	Hope Foundation's **Finolex Academy of Management and Technology, Ratnagiri** **Department of MCA**

Subject name: Java Lab (SBL)			Subject Code: ITC304
Class	SEIT	Semester –III (CBCGS)	Academic year: 2022-23
Name of Student			**QUIZ Score :**
Roll No		Experiment No.	03

Title: **To study the concepts of inheritance, packages and interfaces and implement programs based on them.**

1. Lab objectives applicable:

LOB3 - To learn the principles of inheritance, interface and packages and demonstrate the concept of reusability for faster development.

2. Lab outcomes applicable:

1. **LO1** - Explain the fundamental concepts of Java Programing.
2. **LO3** - Demonstrate how to extend java classes and achieve reusability using Inheritance, Interface and Packages.

3. Learning Objectives:

1. To understand how concepts of inheritance.
2. To realize the importance of interface in providing limited accessibility.

4. Practical applications of the assignment/experiment: Concepts are used to improve reusability and accessibility.

5. Prerequisites:

1. Understanding of C programming.

6. Minimum Hardware Requirements:

1. P-IV PC with 2GB RAM, 500GB HDD, NIC

7. Software Requirements:

1. Windows / Linux operating systems, Java Developer Kit (1.8 or higher), Notepad++, Java Editors like Netbeans or Eclipse

8. Quiz Questions (if any): (Online Exam will be taken separately batch-wise, attach the certificate/ Marks obtained)

1. What is inheritance? What are its types?
2. What is the restriction on implementing an interface?

9. Experiment/Assignment Evaluation:

Sr. No.	Parameters	Marks obtained	Out of
1	Technical Understanding (Assessment may be done based on Q & A **or** any other relevant method.) Teacher should mention the other method used -		6
2	Lab Performance		2
3	Punctuality		2
Date of performance (DOP)		**Total marks obtained**	**10**

Signature of Faculty

10. Theory:

Java-Creating inheritance in java

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

extends Keyword

extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax

```
class Super {
  .....
  .....
}
class Sub extends Super {
  .....
  .....
}
```

The super keyword

The **super** keyword is similar to **this** keyword. Following are the scenarios where the super keyword is used.

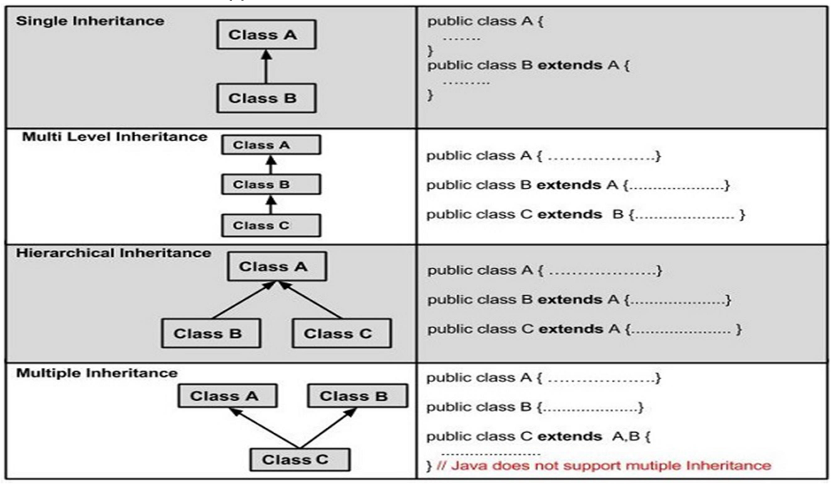
- It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
- It is used to **invoke the superclass** constructor from subclass.

Differentiating the Members

If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword.

Types of Inheritance

There are various types of inheritance as demonstrated below.



```
public class A {
  .......
}
public class B extends A {
  .........
}
```

```
public class A { ...................}
public class B extends A {....................}
public class C extends  B {..................... }
```

```
public class A { ...................}
public class B extends A {....................}
public class C extends A {..................... }
```

```
public class A { ...................}
public class B {....................}
public class C extends  A,B {
  ......................
} // Java does not support mutiple Inheritance
```

A very important fact to remember is that Java does not support multiple inheritance. This means that a class cannot extend more than one class.

