VIT Vidyalankar Institute of Technology Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING

Experiment No. 10

Semester	S.E. Semester III – Computer Engineering
Subject	Skill Based Lab Course: OOP with Java (CSL304)
Subject Professor In-charge	Prof. Swapnil S. Sonawane
Assisting Teachers	Prof. Swapnil S. Sonawane

Student Name	HERAMBA PRASANNA LIMAYE
Roll Number	19102A0033

Title: Program on abstract class and abstract methods

Objective:
To implement the concept of inheritance and interfaces.

Explanation:

An *abstract class* is a class that is declared `abstract`—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract void moveTo(double deltaX, double deltaY);
```

If a class includes abstract methods, then the class itself *must* be declared `abstract`, as in:

```
public abstract class GraphicObject {
   // declare fields
   // declare nonabstract methods
```

```
    abstract void draw();
}
```

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared `abstract`.

Abstract classes are similar to interfaces. You cannot instantiate them, and they may contain a mix of methods declared with or without an implementation. However, with abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods. With interfaces, all fields are automatically public, static, and final, and all methods that you declare or define (as default methods) are public. In addition, you can extend only one class, whether or not it is abstract, whereas you can implement any number of interfaces.

Program Code:

```
import java.util.*;
abstract class shape
{
int radius;
void accept()
{
Scanner t=new Scanner(System.in);
System.out.println("Enter radius=");
radius=t.nextInt();
}
abstract void area();
};

class circle extends shape
{
void area()
{
double a=3.14*radius*radius;
```

```
System.out.println("Area of Circle="+a);
}
}

class sphere extends shape
{
void area()
{
double a=4*3.14*radius*radius;
System.out.println("Area of Sphere="+a);
}
}

class abst
{
public static void main(String args[])
{
circle c=new circle();
c.accept();
c.area();
sphere s=new sphere();
s.accept();
s.area();
}
}
```

Output:

```
C:\>cd C:\java store

C:\java store>set path=D:\jdk\bin

C:\java store>javac abst.java

C:\java store>java abst
Enter radius=
3
Area of Circle=28.259999999999998
Enter radius=
3
Area of Sphere=113.03999999999999

C:\java store>
```

Conclusion:

In an object-oriented drawing application, you can draw circles, rectangles, lines, Bezier curves, and many other graphic objects. These objects all have certain states (for example: position, orientation, line color, fill color) and behaviors (for example: moveTo, rotate, resize, draw) in common. This is a perfect situation for an abstract superclass.And in similar situations abstract methods and classes help in getting code reusability along with ability to change and modify it.
