Recitation 5

Abstract Classes, Polymorphism, Javadoc

1. Abstract Classes.

   For each of the following, tell whether the code will compile. If not, explain why.

   1.
      ```
      public abstract class X { }
      ```

   2.
      ```
      public class X {
        public abstract void stuff();
      }
      ```

   3.
      ```
      public abstract class X {
        public abstract void stuff() {
          System.out.println("abstract");
      }
      ```

   4.
      ```
      public abstract class X {
        public void stuff() {
          System.out.println("go figure");
      }
      ```

   5.
      ```
      public abstract class X {
        public abstract void stuff();
      }
      public class Y extends X { }
      ```

   6.
      ```
      public interface I {
        void stuff();
      }
      public abstract class X {
        public abstract void stuff();
      }

      public class Y extends X implements I {
        public void stuff() { }
      }
      ```

   7.
      ```
      public abstract class X {
        private int i,j;
        public void stuff1() { }
        public void stuff2() { }
      }
      ```

   8.
      ```
      public abstract class C {
        public void write() {
          System.out.println("C");
        }
        public static void main(String[] args) {
          C c = new C().write();
        }
      }
      ```

9.

```
public abstract class C {
  public abstract void write();
}
public class D extends C {
  public void write() {
    System.out.println("D");
  }
  public static void main(String[] args) {
    C c = new D();
    c.write();
  }
}
```

---

2. There is an application that defines a Person class and a Student class. The Student class is defined as a subclass of Person. Every person has a home address, while every student has a school address as well.

   Consider printing addresses of all people in the application, assuming there is a single array list that stores all Person and Student objects. How would the address that is printed for students depend on the way the Student class address methods are designed/implemented? What alternatives in design can you think of, and what are the pros and cons of these alternatives in printing the addresses?

---

3. This problem gives an example where polymorphism is useful. Consider the class hierarchy given below :

```
public abstract class Shape implements Comparable<Shape> {

  public void print() {
    System.out.println("Shape");
  }

  public abstract double getArea();

  public static final Shape biggest(Shape[] s)  {
    /** TO BE COMPLETED BY YOU **/
  }

  ... // OTHER METHODS/FIELDS YOU MAY NEED TO ADD TO ANSWER THE QUESTION
}

public class Circle extends Shape {
  double radius;

  public Circle(double r) {
    radius = r;
  }

  public void print() {
    System.out.println("Circle");
  }

  public double getArea() {
    return Math.PI*radius*radius;
  }
}
```

```
public class Rectangle extends Shape {

   double height;
   double length;

   public Rectangle(double l,double h) {
      length = l;
      height = h;
   }

   public void print() {
      System.out.println("Rectangle");
   }

   public double getArea() {
      return length*height;
   }
}

public class App {

   public static void main(String[] args) {
      Shape[] s = new Shape[3];

      s[0] = new Circle(7);
      s[1] = new Rectangle(5,10);
      s[2] = new Circle(4);

      System.out.println("The biggest area of all shapes is : "+Shape.biggest(s));
      return;
   }
}
```

Complete the method

```
public static Shape biggest(Shape[] s)
```

in the Shape class. This method should return the shape with the largest area. Note that Shape implements the Comparable interface. Different Shapes should be compared using their area. Now if we extend the Shape hierarchy to include more shapes, say rhombus, then will your method run without any problems?