# LAB - 6

## Java Programming Lab

### Topics Covered

Inheritance, Polymorphism, Method overriding

Syeda Reeha Quasar
14114802719
4C7

# EXPERIMENT – 6.1

## Aim:

Create an abstract class shape. Let rectangle and triangle inherit this shape class. Add necessary functions.

## Theory:

**Class :** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

**Object :** It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

**This keyword :** this() reference can be used during constructor overloading to call default constructor implicitly from parameterized constructor.

**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). **Abstraction** is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details. You don't know the internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it. A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

**Concepts:**

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.
- **Java Utils:** Resources Job Search Discussion. Java. util package contains the **collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes**. This reference will take you through simple and practical methods available in java.
- util. Java util package contains **collection framework, collection classes**, classes related to date and time, event model, internationalization, and miscellaneous utility classes. ... On importing this package, you can access all these classes and methods.
- **Scanner** is a class in **java. util package used for obtaining the input of** the primitive types like int, double, etc. and strings. ... To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream.
- A switch statement **allows a variable to be tested for equality against a list of values**. Each value is called a case, and the variable being switched on is checked for each case.
- The input is **the data that we give to the program**. The output is the data what we receive from the program in the form of result. Stream represents flow of data or the sequence of data.

## Source Code:

Shape class :

```
public abstract class Shape
{
   private double height;
   private double width;

   public void setValues(double height, double width)
   {
      this.height = height;
      this.width = width;
   }

   public double getHeight()
   {
       return height;
   }

   public double getWidth()
   {
       return width;
   }

   public abstract double getArea();
}
```

Rectangle Class :

```
public class Rectangle extends Shape
{
    public double getArea()
    {
        return getHeight() * getWidth();
    }
}
```

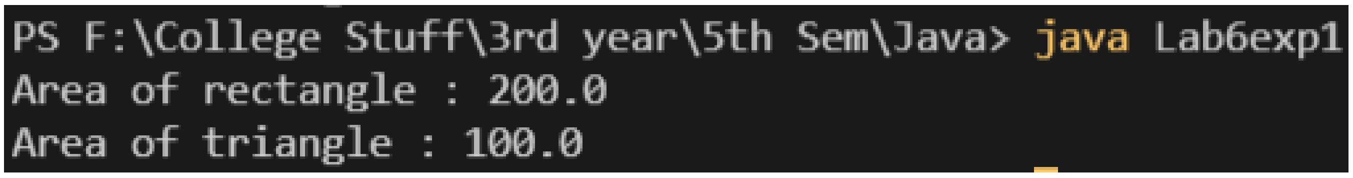
Triangle Class :

```
public class Triangle extends Shape
{
    public double getArea()
    {
        return (getHeight() * getWidth()) / 2;
    }
}
```

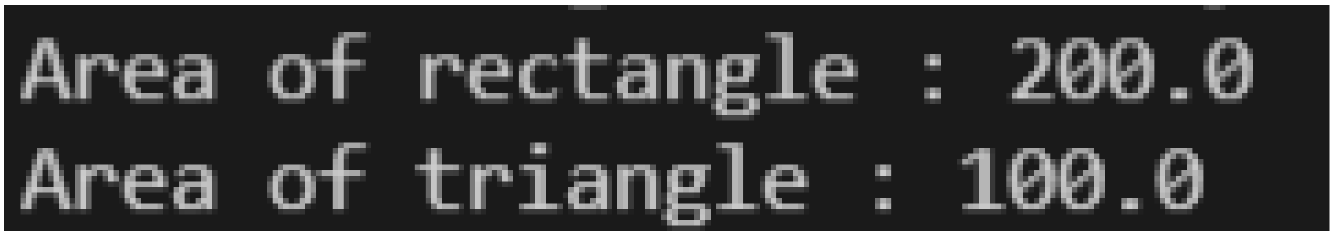
Main Class :

```
public class AbstractDemo
{
    public static void main(String[] args)
    {
        Shape obj;

        //Rectangle object
        Rectangle rect = new Rectangle();
        obj = rect;
        obj.setValues(10, 20);
        System.out.println("Area of rectangle : " + obj.getArea());
        /Triangle object
        Triangle tri = new Triangle();
        obj = tri;
        obj.setValues(10,20);
        System.out.println("Area of triangle : " + obj.getArea());
    }
}
```

**Output:**



```
PS F:\College Stuff\3rd year\5th Sem\Java> java Lab6exp1
Area of rectangle : 200.0
Area of triangle : 100.0
```



```
Area of rectangle : 200.0
Area of triangle : 100.0
```

# EXPERIMENT – 6.2

## Aim:

Write a java package to show dynamic polymorphism and interfaces.

