Jinan University

Java Programming Lab Report

Major: Computer Science & Technology
School: International School

Student name : _____
(p.s. your name on JNU academic system)

Student number : _____

Date of Submission (mm-dd-yyyy): _____

Instructor: Yuxia Sun

Table of Content

LAB 7 DATE: 5/23/2023

Student Name:_____ Student ID: _____

Problem 1. (11.1)

11.1 (*The* **`Triangle`** *class*) Design a class named **`Triangle`** that extends **`GeometricObject`**. The class contains:

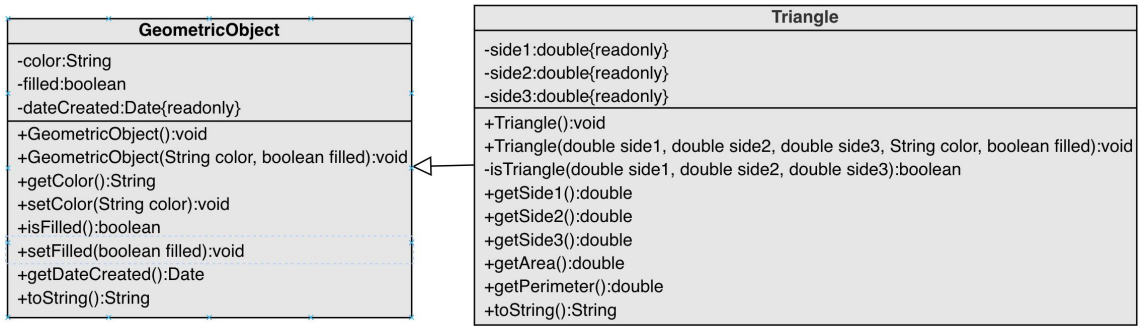
- Three **`double`** data fields named **`side1`**, **`side2`**, and **`side3`** with default values **`1.0`** to denote three sides of a triangle.
- A no-arg constructor that creates a default triangle.
- A constructor that creates a triangle with the specified **`side1`**, **`side2`**, and **`side3`**.
- The accessor methods for all three data fields.
- A method named **`getArea()`** that returns the area of this triangle.
- A method named **`getPerimeter()`** that returns the perimeter of this triangle.
- A method named **`toString()`** that returns a string description for the triangle.

For the formula to compute the area of a triangle, see Programming Exercise 2.19. The **`toString()`** method is implemented as follows:

```
return "Triangle: side1 = " + side1 + " side2 = " + side2 +
  " side3 = " + side3;
```

Draw the UML diagrams for the classes **`Triangle`** and **`GeometricObject`** and implement the classes. Write a test program that prompts the user to enter three sides of the triangle, a color, and a Boolean value to indicate whether the triangle is filled. The program should create a **`Triangle`** object with these sides and set the **`color`** and **`filled`** properties using the input. The program should display the area, perimeter, color, and true or false to indicate whether it is filled or not.

