UNIT – II

SYLLABUS

Inheritance
Packages and Interfaces
Strings: String Constructors, Program using 10 String methods
StringBuffer class, Program using 10 StringBuffer methods Introducing StringBuilder class.
Type Wrappers, Auto boxing/unboxing.
Collections: Collections Overview, Names of Collection Interfaces, Classes, Programs using Collection

2.1 Inheritance

Inheritance Basics

- Inheritance is a mechanism to obtaining the one class properties into another class and it creates new class from existing class.

(Or)

- Inheritance is a mechanism to obtaining the one object properties into another object and it creates new class from existing class.
- The class which is giving properties is called **base class** (or) **parent class** (or) **super class**.
- The class which is taking properties is called **derived class** (or) **child class** (or) **sub class**.
- Inheritance is basically used for reducing the overall code size of the program (**code reusability**).

Derived class:

- The class which is taking properties is called **derived class** (or) **child class** (or) **sub class**.
- Syntax:

```
class  Derivedclass-name extends Baseclass-name
{
        //methods and fields
}
```

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

Example: Demonstrate the inheritance

```
class Base
{
    void display( )
    {
            System.out.println("I am base class display");
    }
}
class Derived extends Base
{
    void show( )
    {
            System.out.println("I am derived class show");
    }
}
class InheritanceDemo
{
    public static void main(String args[])
    {
            Derived obj=new Derived();
            obj.display();
            obj.show();
    }
}
```

Output:

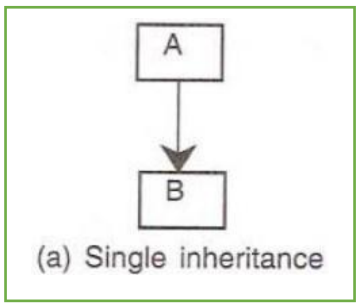
```
I am base class display
I am derived class show
```

UNIT – II

Types of inheritance: there are 5 types of inheritances.

1. **Single inheritance**
2. **Multiple inheritance**
3. **Hierarchical inheritance**
4. **Multilevel inheritance**
5. **Hybrid inheritance**

1. Single inheritance



Single derived class with only one base class, is called single inheritance.

(Or)

A class inherits members of only one class is called single inheritance.

Example program:

```
class  Base
{
       public void add(int x, int y)
      {
                 System.out.println("Addition of x and y :"+(x+y));
      }
}
class  Derived  extends  Base
{
      public void sub(int x, int y)
      {
                 System.out.println("Subtraction of x and y :"+(x-y));
      }
}
class   SingleInheritance
{
      public  static  void main(String args[])
      {
                 public int x = 10 , y = 20;
                 Derived  obj=new  Derived();
                 obj.add(x,y);
                 obj.sub(x,y);
      }
}
```

Output:

```
Addition of x and y :30
Subtraction of x and y :-10
```

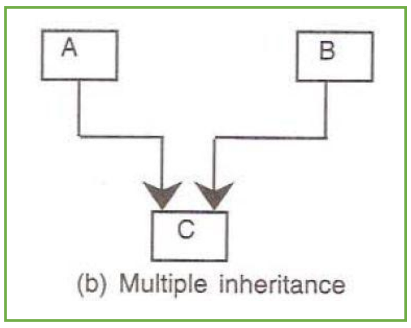
2. Multiple inheritance

A derived class with several base class, is called multiple inheritance.

(Or)

A class inherits members of two or more classes is called multiple inheritance.

Note: Single inheritance achieved through interface, not classes



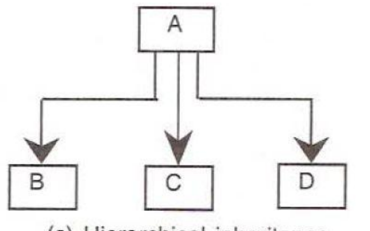
3. Hierarchical inheritance

Several derived classes with only one base class is called hierarchical inheritance.

(Or)

Two or more classes inherits members of only one class is called hierarchical inheritance.