Java-Creating abstract classes and interfaces in java:

As per dictionary, abstraction is the quality of dealing with ideas rather than events. For example, when you consider the case of e-mail, complex details such as what happens as soon as you send an e-mail, the protocol your e-mail server uses are hidden from the user. Therefore, to send an e-mail you just need to type the content, mention the address of the receiver, and click send.

Likewise, in Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

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.

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.
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.

Java-Creating and using user defined packages:

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.
A **Package** can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and namespace management.
Some of the existing packages in Java are −

- **java.lang** − bundles the fundamental classes
- **java.io** − classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, and annotations are related.
Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

Creating a Package

While creating a package, you should choose a name for the package and include a **package** statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.
The package statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.
If a package statement is not used then the class, interfaces, enumerations, and annotation types will be placed in the current default package.
To compile the Java programs with package statements, you have to use -d option as shown below.

```
javac -d Destination_folder file_name.java
```

Then a folder with the given package name is created in the specified destination, and the compiled class files will be placed in that folder.

The import Keyword

If a class wants to use another class in the same package, the package name need not be used. Classes in the same package find each other without any special syntax.

The Directory Structure of Packages

Two major results occur when a class is placed in a package −

- The name of the package becomes a part of the name of the class, as we just discussed in the previous section.
- The name of the package must match the directory structure where the corresponding bytecode resides.

11. Installation Steps / Performance Steps and Results –

Q1	Create a Teacher class and derive Professor/ Associate_Professor/Assistant_Professor class from Teacher class. Define appropriate constructor for all the classes. Also define a method to display information of Teacher. Make necessary assumptions as required.

Source code:

```
class teacher1
{
  String sname,saddress;
  float fid,fsallary;

  teacher1()
  {
    sname = "Unknown";
    saddress = "Not Given";
    fid = fsallary = 0f;
  }
  public void getValue(String sn,String sa,float fi, float fs)
  {
        this.sname=sn;
        this.saddress=sa;
        this.fid=fi;
        this.fsallary=fs;
    }
    public void display()
    {
        System.out.println("Name= "+sname+"\nAddress= "+saddress+"\nId= "+fid+"\nSallary= "+fsallary);

    }
}
class professor extends teacher1
{
    double dbonus;
    professor()
    {
        dbonus = 0.0;
    }
    public void getValue(String sn,String sa,float fi, float fs,double db)
    {
        super.getValue(sn, sa, fi, fs);
        this.dbonus=db;
    }
    public void display()
    {
        super.display();
        System.out.println("Bonus = "+dbonus);
    }
}
class school
{
    public static void main(String args[])
    {
        teacher1 t = new teacher1();
        t.getValue("Ayush","Teli Aali",01,10000);
        t.display();

        professor p = new professor();
        p.getValue("Ayush","Teli alli",04,10000,2000);
        p.display();
    }
}
```

Output:

```
OUTPUT:-
        Name= Ayush
        Address= Teli Aali
        Id= 1.0
        Sallary= 10000.0


        Name= Ayush
        Address= Teli alli
        Id= 4.0
        Sallary= 10000.0
        Bonus = 2000.0
```

Q2	Create a class Book and define a display method to display book information. Inherit Reference_Book and Magazine classes from Book class and override display method of Book class in Reference_Book and Magazine classes. Make necessary assumptions required.

Source code:

```
import java.util.Scanner;

class book {
   String book_name;
   String author_name;
   int ipages;

  public void kook(){



      Scanner sc = new Scanner(System.in);
      System.out.print("\nEnter name of book: ");
      book_name = sc.next();
      System.out.print("enter author of book : ");
      author_name = sc.next();
      System.out.print("enter pages of book : ");
      ipages = sc.nextInt();

   }
```

```
    public void display(){

        System.out.print(book_name+"\t\t"+author_name+"\t\t"+ipages);

    }

}
class referencebook extends book{
    String sn; String sa; int ip; double dp;

    double prize;
    public void getvalue(){

        super.kook();
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter prize: ");
        dp = sc.nextDouble();
        this.prize = dp;
    }
    public void display(){

        super.display();
        System.out.print("\t\t"+prize);
        System.out.println("\n");
    }

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

    /*    referencebook r = new referencebook();
        for (int i=0;i<=2;i++){

        r.getvalue();
        System.out.println("\n");
        System.out.println("BOOk NAME\tAUTHOR NSMR\tPAGES\t\tPRIZE");

        r.display();
```

```
      }
*/     int f;
      Scanner sc = new Scanner(System.in);


      System.out.println("entere number of books: ");
      f=sc.nextInt();

      referencebook r[]= new referencebook[f];
      try{ for( int i =0; i<f; i++){
      r[i]=new referencebook();
      r[i].getvalue();
      }
      System.out.println("BOOk NAME\tAUTHOR NSMR\tPAGES\t\tPRIZE");
      for(int i=0;i<f;i++){
         r[i].display();
      }
   }
   catch (Exception e){
         System.out.println("worong input");
   }
   }
}
```

