Program 2 – Shape System

The purpose of this assignment is to practice using the various concepts learned in Chapter 10, Sections 10.6 to 10.12 of our textbook. Specifically, this assignment will require knowledge of the following concepts.

- Aggregation (8.7, CPS141)
- Inheritance (10.1 to 10.5, CPS141)
- Polymorphism (10.7)
- Abstract Classes & Methods (10.8)
- Interfaces (10.9)
- Anonymous Inner Classes (10.10)

This program involves creating a simple command line application to test the Object-Oriented system of Shape classes to be built as described below. This application will not involve any user input but only an output report.

1 User Interface Specifications

The program should be written to meet the following User Interface specifications. This section corresponds to Item 3 of the "Program Rubric" at the end of this specification. Please refer to the sample output in Section 4. The following command-line interface will be implemented.

User Experience:

1. Display a user-friendly welcome message.

2. Display information for an instance of each shape type including Rectangle, Circle, RightTriangle, and IsoTriangle. Information for each shape should include the following:
    a. The shape type
    b. width (2 decimal places)
    c. height (2 decimal places)
    d. area (2 decimal places)
    e. perimeter (2 decimal places)

3. Display the shape with the smallest perimeter.

4. Display the shape with the largest area.

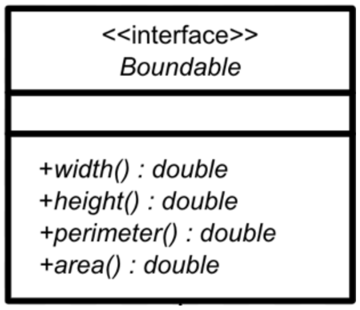
2 Java Implementation

The program should be implemented in Java using the Eclipse IDE as specified below. This section corresponds to Item 4 of the "Program Rubric". This program will require TEN Java (.java) files that must be uploaded to Brightspace. It is recommended that you work on each file separately and in the order that they are described below. It is also recommended that you create a "Playground.java" file to use for testing out the various components prior to putting them all together in the main program. If there is something you don't understand please reach out to me for help.

NOTE: you will need to create a separate .java file for each of the following items.

2.1 Boundable.java - Interface

This interface specifies four methods related to aspects of a 2D object that has a bounding box or size. The Boundable interface will be implemented by the Shape class described below.



2.1.1 Fields

None - interfaces do not declare instance fields.

2.1.2 Methods

`+width() : double`

Defines the header of a method that returns the width of a 2D Boundable object.

`+height() : double`

Defines the header of a method that returns the height of a 2D Boundable object.

`+perimeter() : double`

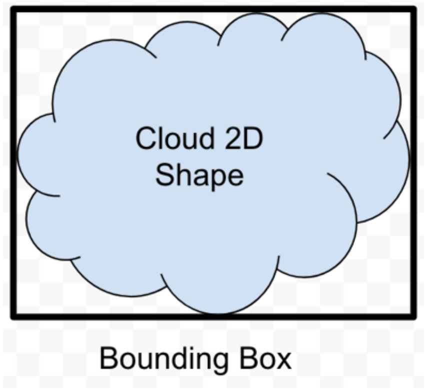
Defines the header of a method that returns the perimeter of a 2D Boundable object.

`+area() : double`

Defines the header of a method that returns the area of a 2D Boundable object.

2.2 BoundingBox.java - Concrete Class

The `BoundingBox` class is a simple concrete class that holds width and height values of a two-dimensional "bounding box" for shapes. A bounding box is the smallest 2D rectangle that will completely hold another 2D object. The figure at the right shows a Cloud 2D Shape inside its "Bounding Box". The `BoundingBox` class will represent the width and height of the Bounding Box for any shape we need to analyze.