## Theory:

**Class :** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

**Object :** It is a basic unit of Object-Oriented Programming and represents the real life entities.  A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator ,  e.g  c1.radius , and invoke other class methods  by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

**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 methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship.** It cannot be instantiated just like the abstract class. An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

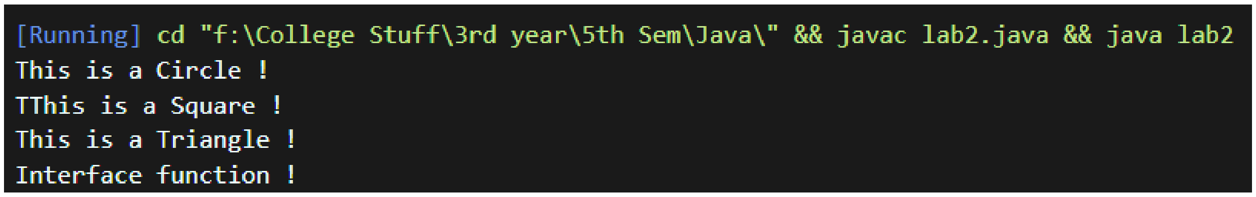
**Polymorphism :** **Polymorphism** in Java occurs when there are one or more classes or objects related to each other by inheritance. It is the ability of an object to take many forms. Inheritance lets users inherit attributes and methods, and polymorphism uses these methods to perform different tasks. So, the goal is communication, but the approach is different.

**Dynamic Polymorphism :** Dynamic polymorphism is a process or mechanism in which a call to an overridden method is to resolve at runtime rather than compile-time. It is also known as runtime polymorphism or dynamic method dispatch. We can achieve dynamic polymorphism by using the method overriding. In this process, an overridden method is called through a 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.

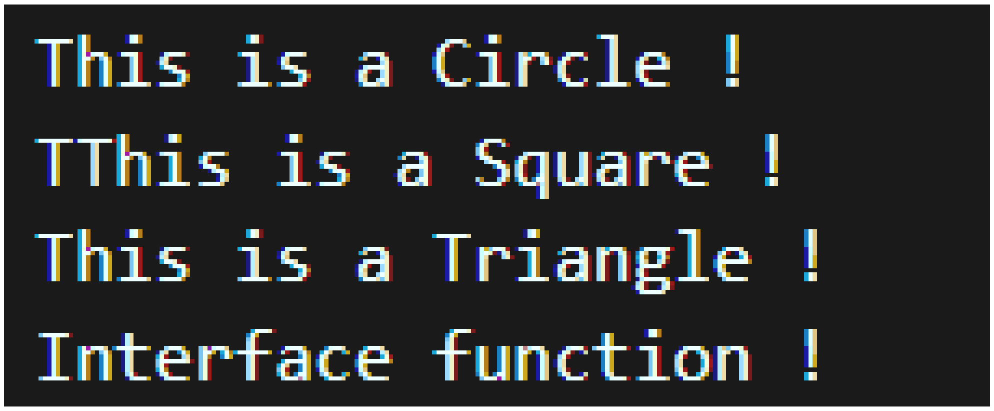
## Source Code:

```
package shapes;

interface i1 {
    void func();
}

class circle implements i1 {
    public void func() {
        System.out.println("Interface function !");
    }

    public void name() {
        System.out.println("This is a Circle ! ");
    }
}

class square extends circle {
    public void name() {
        System.out.println("TThis is a Square ! ");
    }
}

class triangle extends circle {
    public void name() {
        System.out.println("This is a Triangle ! ");
    }
}

public class lab2 {
    public static void main(String[] args) {
        circle obj1 = new circle();
        circle obj2 = new square();
        circle obj3 = new triangle();
        obj1.name();
        obj2.name();
        obj3.name();
        obj1.func();
    }
}
```

## Output:



```
[Running] cd "f:\College Stuff\3rd year\5th Sem\Java\" && javac lab2.java && java lab2
This is a Circle !
TThis is a Square !
This is a Triangle !
Interface function !
```



```
This is a Circle !
TThis is a Square !
This is a Triangle !
Interface function !
```

# EXPERIMENT – 6.3

## Aim:

Write an application that creates an ‘interface’ and implements it.

