can be added and should be used as it is

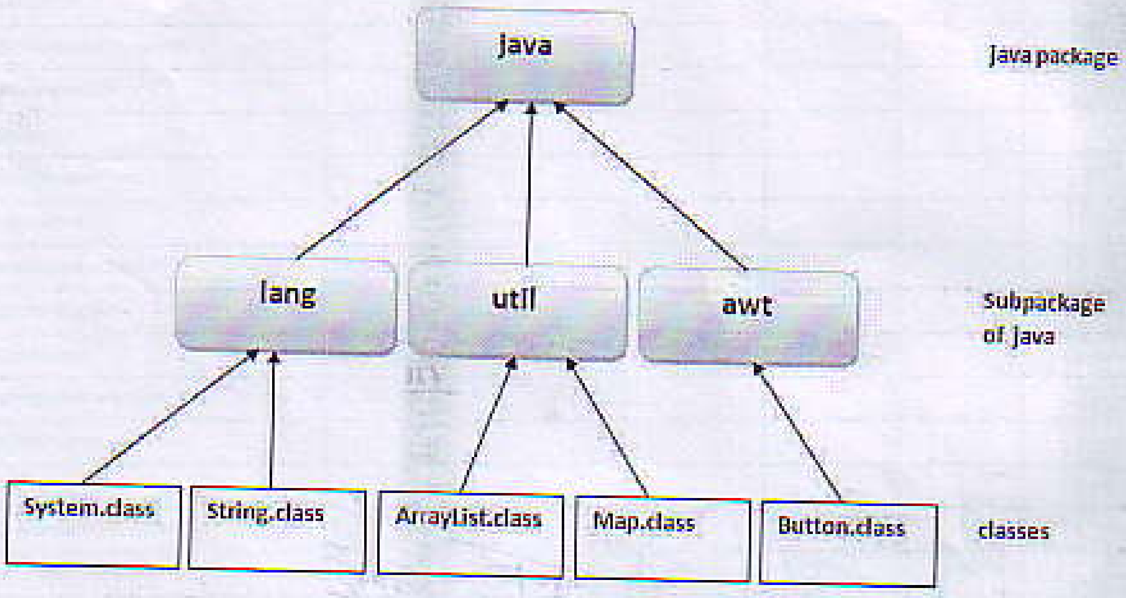
Q19) Define package and interface.

BASIS FOR COMPARISON	PACKAGES	INTERFACES
Basic	Packages is a group of classes and/or interfaces together.	Interfaces is a group of abstract methods and constant fields.
Keyword	Packages are created using "Package" keyword.	Interface are created using "Interface" keyword.

Syntax	package package_name;
	public class class_name{
	.
	(body of class)
	.
	}
		interface interface_name{
		variable declaration;
		method declaration;
		}

Access	A package can be imported	An interface can be extended by another interface and implemented by the class.
Access keyword	Packages can be imported using "import" keyword.	Interfaces can be implemented using "implement" keyword.

Q20) State the name of any four inbuilt Java package.



Q21) Write an application that illustrates method overriding in the different packages.

```
// Filename: Hello.java
package a;
public class Hello {
    private void printMessage()
    {
        System.out.println("Hello");
    }
    public void fun()
    {
        printMessage();
    }
}

// Filename: World.java
package b;
import a.Hello;
public class World extends Hello {
    private void printMessage()
    {
        System.out.println("World");
    }

    public static void main(String[] args)
    {
        Hello gfg = new World();
        gfg.fun();
    }
}
```

Output:

```
Hello
```

Q21) Explain JAVA ENCAPSULATION

Encapsulation in Java is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

Advantage of Encapsulation in Java

By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.

It provides you the **control over the data.** Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

The encapsulate class is **easy to test**. So, it is better for unit testing.

The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

Simple Example of Encapsulation in Java

File: Student.java

```
//A Java class which is a fully encapsulated class.
//It has a private data member and getter and setter methods.
package com.javatpoint;
public class Student{
//private data member
private String name;
//getter method for name
public String getName(){
return name;
}
//setter method for name
public void setName(String name){
this.name=name
}
}
```

File: Test.java

```
//A Java class to test the encapsulated class.
```

```
package com.javatpoint;
class Test{
public static void main(String[] args){
//creating instance of the encapsulated class
Student s=new Student();
//setting value in the name member
s.setName("vijay");
//getting value of the name member
System.out.println(s.getName());
}
}
```

Output:

```
vijays
```

Q22) ACCESS MODIFIER IN JAVA

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

Access Modifier	within class	within package	outside package by subclass only
Private	Y	N	N
Default	Y	Y	N
Protected	Y	Y	Y
Public	Y	Y	Y

Q23) Explain Polymorphism in Java

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

There are two types of polymorphism in Java: **compile-time polymorphism and runtime polymorphism.** We can perform polymorphism in java by method overloading and method overriding.

RUNTIME polymorphism/Dynamic method Dispatch

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

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

Example of upcasting

```
class A{}
class B extends A{}
A a=new B();//upcasting
```

Example of Java Runtime Polymorphism

```
class Bike{
 void run(){System.out.println("running");}
}
class Splendor extends Bike{
 void run(){System.out.println("running safely with 60km");}
  public static void main(String args[]){
  Bike b = new Splendor();//upcasting
  b.run();
 }
}
```

Output:

```
running safely with 60km.
```

Define Package in Java.(same as Q3)
Explain interface in Java (same as Q2)
Write a program in Java to demonstrate implementation of multiple inheritance using interfaces.(same as 14)
Explain packages(same as Q3)
Write the importance of super keyword in Java.(same as Q5)
Differentiate abstract class and final class(same as Q16)
What is Package ? Write the steps to create a Package with example.(same as Q3)
What is Interface ? Discuss the differences and similarities between Class and Interface.(same as Q15)
Explain how abstract class differs from final class.(same as Q16)
Write the steps to create user defined package.(same as Q8)
List different types of inheritance and explain any one with example.(same as Q26)
Explain with example how multiple inheritance can be implemented by interface.(same as Q14)
Define interface in Java.(same as Q2)
Explain how to implement multiple inheritances in java through interface?(same as Q14)
Write a program in Java to demonstrate implementation of multiple inheritance using interfaces.(same as Q14)
State the name of any four inbuilt Java package.
List four types of Inheritance.(same as Q9)