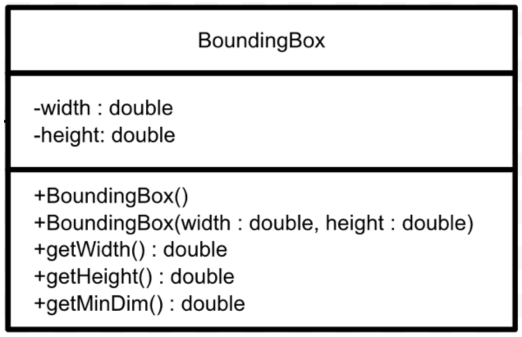
2.2.1 Fields

`-width : double`

The "x-axis" or "width" value of the BoundingBox object

`-height : double`

The "y-axis" or "height" value of the BoundingBox object



2.2.2 Constructors

`+BoundingBox()`

A no-arg constructor that sets the width and height fields to 0.0.

`+BoundingBox(width : double, height : double)`

A constructor that takes the width and height of the BoundingBox as double parameters. Use these parameters to set the width and height fields respectively.

2.2.3 Methods

Write only the following methods as shown in the UML diagram.

`+getWidth() : double`

Returns the value of the width field.

`+getHeight() : double`

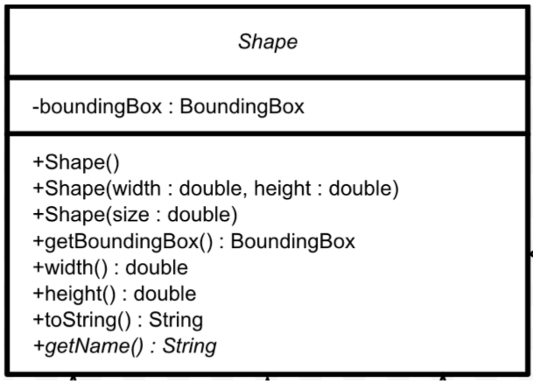
Returns the value of the height field.

`+getMinDim() : double`

Returns the minimum of the width and height fields. That is if width is less than height, width is returned and *vice versa*. HINT: you can write the body in one line with the “ternary” operator “<condition> ? <true value> : <false value>”

2.3 Shape.java - Abstract Class

The Shape class represents an “undifferentiated” or “unspecified” 2D Shape. It is an abstract class that partially implements the `Boundable` interface. It cannot be instantiated since it does not implement the `area` or `perimeter` methods defined in the `Boundable` interface. These methods must be implemented by specific Shape child classes such as the Rectangle and Circle concrete classes described later in the specification.



2.3.1 Class Header

The class header must be written to implement the `Boundable` interface.

2.3.2 Fields

Write only the following fields as shown in the UML diagram for Shape.

`-boundingBox : BoundingBox`

An instance of the `BoundingBox` class that will represent the 2D “bounding box” of the shape.

2.3.3 Constructors

`+Shape()`

A no-arg constructor that initializes the `boundingBox` field with a default `BoundingBox` instance using :

"`new BoundingBox()`".

`+Shape(width : double, height : double)`
Constructor that initializes the `boundingBox` field with an instance using

"`new BoundingBox(width,height)`".

`+Shape(size : double)`
Constructor that initializes the `boundingBox` field with an instance using

"`new BoundingBox(size)`".

2.3.4 Methods

Write only the following methods as shown in the UML diagram.

`+getBoundingBox() : BoundingBox`
Returns the value of the `boundingBox` field.

`+getName() : String`
An abstract method that will be implemented by sub-classes. Returns the name of the kind of shape.

`+width() : double`
Returns the value returned by the `boundingBox` field's `getWidth()` method.

`+height() : double`
Returns the value returned the `boundingBox` field's `getHeight()` method.

`+toString() : String`
This method will return a well-formatted String object to be used in the shape analysis report. Be sure to use the proper @Override annotation to indicate to the compiler that you are overriding this method.