## Theory:

- **Class :** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

- **Object :** It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g c1.radius , and invoke other class methods by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

- **New Operator :** The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

- **Constructor :** In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to initialize the object’s values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

- **Constructor Overloading :** Java Constructor overloading is a technique in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

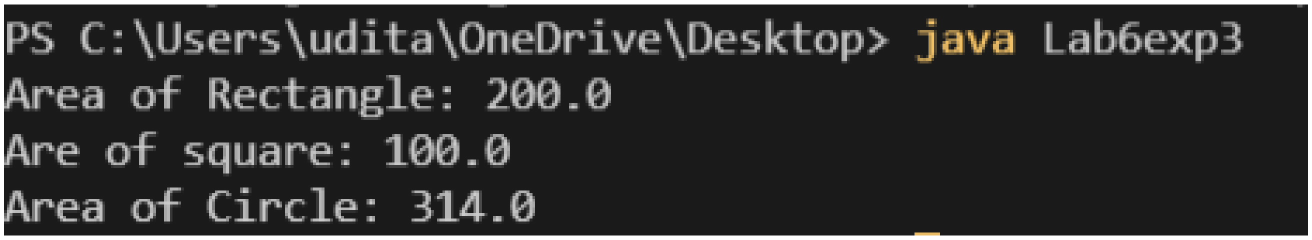
- **Interface :** An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**. It cannot be instantiated just like the abstract class.

- An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

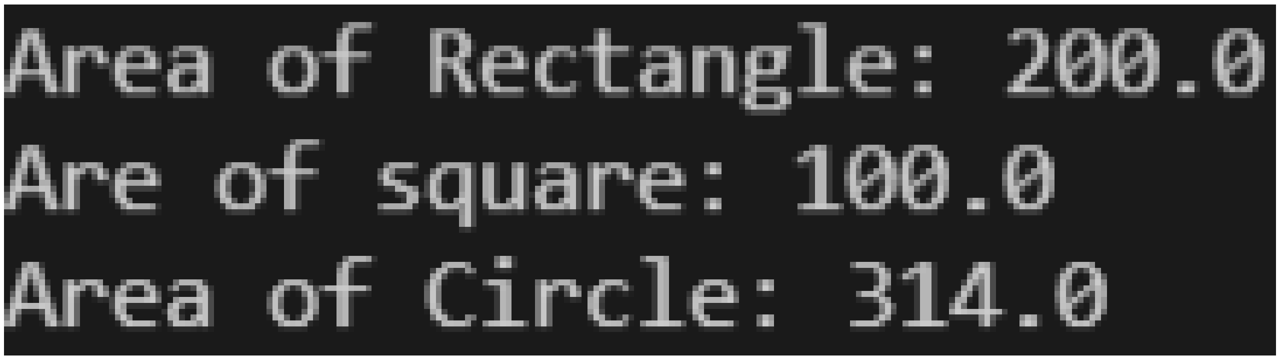
## Source Code:

```
interface Results {
    final static float pi = 3.14f;
    float areaOf(float l, float b);
}

class Rectangle implements Results {
    public float areaOf(float l, float b) {
        return (l * b);
    }
}

class Square implements Results {
    public float areaOf(float l, float b) {
        return (l * l);
    }
}

class Circle implements Results {
    public float areaOf(float r, float b) {
        return (pi * r * r);
    }
}

public class Lab6exp3 {
    public static void main(String args[]) {
        Rectangle rect = new Rectangle();
        Square square = new Square();
        Circle circle = new Circle();
        System.out.println("Area of Rectangle: " + rect.areaOf(10, 20));
        System.out.println("Are of square: " + square.areaOf(10, 20));
        System.out.println("Area of Circle: " + circle.areaOf(10, 20));
    }
}
```

**Output:**



```
PS C:\Users\udita\OneDrive\Desktop> java Lab6exp3
Area of Rectangle: 200.0
Are of square: 100.0
Area of Circle: 314.0
```



```
Area of Rectangle: 200.0
Are of square: 100.0
Area of Circle: 314.0
```

# EXPERIMENT – 6.4

## Aim:
Write an application to illustrate Interface Inheritance.

## Theory:

- **Class :** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. It represents the set of properties or methods that are common to all objects of one type. A class in java has its methods, variables.

- **Object :** It is a basic unit of Object-Oriented Programming and represents the real life entities.  A typical Java program creates many objects, interact by invoking methods. We create object of a class in another class to access its methods and variables. We can declare an object by using the Class Name of the class of which we want to create an object, followed by object name, e.g Circle c1; . An object can access variables of other class by using the . operator , e.g  c1.radius , and invoke other class methods  by using the method name and passing the parameters(if required), e.g c1.getArea(radius);

- **New Operator :** The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor. Example of object initialisation : Circle c1 = new Circle();

- **Constructor :** In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. We can pass our values , when calling out the constructor using new operator, to initialize the object's values. If there is no constructor available in the class It calls a default constructor. In such case, Java compiler provides a default constructor by default.