Example program:



(c) Hierarchical inheritance

```
class Base
{
      int x = 10,y = 20;
      public void display()
     {
      System.out.println("x value:"+x);
                  System.out.println("y value:"+y);
     }
}
class Derived1 extends Base
{
      public void Biggest()
     {
      if(x>y)
                  System.out.println("biggest number is:"+x);
       else
                  System.out.println("biggest number is:"+y);
     }
}
class Derived2 extends Base
{
      public void mean()
     {
                  float z = (x + y)/2;
      System.out.println("The mean of the given numbers:"+z);
     }
}
class HierachielInheritance
{
      public static void main(String args[])
     {
                  Derived1 obj1 = new Derived1();
                  obj1.display();
                  obj1.Biggest();

      Derived2 obj2 = new Derived2();
      obj2.display();
      obj2.mean();
   }
}
```

Output:

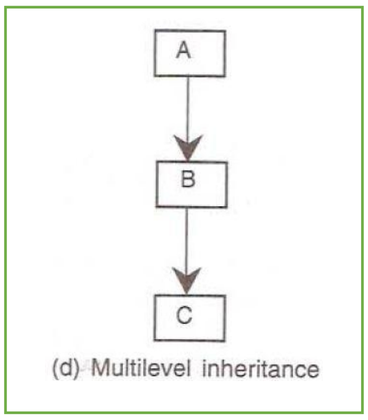
```
x value:10
y value:20
biggest number is:20
x value:10
y value:20
The mean of the given numbers:15.0
```

4. Multilevel inheritance

A derived class with another derived class is called multilevel inheritance.

(Or)

A class inherits members another derived class is called multilevel inheritance.



Example program:

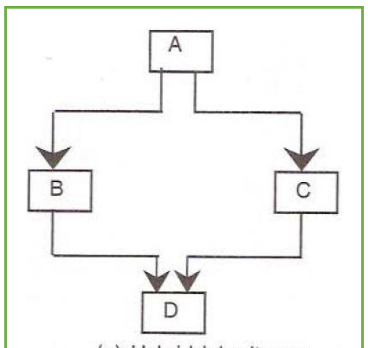
```
class Base
{
    int z;
    public void addition(int x, int y)
   {
            z = x + y;
    System.out.println("The sum of the given numbers:"+z);
   }
}
class Derived1 extends Base
{
    public void Subtraction(int x, int y)
   {
            z = x - y;
    System.out.println("The difference between the given numbers:"+z);
   }
}
class Derived2 extends Derived1
{
    public void multiplication(int x, int y)
   {
            z = x * y;
    System.out.println("The product of the given numbers:"+z);
   }
}
class MultilevelInheritance
{
    public static void main(String args[])
   {
            int a=10, b=20;
    Derived2 obj = new Derived2();
    obj.addition(a, b);
    obj.Subtraction(a, b);
    obj.multiplication(a, b);
  }
}
```

Output:

```
The sum of the given numbers:30
The difference between the given numbers:-10
The product of the given numbers:200
```

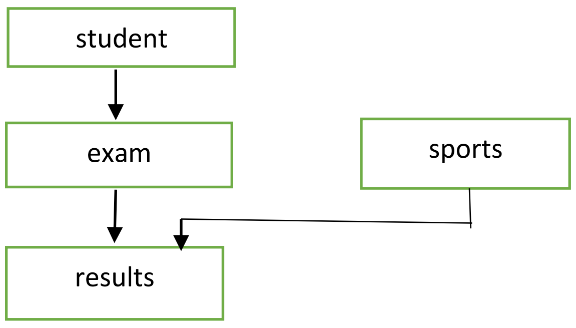
5. **Hybrid inheritance**

The combination of two or more inheritances is called hybrid inheritance.



(e) Hybrid inheritance

Example (student):



Programming exercises on Inheritance

1. W J P to implement **calculator** using inheritances.
2. W J P to implement **Student** using inheritances.
3. W J P to implement **Animal** using inheritances.
4. W J P to implement **vehicle** using inheritances.