Output:

```
enter number of books:
5

Enter name of book: Wings
enter author of book : Ayush
enter pages of book : 200
Enter prize: 120

Enter name of book: c
enter author of book : Prasad
enter pages of book : 150
Enter prize: 199

Enter name of book: Sailers
enter author of book : SisMarine
enter pages of book : 356
Enter prize: 400
Enter name of book: Teach
enter author of book : Saroj
enter pages of book : 150
Enter prize: 99

Enter name of book: Sugran
enter author of book : Sandhya
enter pages of book : 300
Enter prize: 299


        BOOk NAME      AUTHOR NAME     PAGES          PRIZE


        Wings                Ayush              200           120.0


        c                      Prasad             150           199.0


        Sailers            SidMarine         356           400.0


        Teach               Saroj              150           99.0


        Sugran           Sandhya            300           299.0
```

Q3	Create an interface vehicle and classes like bicycle, car, bike etc, having common functionalities and put all the common functionalities in the interface. Classes like Bicycle, Bike, car etc implement all these functionalities in their own class in their own way.

Source code:

```
interface vehicle
{
    void acceleration();
    void breke();
    void fuel();


}
class bicycle implements vehicle
{
   bicycle()
   {
      System.out.println("\n\tObject Bicycle");
   }
   public void acceleration()
```

```
    {
        System.out.println("Acceleration=\tPadling");
    }
    public void breke()
    {
        System.out.println("Break=\tHanded break");
    }
    public void fuel()
    {
        System.out.println("Fuel=\tBody Stamina");
    }
}
class bike implements vehicle
{
    bike()
    {
        System.out.println("\n\tobjetc Bike");
    }
    public void acceleration()
    {
        System.out.println("Acceleration=\tRight hand Accelerator");
    }
    public void breke()
    {
        System.out.println("Break=\tHanded break");
    }
    public void fuel()
    {
        System.out.println("Fuel=\tPetrol");
    }
}
class car implements vehicle
{
    car()
    {
        System.out.println("\n\tobject car");
    }
    public void acceleration()
    {
        System.out.println("Acceleration=\tRight Foot Accelerator");
    }
```

```
  public void breke()
  {
    System.out.println("Break=\tLeg break");
  }
  public void fuel()
  {
    System.out.println("Fuel=\tdisel");
  }
}
class interfaceDemo
{
  public static void main(String args[])
  {
    bicycle b = new bicycle();
    b.acceleration();
    b.breke();
    b.fuel();

    bike b1 = new bike();
    b1.acceleration();
    b1.breke();
    b1.fuel();

    car c = new car();
    c.acceleration();
    c.breke();
    c.fuel();
  }
}
```

Output:

```
            Object Bicycle
        Acceleration=   Padling
        Break=  Handed break
        Fuel=   Body Stamina

            objetc Bike
        Acceleration=   Right hand Accelerator
        Break=  Handed break
        Fuel=   Petrol
```

object car

Acceleration= Right Foot Accelerator

Break= Leg break

Fuel= disel

12. Learning Outcomes Achieved

1. Students have understood inheritance and its types.
2. Students have understood how interface can be efficiently used to improve functionality.

13. Conclusion:

1. **Applications of the studied technique in industry**
    a. The concept of abstract classes and interfaces can be used to create basic frameworks.
2. **Engineering Relevance**
    a. Complex design can be effectively managed using inheritance and interfaces.
3. **Skills Developed**
    a. Code efficiency with inheritance and interface.

14. References:

[1] Stackoverflow.com. Stackoverflow.com is probably the most popular website in the programming world.

[2] Dzone.com.

[3] Java SE Technical Documentation. ...

[4] Java 2: The Complete Reference Book by Herbert Schildt

[5] The Java Language Specification Book by Bill Joy, Gilad Bracha, Guy L. Steele Jr., and James Gosling