**** Source Code / Solution :***




```
import java.util.Date;

public class GeometricObject {
```

```java
    private String color;
    private boolean filled;
    private final Date dateCreated;

    GeometricObject() {
        this.color = "white";
        this.filled = false;
        this.dateCreated = new Date();
    }

    GeometricObject(String color, boolean filled) {
        this.color = color;
        this.filled = filled;
        this.dateCreated = new Date();
    }

    public String getColor() {
        return this.color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public boolean isFilled() {
        return this.filled;
    }

    public void setFilled(boolean filled) {
        this.filled = filled;
    }

    public Date getDateCreated() {
        return this.dateCreated;
    }

    public String toString() {
        return "created on " + dateCreated + "\ncolor: " + color + " and filled: " +
filled;
    }
}
```

```java
public class Triangle extends GeometricObject {
    private final double side1;
    private final double side2;
    private final double side3;

    Triangle() {
        super();
        this.side1 = 1.0;
        this.side2 = 1.0;
        this.side3 = 1.0;
    }
```

```java
    Triangle(double side1, double side2, double side3, String color, boolean filled) {
        super(color, filled);
        if (!isTriangle(side1, side2, side3)) {
            throw new IllegalArgumentException("These 3 side cannot form a triangle!");
        }
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    private boolean isTriangle(double side1, double side2, double side3) {
        return side1 + side2 > side3 &&
                side1 + side3 > side2 &&
                side2 + side3 > side1;
    }

    public double getSide1() {
        return side1;
    }

    public double getSide2() {
        return side2;
    }

    public double getSide3() {
        return side3;
    }

    public double getArea() {
        double factor = this.getPerimeter() / 2;
        return Math.sqrt(factor * (factor - side1) * (factor - side2) * (factor - side3));
    }

    public double getPerimeter() {
        return this.side1 + this.side2 + this.side3;
    }

    @Override
    public String toString() {
        return "Triangle: side1 = " + side1 + " side2 = " + side2 +
                " side3 = " + side3;
    }
}
```

```java
import java.util.Scanner;

public class TriangleTest {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        double[] sides = new double[3];
        String color;
        boolean isFilled;
```

```java
        System.out.println("Enter the properties of a triangle:");
        System.out.print("Enter side1 length: ");
        sides[0] = input.nextDouble();
        System.out.print("Enter side2 length: ");
        sides[1] = input.nextDouble();
        System.out.print("Enter side3 length: ");
        sides[2] = input.nextDouble();
        System.out.print("Enter its color: ");
        color = input.next();
        System.out.print("Enter whether the triangle is filled: ");
        isFilled = input.nextBoolean();
        input.close();

        Triangle triangle = new Triangle(sides[0], sides[1], sides[2], color, isFilled);

        System.out.println(
                "The triangle's area is " + triangle.getArea() +
                        ", its perimeter is " + triangle.getPerimeter() +
                        "\nIts color is " + triangle.getColor() +
                        ", is it filled? " + triangle.isFilled()
        );
    }
}
```

**** Output:***

```
TriangleTest
/Users/h3art/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java
Enter the properties of a triangle:
Enter side1 length: 2
Enter side2 length: 3
Enter side3 length: 4
Enter its color: red
Enter whether the triangle is filled: true
The triangle's area is 2.9047375096555625, its perimeter is 9.0
Its color is red, is it filled? true

进程已结束,退出代码0
```

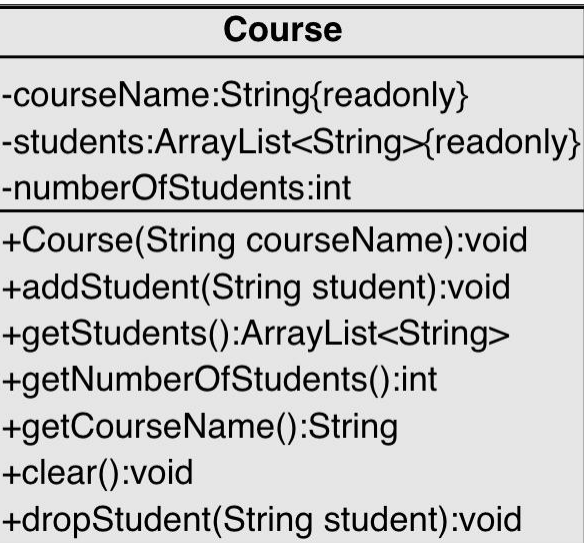
**** Debugging/Testing:***

Bug1: A call to super() method in an inherited subclass must be the first statement in the constructor body.
Fix: Move the super() method call to the first statement in the constructor body.

Problem 2. (11.5)

11.5 (*The* **Course** *class*) Rewrite the **Course** class in Listing 10.6. Use an **ArrayList** to replace an array to store students. Draw the new UML diagram for the class. You should not change the original contract of the **Course** class (i.e., the definition of the constructors and methods should not be changed, but the private members may be changed.)

