Shaikh Salif

S22-94

Experiment No. 10

Aim: To implement program on abstract class and abstract methods.

Theory:

Abstract class in Java

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).

Before learning the Java abstract class, let's understand the abstraction in Java first.

Abstraction in Java

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, for example, sending SMS where you type the text and send the message. 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.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

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.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors

  and static methods also.

- It can have final methods which will force the subclass not to change the body of the method.

Example of abstract class

```
abstract class A{}
```

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method

```
abstract void printStatus();//no method body and abstract
```

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();
}
}
```

Mostly, we don't know about the implementation class (which is hidden to the end user), and an object of the implementation class is provided by the **factory method**.

A **factory method** is a method that returns the instance of the class. We will learn about the factory method later.

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

```
 abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., g
etShape() method
s.draw();
}
}
```

Abstract class having constructor, data member and methods

```
//Example of an abstract class that has abstract and non-abstract methods
 abstract class Bike{
```

```
  Bike(){System.out.println("bike is created");}
  abstract void run();
 void changeGear(){System.out.println("gear changed");}
 }
//Creating a Child class which inherits Abstract class
 class Honda extends Bike{
void run(){System.out.println("running safely..");}
 }
//Creating a Test class which calls abstract and non-abstract methods
 class TestAbstraction2{
public static void main(String args[]){
 Bike obj = new Honda();
  obj.run();
 obj.changeGear();
 }
}
```

Code:

```
import java.util.*;
abstract class Figure
{
abstract double area(double a, double b);
}
class Triangle extends Figure
{
double area(double b, double h)
{
return(0.5*b*h);
}
}
class Circle extends Figure
{
double area(double r, double h)
{
return(3.14*r*r);
}
}
```

```
class Square extends Figure
{
double area(double s, double x)
{
return(s*s);
}
}
class Rectangle extends Figure
{
double area(double l, double b)
{
return(l*b);
}
}
public class Main
{
public static void main(String[] args)
{
Scanner sc=new Scanner(System.in);
System.out.println("DIPSHREE PARMAR S21-78");
System.out.println("Enter two numbers: ");
double a= sc.nextDouble();
double b=sc.nextDouble();
Triangle obj1=new Triangle();

Circle obj2= new Circle();
Square obj3=new Square();
Rectangle obj4= new Rectangle();
System.out.println("Area of triangle:"+obj1.area(a,b));
System.out.println("Area of circle: "+obj2.area(a,b));
System.out.println("Area of square: "+obj3.area(a,b));
System.out.println("Area of rectangle: "+obj4.area(a,b));
}
}
```

Output:

```
C:\Users\shaik\OneDrive\Documents\JAVA>java Main
Shaikh Salif S22-94
Enter two numbers:
12 20
Area of triangle:120.0
Area of circle: 452.15999999999997
Area of square: 144.0
Area of rectangle: 240.0

C:\Users\shaik\OneDrive\Documents\JAVA>
```

Conclusion: In conclusion, abstract classes and abstract methods are powerful tools in object-oriented programming that allow us to design flexible and extensible software systems. They provide a way to define a common interface and set of behaviors for a group of related classes while leaving the implementation details to the concrete subclasses.