Method Overriding:

- A derived class has the same method heading (return type, name, and parameters) as a method in its base class, then the method in the derived class is said to *override* the method in the base class.

Dynamic method dispatch:

- A mechanism in which call the derived class override method by using the base class reference is called dynamic method dispatch.
- Dynamic method dispatch resolved at runtime polymorphism.

Rules for method overriding:

- Base class and derived class method headings must be same.
  If a method cannot be inherited then it cannot be overridden.
  Instance methods can be overridden only if they are inherited by the subclass.
  A method declared static cannot be overridden but can be re-declared.
  A method declared final cannot be overridden.
  Constructors cannot be overridden.
  The method in base is need not to inherit in derived class.

Example: // Dynamic Method Dispatch

```
class A
{
        void display()
        {
                System.out.println("Inside A's display method");
        }
}
class B extends A {
        void display() {
                System.out.println("Inside B's display method");
        }
}
class Dispatch
{
        public static void main(String args[])
        {
                A obj1 = new A();
                B obj2 = new B();
                A ref;
                ref = obj1;
                ref.display();
                ref = obj2;
                ref.display();
        }
}
```

Output:

```
            Inside A's display method
            Inside B's display method
            Inside C's display method
```

UNIT – II

Constructors and Inheritance:

- Constructors are never participating in inheritance mechanism.
- When the derived class object is created then automatically execute the base class parameter less constructor.
- If you want to call the constructor manually use super keyword.

Example: demonstrate the constructors and inheritance

```
class Example1
{
        public Example1()
        {
                System.out.println("Base class constructor");
        }
}
class Example2 extends Example1
{
        public Example2()
        {
                System.out.println("Derived class constructor");
        }
}
class ConstDemo
{
        public static void main(String args[])
        {
                Example2 obj=new Example2();
        }
}
```

super keyword:

- super() calls the base class constructor.
- **super.methodname**() calls method from base class.
- It is used to solve the ambiguity (confusion) between base class and derived class variables.

//demonstrate the super keword

```
class Example1
{
        String color;
        public Example1()     {
                color="white";
        }
}
class Example2 extends Example1
{
        String color="black";
        public Example2()     {
                super();
        }
        public void display()
         {
                System.out.println("derived class color :"+color);
                System.out.println("base class color :"+super.color);
        }
}
class SuperDemo
{
        public static void main(String args[])
        {
                Example2 obj=new Example2();
                obj.display();
        }
}
```

Example: demonstrate the method overriding using Figure class.

```
class Shapes
{
      double dim1;
      double dim2;
      Shapes(double a, double b)
      {
            dim1 = a;
            dim2 = b;
      }
      double area()
      {
            System.out.println("Area for Shapes is undefined.");
            return 0;
      }
}
class Rectangle extends Shapes
{
      Rectangle(double a, double b)
      {
             super(a, b);
      }
      double area()
      {
             System.out.println("Inside Area for Rectangle.");
             return dim1 * dim2;
      }
}
class Triangle extends Shapes
{
      Triangle(double a, double b)
      {
         super(a, b);
      }
      double area()
      {
         System.out.println("Inside Area for Triangle.");
         return dim1 * dim2 / 2;
      }
}
class FindAreas {
        public static void main(String args[])
        {
              Shapes f = new Shapes(10, 10);
              Rectangle r = new Rectangle(9, 5);
              Triangle t = new Triangle(10, 8);
              Shapes figref;
              figref = r;
              System.out.println("Area is " + figref.area());
              figref = t;
              System.out.println("Area is " + figref.area());
              figref = f;
              System.out.println("Area is " + figref.area());
        }
  }
```

Output:

```
Inside Area for Rectangle.
Area is 45
Inside Area for Triangle.
Area is 40
Area for Shapes is undefined.
Area is 0
```

UNIT – II

abstract method:

- A method having declaration without any implementation or body is called abstract method.
- Abstract methods are declraed using ***abstract*** keyword.
- Syntax:

  abstract*type name*(*parameter-list*);

abstract class:

- A class having at least one abstract method is class abstract class.
- abstract class can have normal member methods and data members along with abstract method.
- Abstract class are defined using ***abstract*** keyword.
- Syntax:

```
abstractclass   classname
{
        //all members and abstract methods
}
```

Characteristics:

- All abstract method in abstract class must be implement in derived class, or else they will become abstract too.
- Abstract classes are mainly used for provide an interface for its derived classes.
- We can’t create object for abstract class but create reference.