**** Source Code / Solution :***




```java
import java.util.ArrayList;

public class Course {
    private final String courseName;
    private final ArrayList<String> students = new ArrayList<>();
    private int numberOfStudents;

    public Course(String courseName) {
        this.courseName = courseName;
    }

    public void addStudent(String student) {
        students.add(student);
        numberOfStudents++;
    }

    public ArrayList<String> getStudents() {
        return new ArrayList<>(students);
    }

    public int getNumberOfStudents() {
        return numberOfStudents;
    }

    public String getCourseName() {
        return courseName;
    }
```

```java
    public void clear() {
        students.clear();
        numberOfStudents = 0;
    }

    public void dropStudent(String student) {
        students.remove(student);
        numberOfStudents--;
    }
}
```

```java
import java.util.ArrayList;

public class CourseTest {
    public static void main(String[] args) {
        Course course1 = new Course("Data Structures");
        Course course2 = new Course("Database Systems");

        course1.addStudent("Peter Jones");
        course1.addStudent("Brian Smith");
        course1.addStudent("Anne Kennedy");
        course1.addStudent("Susan Kennedy");
        course1.addStudent("John Kennedy");
        course1.addStudent("Kim Johnson");
        course1.addStudent("S1");
        course1.addStudent("S2");
        course1.addStudent("S3");
        course1.addStudent("S4");
        course1.addStudent("S5");
        course1.addStudent("S6");
        course1.addStudent("S7");

        course2.addStudent("Peter Jones");
        course2.addStudent("Steve Smith");

        System.out.println("Number of students in course1: "
                + course1.getNumberOfStudents());
        ArrayList<String> students = course1.getStudents();
        for (String student : students) System.out.print(student + ", ");

        System.out.println();
        System.out.println("Number of students in course2: "
                + course2.getNumberOfStudents());

        course1.dropStudent("S1");
        System.out.println("Number of students in course1: "
                + course1.getNumberOfStudents());

        students = course1.getStudents();

        for (int i = 0; i < course1.getNumberOfStudents(); i++)
            System.out.print(students.get(i) + (i < course1.getNumberOfStudents() - 1 ? ",
```

```
" : " "));

        course1.clear();
        System.out.println("\nNumber of students in course1: "
                + course1.getNumberOfStudents());
    }
}
```

**** Output:***

```
CourseTest ×
/Users/h3art/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java ...
Number of students in course1: 13
Peter Jones, Brian Smith, Anne Kennedy, Susan Kennedy, John Kennedy, Kim Johnson, S1, S2, S3, S4, S5, S6, S7,
Number of students in course2: 2
Number of students in course1: 12
Peter Jones, Brian Smith, Anne Kennedy, Susan Kennedy, John Kennedy, Kim Johnson, S2, S3, S4, S5, S6, S7
Number of students in course1: 0

进程已结束,退出代码0
```

**** Debugging/Testing:***

Bug1: Forget to update the numberOfStudents variable when the clear() and dropStudent() methods are executed.
Fix: Update the numberOfStudents variable at the end of these two methods.

Problem 3. (11.11)

11.11 (*Sort* **ArrayList**) Write the following method that sorts an **ArrayList** of numbers:

```
public static void sort(ArrayList<Integer> list)
```

Write a test program that prompts the user to enter five numbers, stores them in an array list, and displays them in increasing order.

**** Source Code / Solution :***

```
import java.util.ArrayList;

public class SortArrayList {
    public static void sort(ArrayList<Integer> list){
        list.sort(null);
    }
}
```

```
import java.util.ArrayList;
import java.util.Scanner;

public class SortArrayListTest {
    private final static int LIMIT = 5;

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
```

```java
        ArrayList<Integer> arrayList = new ArrayList<>();

        System.out.println("Please enter 5 numbers(integer):");

        for (int i = 0; i < LIMIT; i++){
            arrayList.add(input.nextInt());
        }

        input.close();

        System.out.println("You have already enter 5 numbers. They are:");

        System.out.println(arrayList);

        SortArrayList.sort(arrayList);

        System.out.println("Sort these numbers in increasing order, the result is:");

        System.out.println(arrayList);
    }
}
```

**** Output:***

```
SortArrayListTest
/Users/h3art/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java
Please enter 5 numbers(integer):
114514 1919810 65536 255 2021102716
You have already enter 5 numbers. They are:
[114514, 1919810, 65536, 255, 2021102716]
Sort these numbers in increasing order, the result is:
[255, 65536, 114514, 1919810, 2021102716]

进程已结束,退出代码0
```

**** Debugging/Testing:***

Bug1: The sorting implementation of ArrayList is wrong, resulting in IndexOutOfBoundsException.
Fix: Change the implementation to the sort method built into ArrayList.

