**JAVA POLYMORPHISM ASSIGNMENT - 1**

Question 1: Consider the above example. Suppose that our program uses many kinds of shapes, such as triangles, rectangles, and so on. We should design a superclass called Shape, which defines the public interfaces (or behaviors) of all the shapes. For example, we would like all the shapes to have a method called getArea(), which returns the area of that particular shape. Hint: (The subclasses override the getArea() method inherited from the superclass, and provide the proper implementations for getArea().) 2.

Solution:

Super class Shape:

public class **Shape** {

private String color;

public Shape (String color) {

this.color = color;

}

public String toString() {

return "Shape[color=" + color + "]";

}

public double getArea() {

System.err.println("Shape unknown! Cannot compute area!");

return 0;

}

}

Subclass Rectangle:

public class **Rectangle extends Shape** {

private int length, width;

public Rectangle(String color, int length, int width) {

super(color);

this.length = length;

this.width = width;

}

public String toString() {

return "Rectangle[length=" + length + ",width=" + width + "," + super.toString() + "]";

}

public double getArea() {

return length\*width;

}

}

Subclass Triangle:

public class **Triangle extends Shape** {

private int base, height;

public Triangle(String color, int base, int height) {

super(color);

this.base = base;

this.height = height;

}

public String toString() {

return "Triangle[base=" + base + ",height=" + height + "," + super.toString() + "]";

}

public double getArea() {

return 0.5\*base\*height;

}

}

Test shape: public class **TestShape** {

public static void main(String[] args) {

Shape s3 = new Shape("green");

System.out.println(s3);

System.out.println("Area is " + s3.getArea());

}

}

Question 2: Consider a game app, we have many types of monsters that can attack. We shall design a superclass called Monster and define the method attack() in the superclass. The subclasses shall then provide their actual implementation. In the main program, we declare instances of the superclass, substitute with the actual subclass; and invoke the method defined in the superclass.

Solution:

Super class Monster:

public class **Monster** {

private String name;

public Monster(String name) {

this.name = name;

}

public String attack() {

return "!^\_&^$@+%$\* I don't know how to attack!";

}

}

**Subclass FireMonster**

public class **FireMonster extends Monster** {

public FireMonster(String name) {

super(name);

}

public String attack() {

return "Attack with fire!";

}

}

**Subclass WaterMonster**

public class **WaterMonster extends Monster** {

public WaterMonster(String name) {

super(name);

}

public String attack() {

return "Attack with water!";

}

}

**Subclass StoneMonster**

public class **StoneMonster extends Monster** {

public StoneMonster(String name) {

super(name);

}

@Override

public String attack() {

return "Attack with stones!";

}

}

**A Test Driver TestMonster**

public class TestMonster {

public static void main(String[] args) {

Monster m1 = new FireMonster("r2u2");

Monster m2 = new WaterMonster("u2r2);

Monster m3 = new StoneMonster("r2r2");

System.out.println(m1.attack());

System.out.println(m2.attack());

System.out.println(m3.attack());

m1 = new StoneMonster("a2b2");

System.out.println(m1.attack());

Monster m4 = new Monster("u2u2");

System.out.println(m4.attack());

}

}