Example:demonstrate the abstract class and abstract method

```
abstract class A
{
    abstract void callme();//abstract method
     void callmetoo()
     {
            System.out.println("This is a concrete method.");
     }
}
class B extends A
{
    void callme()
    {
            System.out.println("B's implementation of callme.");
    }
}
class AbstractDemo
{
    public static void main(String args[])
    {
         B b = new B();
         b.callme();
         b.callmetoo();
    }
}
```

Example: demonstrate the abstract class and method.

```
abstract class   Shapes
{
      double dim1;
      double dim2;
      Shapes(double a, double b)
      {
             dim1 = a;
             dim2 = b;
      }
      abstract double area();      //abstract method
}
class Rectangle extends Shapes
{
      Rectangle(double a, double b)
      {
              super(a, b);
      }
      double area()
      {
              System.out.println("Inside Area for Rectangle.");
              return dim1 * dim2;
      }
}
class Triangle extends Shapes
{
      Triangle(double a, double b)
      {
          super(a, b);
      }
      double area()
      {
          System.out.println("Inside Area for Triangle.");
          return dim1 * dim2 / 2;
      }
}
class FindAreas {
        public static void main(String args[])
        {
               Rectangle r = new Rectangle(9, 5);
               Triangle t = new Triangle(10, 8);
               Shapes figref;
               figref = r;
               System.out.println("Area is " + figref.area());
               figref = t;
               System.out.println("Area is " + figref.area());
        }
  }
```

Output:

```
Inside Area for Rectangle.
Area is 45.0
Inside Area for Triangle.
Area is 40.0
```

UNIT – II

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

1. final variable
2. final method
3. final class

1. **final variable:**
   - If you make any variable as final, you cannot change the value of final variable(It will be constant).
   - You can initialize a final variable when it is declared.

```
class Bike
{
       final int speedlimit=90;//final variable
       void run()
      {
           speedlimit=400;
 }
       public static void main(String args[])
       {
           Bike obj=new  Bike();
           obj.run();
 }
}
```

2. **final method:**
   - When a method is declared with *final* keyword, it is called a final method.
   - A final method cannot be overridden.

```
class A
{
       final void m1()
       {
      System.out.println("This is a final method.");
       }
}
class B extends A
{
    Public static void main(String args[])
    {
      System.out.println("Illegal!");// COMPILE-ERROR! Can't override.
    }
}
```

3. **final class:**
   - When a class is declared with final keyword, it is called a final class.
   - A final class cannot beextended(inherited).
   - There are two uses of a final class :
     - ❖ One is definitely to prevent inheritance, as final classes cannot be extended.
     - ❖ The other use of final with classes is to create an immutable class like the predefined String class. You cannot make a class immutable without making it final.

```
final class A
{
    // methods and fields
}
class B extends A
{
// COMPILE-ERROR! Can't subclass A
}
```