Problem 4. (11.13)(Optional)

***11.13** (*Remove duplicates*) Write a method that removes the duplicate elements from an array list of integers using the following header:

```
public static void removeDuplicate(ArrayList<Integer> list)
```

Write a test program that prompts the user to enter 10 integers to a list and displays the distinct integers in their input order and separated by exactly one space. Here is a sample run:

```
Enter 10 integers: 34 5 3 5 6 4 33 2 2 4 ↵Enter
The distinct integers are 34 5 3 6 4 33 2
```

**** Source Code / Solution :***

```java
import java.util.ArrayList;

public class RemoveDuplicates {
    public static void removeDuplicate(ArrayList<Integer> list) {
        ArrayList<Integer> record = new ArrayList<>();

        for(int originElement: list){
            boolean duplicate = false;

            for(int recordElement: record){
                if (originElement == recordElement) {
                    duplicate = true;
                    break;
                }
            }

            if(!duplicate){
                record.add(originElement);
            }
        }

        list.clear();
        list.addAll(record);
    }
}
```

```java
import java.util.ArrayList;
import java.util.Scanner;

public class RemoveDuplicatesTest {
    private final static int LIMIT = 10;

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        ArrayList<Integer> arrayList = new ArrayList<>();

        System.out.print("Enter 10 integers: ");
```

```java
        for (int i = 0; i < LIMIT; i++) {
            arrayList.add(input.nextInt());
        }

        input.close();

        RemoveDuplicates.removeDuplicate(arrayList);

        System.out.print("The distinct integers are ");

        for (int result : arrayList) {
            System.out.print(result + " ");
        }
    }
}
```

**** Output:***

```
RemoveDuplicatesTest ×
/Users/h3art/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java
Enter 10 integers: 34 5 3 5 6 4 33 2 2 4
The distinct integers are 34 5 3 6 4 33 2
进程已结束,退出代码0
```

**** Debugging/Testing:***

Bug1: The new ArrayList created in removeDuplicate() records non-duplicate elements, but the original ArrayList remains unchanged, and the result is lost.
Fix: After constructing the ArrayList with no duplicate elements in removeDuplicate(), clear the original ArrayList and copy the result into it.

Problem 5. (11.17)(Optional)

****11.17** (*Algebra: perfect square*) Write a program that prompts the user to enter an integer **m** and find the smallest integer **n** such that **m * n** is a perfect square. (*Hint:* Store all smallest factors of **m** into an array list. **n** is the product of the factors that appear an odd number of times in the array list. For example, consider **m** = 90, store the factors 2, 3, 3, and 5 in an array list. 2 and 5 appear an odd number of times in the array list. Thus, **n** is 10.) Here is a sample run of the program:

```
Enter an integer m: 1500 ↵Enter
The smallest number n for m * n to be a perfect square is 15
m * n is 22500
```

```
Enter an integer m: 63 ↵Enter
The smallest number n for m * n to be a perfect square is 7
m * n is 441
```

**** Source Code / Solution :***

```java
import java.util.ArrayList;
import java.util.Scanner;

public class PerfectSquare {
    /*
    Store all smallest factors of m into an array list.
    n is the product of the factors that appear an odd number of times
    in the array list.
     */
    private static final ArrayList<Integer> primeRecord = new ArrayList<>();
    private static final ArrayList<Integer> exponentRecord = new ArrayList<>();

    private static void findPrimeFactor(int num) {
        for (int i = 2; i <= num; i++) {
            if (num % i == 0) {
                int expo = 0;
                while (num % i == 0) {
                    num /= i;
                    expo++;
                }
                primeRecord.add(i);
                exponentRecord.add(expo);
            }
        }
    }

    private static int findFactorN() {
        int limit = primeRecord.size(), result = 1;

        for (int i = 0; i < limit; i++) {
            if (exponentRecord.get(i) % 2 != 0) {
```

```
                result *= primeRecord.get(i);
            }
        }

        return result;
    }

    public static void main(String[] args) {
        int m, n;
        Scanner input = new Scanner(System.in);
        System.out.print("Enter an integer m: ");
        m = input.nextInt();
        input.close();

        findPrimeFactor(m);
        n = findFactorN();

        System.out.println("The smallest number n for m * n to be a perfect square is " +
n);
        System.out.println("m * n is " + m * n);
    }
}
```

**** Output:***

```
PerfectSquare
/Users/h3art/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java
Enter an integer m: 1500
The smallest number n for m * n to be a perfect square is 15
m * n is 22500

进程已结束,退出代码0
PerfectSquare
/Users/h3art/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java
Enter an integer m: 63
The smallest number n for m * n to be a perfect square is 7
m * n is 441

进程已结束,退出代码0
```

**** Debugging/Testing:***

Bug1: The algorithm of prime factorization is wrong: it is thought that only prime factors less than or equal to m/2 need to be decomposed.
Fix: Expand the range of decomposed prime factors to [2, m].