Problem Set #6 - Deadline 10.00 PM - 07th October, 2013

OOP Exercises

Exercises on Classes and Instances

Exercise 1: The Circle Class

A class called circle is designed as shown in the following class diagram. It contains:

- Two private instance variables: radius (of type double) and color (of type String), with default value of 1.0 and "red", respectively.
- Two overloaded constructors;
- Two public methods: getRadius() and getArea().

The source codes for Circle is as follows:

```java
public class Circle {
    // private instance variable, not accessible from outside this class
    private double radius;
    private String color;

    // 1st constructor, which sets both radius and color to default
    public Circle() {
        radius = 1.0;
        color = "red";
    }

    // 2nd constructor with given radius, but color default
    public Circle(double r) {
        radius = r;
        color = "red";
    }

    // A public method for retrieving the radius
    public double getRadius() {
        return radius;
    }

    // A public method for computing the area of circle
    public double getArea() {
        return radius*radius*Math.PI;
    }
}
```

Compile "Circle.java". Can you run the Circle class? Why? This Circle class does not have a main() method. Hence, it cannot be run directly. This Circle class is a "building block" and is meant to be used in another program. Let us write a test program called TestCircle which uses the Circle class, as follows:

```java
public class TestCircle {
    public static void main(String[] args) {
        // Declare and allocate an instance of class Circle called c1
        // with default radius and color
        Circle c1 = new Circle();
        // Use the dot operator to invoke methods of instance c1.
    }
}
```
System.out.println("The circle has radius of "+ c1.getRadius() + " and area of "+ c1.getArea());

// Declare and allocate an instance of class circle called c2
// with the given radius and default color
Circle c2 = new Circle(2.0);
// Use the dot operator to invoke methods of instance c2.
System.out.println("The circle has radius of "+ c2.getRadius() + " and area of "+ c2.getArea());
}

Now, run the TestCircle and study the results.
TRY:

1. **Constructor:** Modify the class Circle to include a third constructor for constructing a Circle instance with the given radius and color.

   ```java
   public Circle(double r, String c) {
   ...
   }
   ```

   Modify the test program TestCircle to construct an instance of Circle using this constructor.

2. **Getter:** Add a getter for variable color for retrieving the color of a Circle instance.

   ```java
   public String getColor() {
   ...
   }
   ```

   Modify the test program to test this method.

3. **Setter:** Is there a need to change the values of radius and color of a Circle instance after it is constructed? If so, add two public methods called setters for changing the radius and color of a Circle instance as follows:

   ```java
   // Setter for instance variable radius
   public void setRadius(double r) {
   radius = r;
   }

   // Setter for instance variable color
   public void setColor(String c) {
   ...
   }
   ```

   Modify the TestCircle to test these methods, e.g.,

   ```java
   Circle c3 = new Circle(); // construct an instance of Circle
c3.setRadius(5.0); // change radius
c3.setColor(...); // change color
   ```

4. **Keyword "this":** Instead of using variable names such as r (for radius) and c (for color) in the methods' arguments, it is better to use variable names radius (for radius) and color (for color) and use the special keyword "this" to resolve the conflict between instance variables and methods' arguments. For example,

   ```java
   // Instance variable
   private double radius;

   // Setter of radius
   public void setRadius(double r) {
   this.radius = r; // "this.radius" refers to the instance variable
   // "radius" refers to the method's argument
   }
   ```

   Modify ALL the constructors and setters in the Circle class to use the keyword "this".

5. **Method toString():** Every well-designed Java class should contain a public method called toString() that returns a short description of the instance (in a return type of String). The toString() method can be called explicitly (via `instanceName.toString()`) just like any other method; or implicitly through println(). If an instance is passed to
the println(anInstance) method, the toString() method of that instance will be invoked implicitly. For example, include the following toString() methods to the Circle class:

```java
public String toString() {
    return "Circle: radius=\" + radius + \" color=\" + color; 
}
```

Try calling toString() method explicitly, just like any other method:

```java
Circle c1 = new Circle(5.0);
System.out.println(c1.toString()); // explicit call
System.out.println("Operator '+' invokes toString() too: " + c2); // '+' invokes toString() too
```

For Correct Program code listing-[25 Points]

**Exercise 2: The Author and Book Classes**

A class called Author is designed as shown in the class diagram. It contains:

- Three private instance variables: name (String), email (String), and gender (char of either 'm' or 'f');
- One constructor to initialize the name, email and gender with the given values;
- public getters/setters: getName(), getEmail(), setEmail(), and getGender();
- A toString() method that returns "author-name (gender) at email", e.g., "Tan Ah Teck (m) at ahTeck@somewhere.com".

Write the Author class. Also write a test program called TestAuthor to test the constructor and public methods. Try changing the email of an author, e.g.,

```java
Author anAuthor = new Author("Tan Ah Teck", "ahteck@somewhere.com", 'm');
System.out.println(anAuthor); // call toString()
anAuthor.setEmail("paul@nowhere.com")
System.out.println(anAuthor);
```
A class called Book is designed as shown in the class diagram. It contains:

Four private instance variables: name (String), author (of the class Author you have just created, assume that each book has one and only one author), price (double), and qtyInStock (int);

Two constructors:

- public Book (String name, Author author, double price) {...}
- public Book (String name, Author author, double price, int qtyInStock) {...}

- public methods getName(), getAuthor(), getPrice(), setPrice(), getQtyInStock(), setQtyInStock().
- toString() that returns "book-name by author-name (gender) at email".

(Take note that the Author's toString() method returns "author-name (gender) at email").

Write the class Book (which uses the Author class written earlier). Also write a test program called TestBook to test the constructor and public methods in the class Book. Take note that you have to construct an instance of Author before you can construct an instance of Book. E.g.,

Author anAuthor = new Author(......);
Book aBook = new Book("Java for dummy", anAuthor, 19.95, 1000);
// Use an anonymous instance of Author
Book anotherBook = new Book("more Java for dummy", new Author(......), 29.95, 888);

Take note that both Book and Author classes have a variable called name. However, it can be differentiated via the referencing instance. For a Book instance say aBook, aBook.name refers to the name of the book; whereas for an Author's instance say anAuthor, anAuthor.name refers to the name of the author. There is no need (and not recommended) to call the variables bookName and authorName.

TRY:

1. Printing the name and email of the author from a Book instance. (Hint: aBook.getAuthor().getName(), aBook.getAuthor().getEmail()).
2. Introduce new methods called getAuthorName(), getAuthorEmail(), getAuthorGender() in the Book class to return the name, email and gender of the author of the book. For example,

   public String getAuthorName() { ...... }