NOTE: You may use the `StringBuilder` class, `String.format`, or any other method to create the `String` object to be returned by this method. When printed to the console this `String` should produce output similar to the following:

```
Rectangle
  width:          24.69
  height:         51.00
  area:        1,259.31
  perimeter:     151.38
```

DO NOT USE "`System.out.println`" anywhere in this method.

Get the name of the shape by calling the getName() method like:

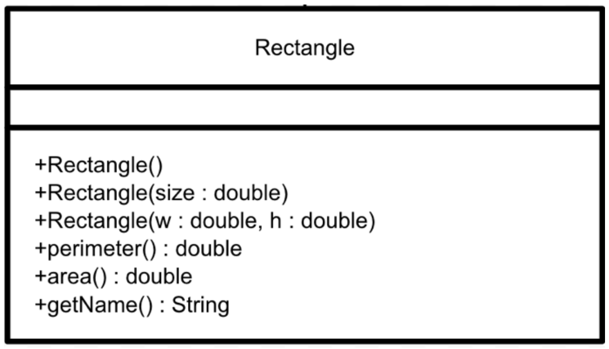
```
this.getName()
```

For example, here is a partial solution showing only the shape type and the width using the String.format method. You may use and update this code for all of the other information including height, area, and perimeter. Notice that the width value label "width:" is left justified (the minus sign after %) and uses 11 total spaces. The width value is displayed as floating point with 10 total characters and 2 decimal places.

```
return String.format("%s\n"
            + "  %-11s%,10.2f\n"
            this.getName(),
            "width:", this.width()
            );
```

2.4 Rectangle.java - Concrete Class

This is a concrete class that represents a two-dimensional Rectangle shape. <u>It inherits directly from the Shape abstract class described above</u>.



2.4.1 Fields

This class does not require any fields.

2.4.2 Constructors

`+Rectangle()`

The no-arg constructor for `Rectangle`. Directly call the no-arg superclass constructor inside. For example:

```
super();
```

`+Rectangle(size : double)`

A constructor taking a single, size, argument. Directly call the appropriate superclass constructor. For example:

```
super(size);
```

`+Rectangle(width : double, height : double)`

A constructor taking width and height arguments. Directly call the appropriate superclass constructor. For example:

```
    super(width, height);
```

2.4.3 Methods

`+perimeter() : double`

Returns the perimeter of the Rectangle object based on the following equation.

$$perimeter = 2(w + h)$$

Where:

w = `this.width()`
h = `this.height()`

```
Algorithm
perimeter = 2 * (w + h)
return perimeter
```

`+area() : double`

Returns the area of the Rectangle object based on the following equation.

$$area = wh$$

Where:

w = `this.width()`
h = `this.height()`

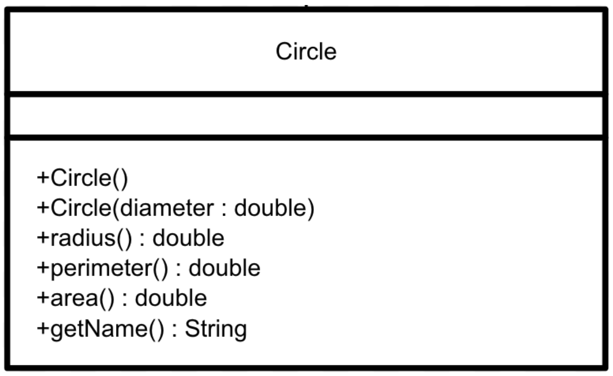
```
Algorithm
area = w * h
return area
```

`+getName() : String`

A concrete method that should simply return the String literal “Rectangle”.

2.5 Circle.java - Concrete Class

This is a concrete class that represents a two-dimensional Circle. It inherits directly from the Shape abstract class described above.



2.5.1 Fields

This class does not require any fields.