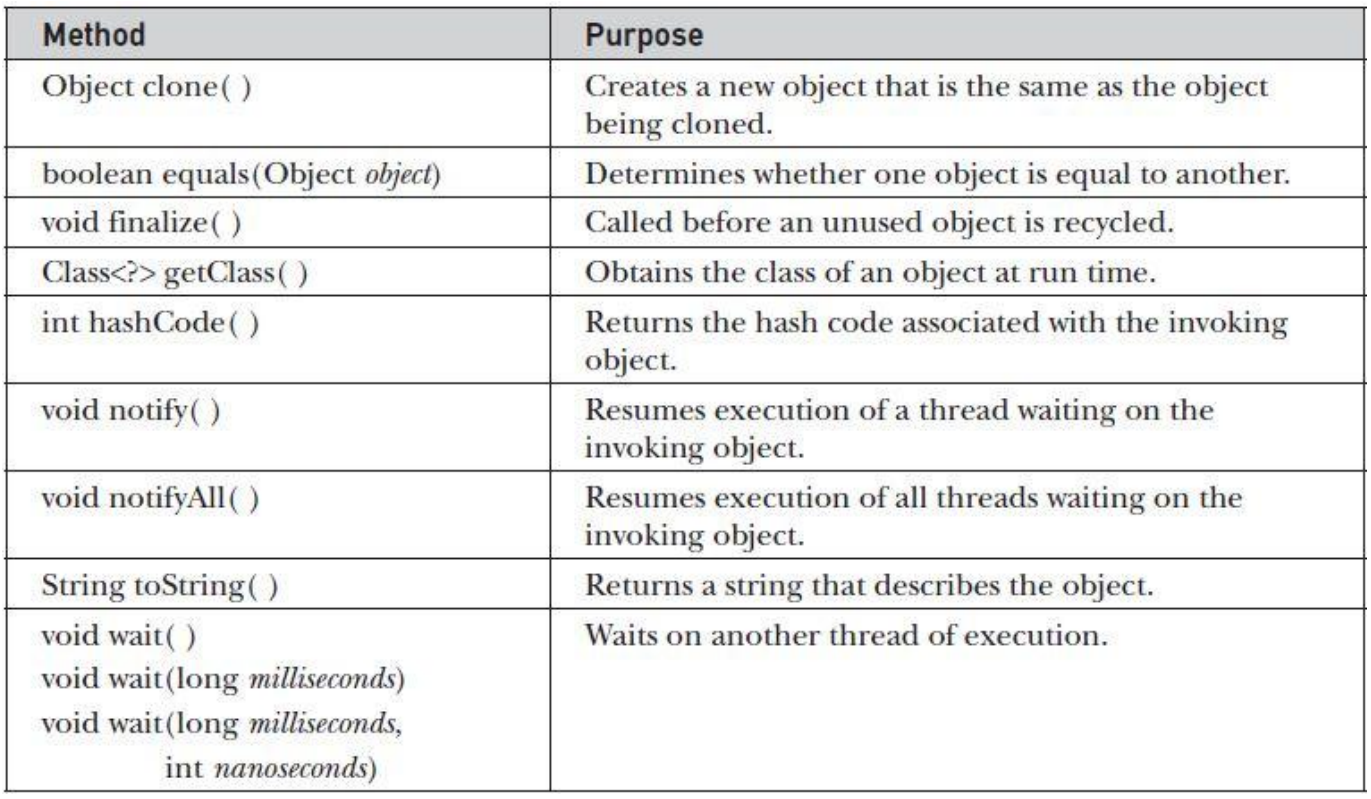
blank final variable:

- A final variable that have no value it is called blank final variable or uninitialized final variable.
- It can be initialized in the constructor only.
- The blank final variable can be static also which will be initialized in the static block only.

```
class  Bike
{
        final int speedlimit;          //blank final variable
        final static int  gares;      //blank static final variable
        public Bike()
       {
             speedlimit=400;
        }
        static
         {
             gares = 4;
        }
         public static void main(String args[])
         {
             Bike obj=new  Bike();
             System.out.println(“Speed Limit :”+obj.speedlimit);
             System.out.println(“No.Of gares :”+gares);
        }
 }
```

The Object Class

- **Object** is a superclass of all other classes.
- This means that a reference variable of type **Object** can refer to an object of any other class.



Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object *object*)	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled.
Class<?> getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long *milliseconds*) void wait(long *milliseconds*, int *nanoseconds*)	Waits on another thread of execution.

