CMSC 131 Spring 2005 Quiz 5 Worksheet

The fifth Quiz of the course will be on Wednesday March 9 in lab session.

The following list provides more information about the quiz:

- You will have 15 minutes to complete the quiz.
- It will be a written quiz (not using any computer).
- It will be closed-book, closed-notes, and no calculator is allowed.
- Answers must be neat and legible. We recommend that you use pencil and eraser.
- The quiz will be based on the exercises you will find below. The quiz will ask you to write a class for a particular application.

The following exercises subsume the material to be covered in Quiz #5. Solutions to these exercises will not be provided, but you are welcome to discuss your solutions with TAs and instructors during office hours.

Problem 1

For this problem you will write a complete Java class called Triangle, which represents a mathematical triangle. The class has the following private instance variables:

- base – double value representing the base of a triangle
- height – double value representing the height of the rectangle

All the following methods are non-static except the method named area.

1. Default Constructor – Initializes a triangle with base and height values equal to 1.
2. Constructor – This constructor takes two formal parameters representing the base and height respectively. The parameters are used to initialize the corresponding instance variables.
3. getBase – This method returns the base value.
4. getHeight – This method returns the height value.
5. setBase – This method updates the base value with the single parameter.
6. setHeight – This method updates the height value with the single parameter.
7. toString – Returns a String with the string “Triangle:” followed by base and height of the triangle (separated by a blank space).
8. equals – Boolean method that tests whether the current object has the same base and height as another Triangle, which is given as the single parameter.
9. tripleBase – Returns a reference to a new Triangle whose base is three times the base of the current object. The height of the new object will be the current object’s height. The current object will not be modified.
10. area – This is a static method that takes as formal parameter a Triangle reference and returns a double representing the area of the triangle.
Problem 2

Define a Student class that represents a college student. A student has a name (String reference), a gpa (double), and an id (integer). The class has the following methods that you must consider non-static unless indicated otherwise.

1. Default constructor – Initializes the name, gpa, and id variables with the values “NONAME”, 0.0, and -1 respectively.
2. Constructor – Takes a name, gpa, and id values as parameters and initializes the current object accordingly.
3. equals – Determines whether the current object and the single parameter value represent the same student.
4. increaseGPA – Increases the student gpa by the single parameter value.
5. toString – Generates a String with all the values associated with a student object using the format: “Student: “ followed by the name, gpa and id (each separated by a blank character).
6. Write the appropriate accessor and mutators methods associated with the class. For example, getId is the accessor for id and setId the mutator.
7. Write a copy constructor for your Student class.
8. Write a method called notEquals which uses the equals method you defined above.
9. Write a static method called total that returns the number of Student object instances that have been created. Feel free to add any instance variables that will allow you to keep track of the number of Student objects created.

Problem 3

Define a class called Scores. The class keeps track of scores associated with a student. The class has the following instance variables:

1. name – String representing the name of the student.
2. sum – double representing the sum of the scores.
3. numberScores – the current number of scores accumulated in the sum instance variable.

The class has the following non-static methods:

1. Constructor – Takes as parameter a String reference representing the name of the student. Initializes the sum and numberScores to 0.
2. getName – Returns the student’s name.
3. validScore – private method that determines whether a score is valid. A score is considered valid if it is in the range 0 → 100.00
4. **addScore** – Increases the sum by the single parameter value if the parameter represents a valid score.

5. **numGrade** – It has no parameters. It returns the numeric grade of the student. The numeric grade is the average of the scores.

6. **letterGrade** – It has no parameters. It returns the letter grade associated with the numeric grade. This method calls the numGrade to calculate the numeric grade. The cutoffs for letter grades are as follows:

   90.0 → A, 80.0 → B, 70.0 → C, other → X

7. **toString** – Generates a String with the name of the student followed by the numeric grade and the letter grade.

8. Write a main method (which is part of this class) that shows how you can use the class you designed.

**Problem 4**

A class called **Complex** represents a complex number. A complex number has a real and imaginary part. An example of a complex number is 5 + 6i where 5 represents the real part and 6i the imaginary part. Define a Complex class that implements the following methods. All the methods are **non-static** unless indicated otherwise. You can use two double type variables to represent the necessary instance variables.

1. **Default Constructor** – Initializes the current object to 0 + 0 i
2. **Constructor (one parameter)** – Initializes the current object to a complex number with a real part corresponding to the single parameter and an imaginary part set to 0.
3. **Constructor (two parameters)** – The parameters are used to initialize the real and imaginary part of the current object respectively.
4. **add** – Returns a reference to a new Complex object which represents the addition of the current object and the single Complex parameter. The current object will not be modified.
5. **subtraction** – Similar to add but performs subtration.
6. **multiply** – Multiples the real and imaginary parts by the single parameter (of double type).
7. **toString** – Returns a string using the format “x + yi”, where x is the real part and y the imaginary part.
8. **equals** – **static** method that takes two Complex objects and returns true if they are considered the same complex number.
9. Write a main method that illustrates how to use the class you designed.
Problem 5

Note: This problem was last semester’s quiz. A solution has been provided at the end.

Write a complete Java class called Circle, which represents a mathematical circle. The class has only one private instance variable name radius which represents the radius of the circle. All the methods of the class are non-static except the method named area. The methods you must implement are:

1. Default Constructor – Initializes the radius to 1.0.
2. Constructor – Initializes the radius with the single parameter value (of type double).
3. getRadius – Returns the radius.
4. increaseRadius – Increases the radius by the single parameter value (of type double).
5. toString – Returns a string with the string “Radius: ” followed by the radius value.
6. equals – Boolean method that tests whether the current object has the same radius as another Circle, which is given as the single parameter.
7. doubleRadius – Returns a reference to a new Circle whose radius is twice the radius of the current object. The current object is not modified.
8. area – A static method that returns the area of the Circle given as the single parameter. The area of a circle is defined as ($\pi * radius * radius$). You can use the Java constant Math.PI for the value of $\pi$.

Restrictions/Assumptions

1. You don’t need to use meaningful variable names; however you must use good indentation.
2. You don’t need to provide comments.
3. You cannot add any other instance variables or methods.

Problem 5, One possible solution

```java
public class Circle {
    private double radius;

    public Circle() {
        radius = 1.0;
    }

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

    public double getRadius() {
        return radius;
    }
}
```
public void increaseRadius(double delta) {
    radius += delta;
}

public String toString() {
    return "Radius: " + radius;
}

public boolean equals(Circle circleIn) {
    if (radius == circleIn.radius)
        return true;
    else
        return false;
}

public Circle doubleRadius() {
    return (new Circle(2 * radius));
}

public static double area(Circle circleIn) {
    double r = circleIn.radius;
    return Math.PI * r * r;
}

public static void main(String[] args) {
    Circle c1 = new Circle();
    System.out.println(c1);

    Circle c2 = new Circle(20.0);
    c2.increaseRadius(10);
    System.out.println("After increasing: " + c2.getRadius());

    String message = "different";
    if (c1.equals(c2))
        message = "Same";
    System.out.println(message);

    Circle c3 = new Circle(4);
    Circle c4 = new Circle(5);
    Circle c5 = c3.doubleRadius();
    System.out.println("Doubling " + c5);

    System.out.println("Area: " + Circle.area(c3));
}