2.5.2 Constructors

`+Circle()`

The no-arg constructor for `Circle`. Directly call the no-arg superclass constructor inside. For example:

```
    super();
```

`+Circle(diameter : double)`

A constructor taking a single, diameter, argument. Directly call the appropriate superclass constructor. For example:

```
    super(diameter);
```

2.5.3 Methods

`+radius() : double`

Returns the radius of the `Circle` object based on the following equation.

$$radius = \frac{1}{2}\min(w, h)$$

Where:

$\min(w, h) =$ `this.getBoundingBox().getMinDim()`

Algorithm

```
radius = this.getBoundingBox().getMinDim() / 2
return radius
```

`+perimeter() : double`

Returns the perimeter of the Circle based on the following equation.

$$perimeter = 2\pi r$$

Where:

r = `this.radius()`

```
Algorithm
perimeter = 2 * Math.PI * r
return perimeter
```

`+area() : double`

Returns the area of the Circle based on the following equation.

$$area = \pi r^2$$

Where:

r = `this.radius()`

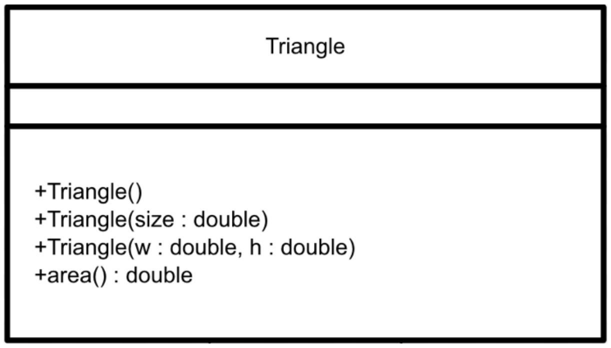
```
Algorithm
area = Math.PI * r * r
return area
```

`+getName() : String`

A concrete method that should simply return the String literal “Circle”.

2.6 Triangle.java - Abstract Class

This is an abstract class that represents a two-dimensional Triangle shape. It inherits directly from the Shape abstract class described above. A generic Triangle is abstract since, although we know how to calculate the area for any triangle, the calculation of the perimeter is different depending on the type of triangle.



2.6.1 Fields

This class does not require any fields.

2.6.2 Constructors

`+Triangle()`

The no-arg constructor for `Triangle`. Directly call the no-arg superclass constructor inside. For example:

```
super();
```

`+Triangle(size : double)`

A constructor taking a single, size, argument. Directly call the appropriate superclass constructor. For example:

```
    super(size);
```

`+Triangle(width : double, height : double)`

A constructor taking width and height arguments. Directly call the appropriate superclass constructor. For example:

```
    super(width, height);
```

2.6.3 Methods

`+area() : double`

Returns the area of the Rectangle object based on the following equation.

$$area = \frac{wh}{2}$$

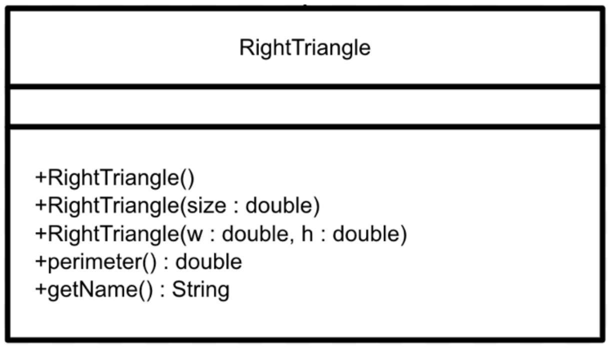
Where:

w = `this.width()`
h = `this.height()`

```
Algorithm
area = (w * h) / 2
return area
```

2.7 RightTriangle.java - Concrete Class

This is a concrete class that represents a two-dimensional RightTriangle shape. It inherits directly from the Triangle abstract class described above.



```
RightTriangle
---------------------------------------------
---------------------------------------------
+RightTriangle()
+RightTriangle(size : double)
+RightTriangle(w : double, h : double)
+perimeter() : double
+getName() : String
```

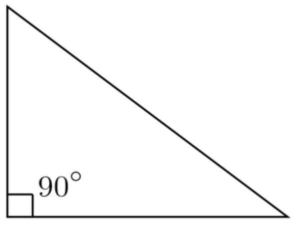
2.7.1 Fields

This class does not require any fields.