2.2 Packages and Interfaces

Interfaces:

- An **interface is** a collection of static constants and abstract methods.
- **Syntax:** defining interface

```
        accessinterface   name
        {
                return-type method-name1(parameter-list);
                return-type method-name2(parameter-list);
                type final-varname1 = value;
                type final-varname2 = value;
                //...
                return-type method-nameN(parameter-list);
                type final-varnameN = value;
        }
```

Implementing Interfaces

- Once an **interface** has been defined, one or more classes can implement that interface.
- To implement an interface, include the **implements** clause in a class definition, and thencreate the methods required by the interface.
- syntax:

```
        class classname [extends superclass] [implements interface[,interface...]]
        {
                // class-body
        }
```

Characteristics:

- Interfaces specify what a class must do and not how. It is the blueprint of the class.
- Like **abstract classes**, interfaces **cannot** be used to create objects.
- Interface methods do not have a body - the body is provided by the "implement" class on implementation of an interface, you must override all of its methods
- Interface methods are by default **public** and **abstract**
- Interface attributes are by default **public**, **static** and **final**
- An interface cannot contain a constructor (as it cannot be used to create objects)

Why use Java interface?

There are mainly three reasons to use interface.

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

Demonstrate the interface:

```
interface   Bank
{
        void rateOfInterest();
}
class  SBI implements Banks
{
        void rateOfInterest()    {
                System.out.println("SBI Rate of interest is 5%");
        }
}
class  HDFC  implements Banks
{
        void rateOfInterest()   {
                System.out.println("HDFC Rate of interest is 7%");
        }
}
```

```
        class  InterfaceDemo
        {
                Public static void main(String args[])
                {
                        Bank b1=new SBI();
                        b1.rateOfInterest();
                        Bank b2=new HDFC();
                        B2.rateOfInterest();
                }
        }
```

Multiple interface:

Write a java program to demonstrate the multiple inheritance

```
interface Sports
{
        void SportsInfo( );
}
interface Exam
{
        void Marks( );
}
class Student implements Sports, Exam
{
        void SportsInfo( )
        {
                System.out.println("He is playing cricket");
                System.out.println("Position: All-rounder");
        }
        void Marks( )
        {
                int s1=60, s2=60, s3=60, s4=60, s5=60, s6=60;
                float per;
                per=( (s1+s2+s3+s4+s5+s6)/600) *100'
                System.out.println("Percentage :"+per);
        }
        public static void main(String args[ ])
        {
                Student obje=new Student();
                obj. SportsInfo( );
                obj.Marks( )
        }
}
```

Output:

```
        He is playing cricket
        Position: All-rounder
        Percentage :60.00
```

Nested interface:

- An interface which is declared inside another interface or class is called nested interface.
- They are also known as inner **interface.**

Important Points:

- Nested interfaces are static by default. You don't have to mark them static explicitly as it would be redundant.
- Nested interfaces declared inside class can take any access modifier, however nested interface declared inside interface is public implicitly.

```
class Example
{
      interface Sample
      {
           void myMethod();
      }
}
class NestedInterfaceDemo  implements  Example.Sample
{
      public void myMethod()
      {
            System.out.println("Nested interface method");
      }
      public static void main(String args[])
      {
            Example.Sample obj = new NestedInterfaceDemo();
            obj.myMethod();
      }
}
```

Output:

```
Nested interface method
```

Default Interface Methods

A default method lets you define a default implementation or body for an interface method.

Important Points:

1. Interfaces can have default methods with implementation from java 8 onwards.
2. Interfaces can have static methods as well similar to static method of classes.
3. Default methods were introduced to provide backward compatibility for old interfaces so that they can have new methods without effecting existing code.

Example:

```
interface  Example
{
        void add(int x, int y);
        void mean()
        {
                System.out.println("This is default mean");
        }
}
class DefaultDemo implements Example
{
        void add(int x, int y)
        {
                System.out.println("Addition is :"(x+y));
        }
        public static void main(String args[])
        {
                DefaultDemo dd=new DefaultDemo();
                dd.add(10,20);
                dd.mean();
        }
}
```

Output:

```
Addition is :30
This is default mean
```

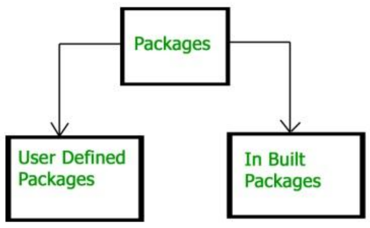
UNIT – II

Packages:

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.

