Computer Laboratory Assignment 5
CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Design
February 21/22, 2017

0. Introduction.

In this lab assignment, you will extend some simple Java classes that represent plane figures from elementary geometry. The object of this assignment is not to do anything useful, but rather to demonstrate how methods can be inherited by extending classes. *You do not need interfaces for this assignment!*

1. Theory.

A *polygon* is a closed plane figure with three or more sides, all of which are line segments. The *perimeter* of a polygon is the sum of the lengths of its sides. A *rectangle* is a polygon with exactly four sides that meet at 90° angles. Like a polygon, it has a perimeter. It also has an *area,* the product of its base and height. A *square* is a rectangle whose sides are all the same length. Like a rectangle, it has a perimeter and an area.

Polygons, rectangles, and squares make up an *is-a hierarchy*. The hierarchy gets its name because a square *is-a* rectangle, and a rectangle *is-a* polygon. Is-a hierarchies can be easily modeled by Java classes using the `extends` keyword.

2. Implementation.

The following is the source code for a Java class whose instances represent polygons. The file **`Polygon.java`** on Moodle contains a copy of this source code. You will need it to complete the laboratory assignment.

```
class Polygon
{
  private int[] sideLengths;

  public Polygon(int sides, int ... lengths)
  {
    int index = 0;
    sideLengths = new int[sides];
    for (int length: lengths)
    {
      sideLengths[index] = length;
      index += 1;
    }
  }

  public int side(int number)
  {
    return sideLengths[number];
  }

  public int perimeter()
  {
    int total = 0;
    for (int index = 0; index < sideLengths.length; index += 1)
    {
      total += side(index);
    }
    return total;
```

```
  }
}
```

The class `Polygon` uses a private array called `sideLengths` to store the lengths of a polygon's sides. The array's length, `sideLengths.length`, is the number of sides that the polygon has. The class `Polygon` also has a has a public constructor and two public methods. To keep things simple, they do not check their arguments for correctness, as they would if `Polygon` was part of a real program.

The constructor takes four or more arguments and returns an instance of `Polygon` that represents a polygon. The first argument is the number of sides that the polygon has. The remaining arguments are the lengths of those sides. For example, the Java statement:

```
Polygon triangle = new Polygon(3, 3, 4, 5);
```

declares the variable `triangle` and sets it to an instance of `Polygon` that represents a triangle (because the first argument says it has 3 sides). The lengths of the triangle's sides are 3, 4, and 5.

The three dots '`...`' in the constructor mean that it can take zero or more extra integer arguments after its first argument. The `for`-loop with the colon visits the extra arguments one at a time. Don't worry if those parts of Java are unfamiliar. You don't have to know how the constructor works, only how to call it.

The method `side` returns the length of a polygon's side. Sides are numbered starting from 0. For example, the expression `triangle.side(0)` returns 3, the expression `triangle.side(1)` returns 4, and the expression `triangle.side(2)` returns 5.

The method `perimeter` returns a polygon's perimeter, the sum of the lengths of its sides. For example, the expression `triangle.perimeter()` returns 3 + 4 + 5 = 12.

For this assignment, you must write a class called `Rectangle`. As its name suggests, each instance of `Rectangle` must represent a rectangle. Along with a constructor, `Rectangle` must provide two methods, called `area` and `perimeter`. The method `area` must return the integer area of the rectangle, and the method `perimeter` must return the integer perimeter of the rectangle.

You must also write another class, called `Square`. As its name suggests, each instance of `Square` must represent a square. Along with a constructor, `Square` must provide two methods, called `area` and `perimeter`. The method `area` must return the integer area of the square, and the method `perimeter` must return the integer perimeter of the square.

The following driver program shows examples of how the constructors and methods of `Rectangle` and `Square` must work.

```
class Shapes
{
  public static void main(String[] args)
  {
    Rectangle wreck = new Rectangle(3, 5);  //  Make a 3 × 5 rectangle.
    System.out.println(wreck.area());       //  Print its area, 15.
    System.out.println(wreck.perimeter());  //  Print its perimeter, 16.

    Square nerd = new Square(7);            //  Make a 7 × 7 square.
    System.out.println(nerd.area());        //  Print its area, 49.
    System.out.println(nerd.perimeter()     //  Print its perimeter, 28.
  }
}
```

Your classes `Rectangle` and `Square` must use the `extends` keyword, so they will inherit methods from other classes. Also, *each class must inherit as many of its methods as possible from those other classes*. You will lose points for defining a method inside a class, if it could have been inherited from another class.

Here's a hint about how to write the constructors for `Rectangle` and `Square`. Suppose that a class `Triangle` extends the class `Polygon`. Then `Polygon` is the *superclass* of `Triangle`. The keyword `super` can be used to call the constructor that belongs to a superclass. For example, `Triangle`'s constructor, which takes the lengths of a triangle's three sides, might look like this.

```
public Triangle(int a, int b, int c)
{
  super(3, a, b, c);
}
```

It uses `Polygon`'s constructor to make a polygon with 3 sides, whose lengths are `a`, `b`, and `c`. If `super` is used in this way, then it must be the first statement in the constructor. You don't have to write `Triangle`—this was only an example!

3. Deliverables.

The file **`Test.java`** contains a driver class whose a `main` method performs 12 *public tests,* worth 1 point each. Each public test is a call to `println`, along with a comment that shows what it must print. To grade your work, the TA's will run the public tests using your `Rectangle` and `Square` classes. If a public test behaves exactly as it should, then you will receive 1 point for it.

In addition, the TA's will do 5 *private tests* on your `Rectangle` and `Square` classes. You will not be told what the private tests are, but they are worth 2 points each, and they determine if `Rectangle` and `Square` have inherited as many methods as possible. The TA's will do the same private tests for all students, and these tests will be made public only after all the work for this lab has been graded.

Your score for this lab is the sum of the points you get for the public tests, and the points you get for the private tests, for a maximum of 22 points. Here is what you must turn in.

1. Source code for the class `Rectangle`. Its instances must provide the methods `side`, `area` and `perimeter`. These methods are not necessarily defined in `Rectangle`: some or all may be inherited.

2. Source code for the class `Square`. Its instances must provide the methods `side`, `area` and `perimeter`. These methods are not necessarily defined in `Square`: some or all may be inherited.

Please submit source code for both classes in one file. If you do not know how to submit your work, please ask your lab TA. If your lab is on Tuesday, February 21, then your work must be submitted by Tuesday, February 28, at 11:55 PM. If your lab is on Wednesday, February 22, then your work must be submitted by Wednesday, March 1, at 11:55 PM. *To avoid late penalties, do not confuse these two dates!*