2.7.2 Constructors

`+RightTriangle()`

The no-arg constructor for `RightTriangle`. Directly call the no-arg superclass constructor inside. For example:



```
    super();
```

`+RightTriangle(size : double)`

A constructor taking a single, size, argument. Directly call the appropriate superclass constructor. For example:

```
    super(size);
```

```
+RightTriangle(width : double, height : double)
```

A constructor taking width and height arguments. Directly call the appropriate superclass constructor. For example:

```
    super(width, height);
```

2.7.3 Methods

```
+perimeter() : double
```

Returns the perimeter of the Rectangle object based on the following equation.

$$perimeter = w + h + \sqrt{w^2 + h^2}$$

Where:

w = `this.width()`
h = `this.height()`

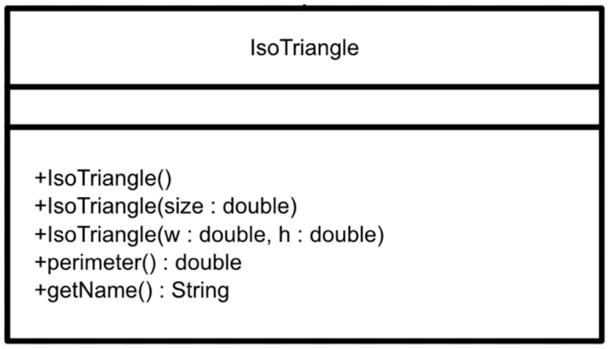
```
Algorithm
len = Math.sqrt(w*w + h*h)
perimeter = w + h + len
return perimeter
```

```
+getName() : String
```

A concrete method that should simply return the String literal “RightTriangle”.

2.8 IsoTriangle.java - Concrete Class

This is a concrete class that represents a two-dimensional Isosceles Triangle shape. <u>It inherits directly from the Triangle abstract class described above</u>.



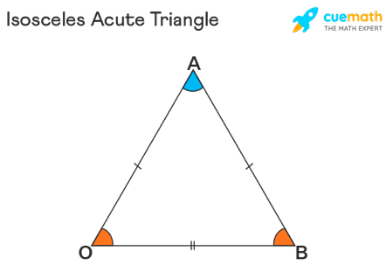
2.8.1 Fields

This class does not require any fields.

2.8.2 Constructors

`+IsoTriangle()`

The no-arg constructor for `IsoTriangle`. Directly call the no-arg superclass constructor inside. For example:



```
    super();
```

`+IsoTriangle(size : double)`

A constructor taking a single, size, argument. Directly call the appropriate superclass constructor. For example:

```
    super(size);
```

`+IsoTriangle(width : double, height : double)`

A constructor taking width and height arguments. Directly call the appropriate superclass constructor. For example:

```
    super(width, height);
```

2.8.3 Methods

`+perimeter() : double`

Returns the perimeter of the Rectangle object based on the following equation.

$$perimeter = w + 2\sqrt{\left(\frac{w}{2}\right)^2 + h^2}$$

Where:

w = `this.width()`
h = `this.height()`

```
Algorithm
w2 = w / 2
len = Math.sqrt(w2*w2 + h*h)
perimeter = w + (2 * len)
return perimeter
```

`+getName() : String`

A concrete method that should simply return the String literal "IsoTriangle".

2.9 Main.java

The Main.java is provided with the project but has several "TODO" items for you to complete. Please see the comments located with the file provided for further instructions. Your updated Main.java file must be included with your submission.

