Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)
Department of Computer Science & Engineering
UCS1313 – Object Oriented Programming Using Java Lab

Exercise – 4 – Abstract class and Interfaces

Objective:

- To test the following Inheritance type: multiple inheritance.
- To test the Polymorphism through Interface / abstract classes by method overriding.

Sample Learning Outcome:

- Need of interface and it's implementation in Java
- Need of abstract class and it's implementation in Java
- Multiple inheritance
- Accessing the derived class objects through base class/interface reference – Dynamic method dispatch/Dynamic binding

Best Practices:

- Class Diagram usage
- Naming convention – for file names, variables
- Comment usage at proper places
- Prompt messages during reading input and displaying output
- Incremental program development
- Modularity
- All possible test cases in output

1. Design a class called **Person** as described below:

Person
-name:String -address:String
+Person(name,address) +getName():String +getAddress():String +setAddress(address):void

A sub-class Employee of class Person is designed as shown below:

Employee
-empid:String -dept:String -basic:int
+Employee(name,address,empid,dept,basic) +getEmpid():int +getDept():String +setDept(dept):void +setBasic(basic):void +getBasic():int +calSalary():float

A sub-class Faculty of class Employee is designed as shown below:

Faculty
-designation:String -course:String
+Faculty(name,address,empid,dept,basic,desig,course) +getDesig():String +setDesig(desig):void +setCourse(course):void +getCourse():float **+calSalary():float**

Design an Interface Student:

<<Student>>
+getMarks():float [] +calcGPA():float

Design a sub-class TeachingAssitant of class Employee, implements <<Student>>

TeachingAssitant
-project:String -course:String -marks:float []
+TeachingAssitant(name,address,empid,dept,basic,project,course,marks) +getProject():String +getCourse():String +setCourse(course):void **+getMarks():float []** **+calcGPA():float** **+calSalary():float**

Write a TestDriver function to get input for Faculty and TeachingAssistant and display their details. Find the class that can be kept as abstract.

==

Shape
#color:String="red"

+Shape() +Shape(color) +getColor():String +setColor(color):void ***abs getArea():float*** ***abs getPerimeter():float***

II) Create a class hierarchy for the following using Interface /

Abstract class: Design **Shape** as described below:

Where abs – abstract method

A sub-class **Circle** of class *Shape* is designed as shown below:

Circle
#radius:float=1.0
+Circle() +Circle(radius) +Circle(radius,color) +getRadius():float +setRadius(radius):void **+getArea():float** **+getPerimeter():float**

A sub-class **Rectangle** of class *Shape* is designed as shown below:

Rectangle
#width:float=1.0 #length:float=1.0

+Rectangle()
+Rectangle(width,length)
+Rectangle(width,length,color)
+getWidth():float
+setWidth(width):void
+getLength():float
+setLength(length):void
+getArea():float
+getPerimeter():float

A sub-class **Square** of class *rectangle* designed as shown below (Square is one where the length and width of rectangle are same):

Square
+Square()
+Square(side)
+Square(side,color)
+getSide():float
+setSide(side):void
+getArea():float
+getPerimeter():float

Note the following:

- Shape contains the abstract methods.
- Those abstract methods are to be implemented by the defining classes.

EXERCISE :

- Draw the class diagram of the above class hierarchy.

- Implement the above class hierarchy by using Interface and Abstract class.

Hint:

<u>*To write an Interface*</u>*:*

- *Only abstract methods can be declared inside the Interface.*

- *Identify the common behavior of the set of objects and declare that as abstract methods inside the Interface.*
- *The classes that implements the Interface will provide the actual implementation of those abstract methods.*

<u>*To write an Abstract class*</u>*:*

- *An abstract class can have constructor(s), abstract or non-abstract method(s).*

- *Define the constructors and non-abstract method in the Abstract class Shape. Declare the common behavior as the abstract method.*

- *Let the classes Rectangle, Circle, Square define its own constructors, member variable and methods.*

- Write a *test driver* called `TestInterface | TestAbstract` . Use an array of objects of type Shape to display the area, perimeter of all the shapes (Circle, Rectangle, Square).

- Note down the differences while implementing the Inheritance through Interface and Abstract class.

- Note the run-time polymorphism in resolving the method call exhibited by Java through method overriding.