Syntax:
package *pkgName*;

packages are categorized as inbuilt packages and user defined packages



Advantage of Java Package
1) package provides access protection.
2) package removes naming collision.
3) package is used to categorize the classes and interfaces so that they can be easily maintained.
4) Packages can be considered as data encapsulation (or data-hiding).

Accessing package:

Packages are accessed in two ways:

1. Using fully qualified name (without import)
2. Using import

1. **Using fully qualified name (without import)**

In fully qualified name mechanism every time we need to refer the class with package name.

Example:

//Save the program as **Employee.java** in directory name of **ITStaff**

```
package ITStaff;
public class Employee
{
        String dept="Information Technology";
        int staff=10;
        public void display()
        {
                System.out.println("Department Name :"+dept);
                System.out.println("No.of Staff :"+staff);
                System.out.println("Here staff details ...");
        }
}
```

//Save the program as PackageDemo**.java** in your working directory.

```
class PackageDemo
{
        public static void main(String s[])
        {
                ITStaff.Employee emp=new ITStaff.Employee();
                emp.display();
        }
}
```

Output:

```
Department Name : Information Technology
No.of Staff :10
Here staff details ...
```

2. Using import

Using import mechanism simply import the package once in a program and access class. Importing the packages by using import keyword.

Syntax: importing the package

```
        import pakagename.*;
```

syntax: importing the class

```
        import pkg1 [.pkg2].(classname | *);
```

Example:

```
//Save the program as Employee.java in directory name of ITStaff
package ITStaff;
public class Employee
{
        String dept="Information Technology";
        int staff=10;
        public void display()
        {
                System.out.println("Department Name :"+dept);
                System.out.println("No.of Staff :"+staff);
                System.out.println("Here staff details ...");
        }
}

//Save the program as PackageDemo.java in your working directory.
import  ITStaff.*;
class PackageDemo
{
        public static void main(String s[])
        {
                Employee emp=new Employee();
                emp.display();
        }
}
```

Output:

```
Department Name : Information Technology
No.of Staff :10
Here staff details ...
```

Member access and inheritance

Access modifier	In class	In package	Outside package by subclass	Outside package
public	Yes	Yes	Yes	No
protected	Yes	Yes	Yes	No
default	Yes	Yes	No	No
private	Yes	No	No	No

UNIT – II

SELF LEARNING EXCERCISES:

One-marks:

1. Define inheritance.
2. Is multiple inheritance possible in java?
3. Write the syntax for derived class.
4. What is the use of super keyword?
5. Define dynamic method dispatch.
6. List any four rules for method overriding.
7. Define abstract class.
8. What is abstract method.
9. What is use of final keyword?
10. What is blank final variable?
11. What is the use of Object class?
12. Define interface and write its syntax.
13. What is the use of default interface method?
14. Write the advantages of interfaces.
15. Define package.
16. List the various ways of accessing the packages.
17. List the advantages of packages.

Essay:

1. What is inheritance? Explain type of inheritance with examples.
2. Define method overriding. Explain with an example.
3. Explain final keyword with an example program.
4. What is abstract method and abstract class with an example program.
5. Define interface. Explain multiple interface with an example.
6. What is package? Write about one way of access the packages java with example.

Programs:

1. Write a java program to implements **hybrid inheritance using student class**.
2. Write a java program to demonstrate the **method overriding using Figure class**.
3. Write a java program to demonstrate the **packages of different Employees**.
4. Write a java program to demonstrate the **interface using Stack**.
5. Write a java program to demonstrate the **interface using Series**.
6. Write a java program to demonstrate the **interface using Addition**.