2.10 Comments

Your programs must always include a project documentation comment at the top of your main program file. For example.

```
/**
 * <Your Name Goes Here>
 * CPS142 – Fall 2024
 * Date: <the date you start goes here>
 * Instructor: Adam Divelbiss
 * Assignment: Program02
 * Purpose: <write the purpose of your program here>
 */
```

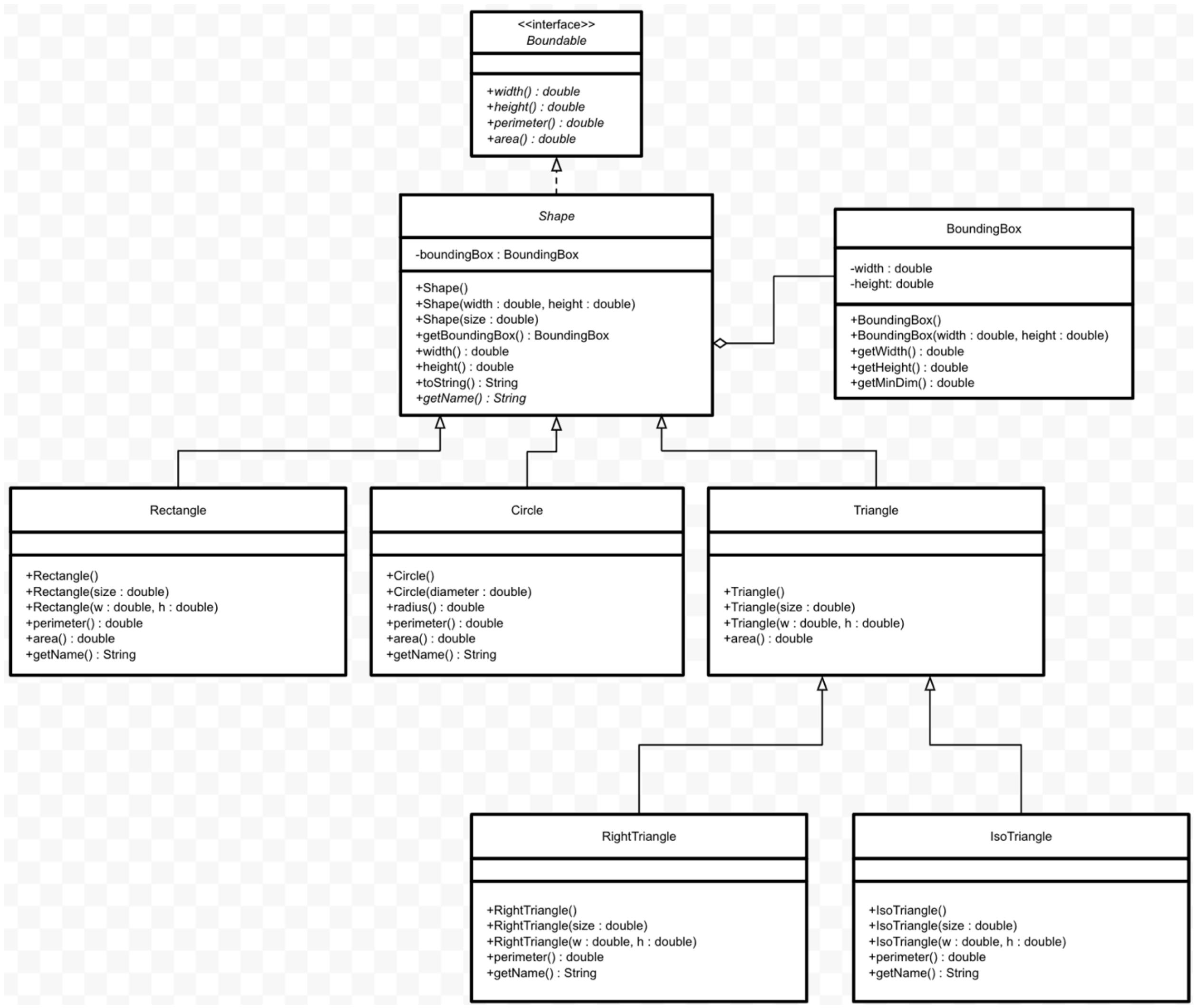
Where text between “<” “>” angle brackets is for you to fill in.