- **Constructor Overloading :** Java Constructor overloading is a technique in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

- **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 methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**. It cannot be instantiated just like the abstract class.

- An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

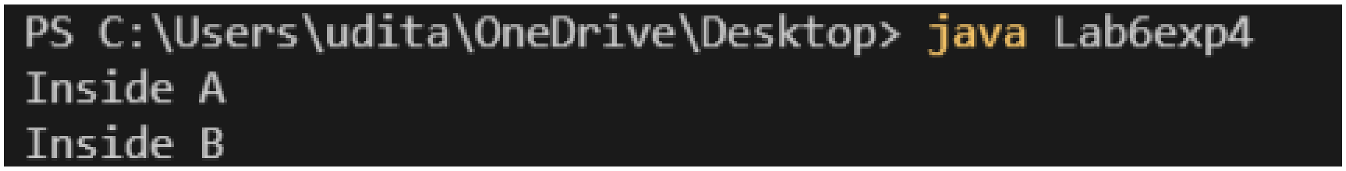
- **Inheritance : Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of Object Oriented programming system. The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also. 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** is the keyword used to inherit the properties of a class.

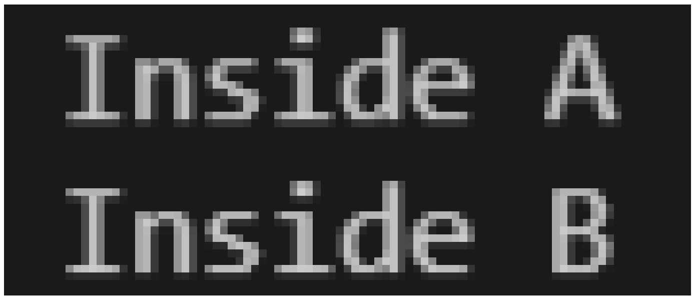
## Source Code:

```
interface A
{
    void func_A();
}

interface B extends A
{
    void func_B();
}

class Lab6exp4 implements B
{
    @Override
    public void func_A()
    {
        System.out.println("Inside A");
    }

    @Override
    public void func_B()
    {
        System.out.println("Inside B");
    }

    public static void main (String[] args)
    {
        Lab6exp4 obj1 = new Lab6exp4();


        obj1.func_A();
        obj1.func_B();
    }
}
```

## Output:



```
PS C:\Users\udita\OneDrive\Desktop> java Lab6exp4
Inside A
Inside B
```



```
Inside A
Inside B
```

# Viva Questions

## 1. What is interface in Java?

Ans.

Interface in java is core part of Java programming language and one of the way to achieve abstraction in Java along with abstract class. Even though interface is fundamental object oriented concept; Many Java programmers thinks Interface in Java as advanced concept and refrain using interface from early in programming career. At very basic level interface in java is a keyword but same time it is an object oriented term to define contracts and abstraction , This contract is followed by any implementation of Interface in Java. Since multiple inheritance is not allowed in Java, interface is only way to implement multiple inheritance at Type level. In this Java tutorial we will see What is an interface in Java, How to use interface in Java and where to use interface in Java and some important points related to Java interface. What is an interface in Java is also a common core Java question which people asked on various programming exams and interviews.

## 2. What is the difference between an Interface and an Abstract class?

Ans.

An abstract class can have instance methods that implement a default behaviour. An Interface can only declare constants and instance methods, but cannot implement default behaviour and all methods are implicitly abstract. An interface has all public members and no implementation. An abstract class is a class which may have the usual flavours of class members (private, protected, etc.), but has some abstract methods.

## 3. I don't want my class to be inherited by any other class. What should i do?

Ans.

You should declared your class as final. But you can't define your class as final, if it is an abstract class. A class declared as final can't be extended by any other class.

## 4. Can an interface extends another interface in Java?

Ans.

Yes, an interface can extend another interface.

## 5. Can an interface implement another interface?

Ans.

No, an interface cannot implement another interface.

## 6. Why do we need to use inheritance?

Ans.

Inheritance is one of the main pillars of OOPs concept. Some objects share certain properties and behaviors. By using inheritance, a child class acquires all properties and behaviors of parent class.

There are the following reasons to use inheritance in java.

- We can reuse the code from the base class.
- Using inheritance, we can increase features of class or method by overriding.
- Inheritance is used to use the existing features of class.
- It is used to achieve runtime polymorphism i.e method overriding.