All classes and methods, except for methods that override well documented superclass methods, must have a documentation comment.

Also be sure to use a single line comment to document/describe each major section in the code.

2.11 Project UML Diagram

The diagram below shows the complete UML diagram for the project.



3 Deliverables

The deliverable files will be uploaded to the assignment in Brightspace. Deliverables consist of the following .java files. DO NOT SEND THE .class FILES. I CANNOT USE THEM.

- Main.java - Updated with your changes
- Boundable.java
- BoundingBox.java
- Shape.java
- Rectangle.java
- Circle.java
- Triangle.java
- RightTriangle.java
- IsoTriangle.java

The due date/time is:
Saturday, September 21, 2024, at 11:59PM

4 Sample Output

The following output demonstrates one possibility that would get full credit for Section 3 of the Program Rubric.

```
CPS142 – Program 2 – Shape System Tests

Shapes:
Rectangle
  width:          16.00
  height:         16.00
  area:          256.00
  perimeter:      64.00

Circle
  width:          16.00
  height:         16.00
  area:          201.06
  perimeter:      50.27

RightTriangle
  width:          16.00
  height:         16.00
  area:          128.00
  perimeter:      54.63

IsoTriangle
  width:          16.00
  height:         16.00
  area:          128.00
  perimeter:      51.78

Shape with smallest perimeter:
Circle
  width:          16.00
  height:         16.00
  area:          201.06
  perimeter:      50.27

Shape with largest area:
Rectangle
  width:          16.00
  height:         16.00
  area:          256.00
  perimeter:      64.00
```

5 General Submission Policy

You may submit your code multiple times up until the due date/time. Any submission after the due date/time will be subject to the late assignment policy described in the course Syllabus. In all cases the last code you submit will be used for the grade.

6 General Requirements

All code you write for this course should meet the following general requirements:

1. All code should be your own work and not copied from another student, the internet, or any solution guide for the textbook.

2. Code from the following sources may be used or modified to complete the project:
    a. Any code provided by me in the class notes.
    b. Any code provided by the textbook itself (excluding any solution guides).
    c. Any code written by you for this or another course.

3. All Java classes should have a documentation comment including, author, date, course (i.e., CPS142-Fall 2024), and a purpose or brief description statement.

4. All methods must have properly formatted and descriptive documentation comments. The only exception is methods that override well documented interface or superclass methods. Private methods must also be commented.

5. Any method with more than just a few lines of code should have additional single-line comments describing important parts of the algorithm.

6. The use of break, continue, or exit statements inside loops is not acceptable in the implementation of any class. If found points will be subtracted from Item 5 of the Programming Rubric.

NOTE: The above requirements should be followed only where applicable.

7 Program Rubric

Your grade for this assignment will be computed using the following rubric.

Item	Description	Percent of Grade
1	Compilation: Compiles with no syntax errors.	10
2	Execution: Runs with no crashes or runtime errors.	10
3	External Operation: Written to meet all requirements for input and output as specified or implied by the problem.	0
4	Internal Operation: Written to meet all requirements for internal operation as specified as specified or implied by the problem statement.	60
5	Coding Style: Written and formatted in accordance with best practices presented in class (indentation, class names, variable names, method names, etc.).	5
6	Project Comment: Contains a **documentation comment** with the assignment name, author, date, class (i.e., CPS142-Fall 2024), and purpose/description at the top of each file.	5
7	Documentation: All methods, classes, and interfaces have **documentation comments**. Methods that override well-documented interface method and the main method are exempt.	5
8	Comments: Includes sufficient other regular comments to describe important parts of the code.	5
	Total	**100**