Program 1 – Shape Analyzer

The purpose of this assignment is to practice using the various concepts learned in Chapters 9, 10, and 11 of our textbook. Specifically, this assignment may require use or knowledge of the following concepts.

- Keyboard input (CPS141)
- Reading data from a file (CPS141)
- Writing a basic class (CPS141)
- Overriding toString (CPS141)
- Aggregation (8.7, CPS141)
- Enumerated Types (8.9, CPS141)
- Inheritance (10.1 to 10.5, CPS141)
- More String methods (9.3)
- StringBuilder class (9.4)
- Tokenizing Strings (9.5)
- Numeric Wrapper Classes (9.6)
- Working with CSV data (9.7)
- Polymorphism (10.7)
- Abstract Classes & Methods (10.8)
- Interfaces (10.9)
- Anonymous Inner Classes (10.10)
- Functional Interfaces (10.11)
- Lambda Expressions (10.11)

This program involves creating a simple command line application to generate a report on an analysis of 10,000 2-dimensional shapes to display the shapes with the largest width, largest height, largest area, and largest perimeter. The report will also include confirmation of the number of valid shape objects in the database. This application will not involve any user input but only an output report.

1 User Interface Specifications

The entire user interface has already been written so that you can focus on the Object-Oriented implementation and other work related to writing Lambda expressions, Anonymous Inner Classes, and handling exceptions. You will be provided with the following file(s):

Main.java	An incomplete main program containing the user interface.
ShapeFactory.java	An incomplete utility class to create Shape objects.
shapes_20230701.csv	A CSV (comma separated value) file containing a database of shape objects.

Copy the Java files into the "src" folder of your Eclipse project. Place the CSV file in the root folder of your project.

Your first task is to create the Java Enumerations, Interfaces and Classes described in Section 2. DO NOT start your work in Main.java until ALL the other work has been completed.

2 Java Implementation

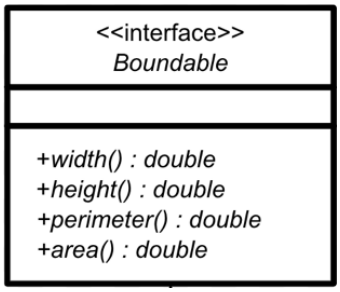
The program should be implemented in Java using the Eclipse IDE as specified below. This section corresponds to Item 3 and Item 4 of the "Program Rubric". This program will require ELEVEN Java (.java) files that must be uploaded to Brightspace. **It is recommended that you work on each file separately 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: Even though there are 11 files to work on, most of these files require very little code.

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.

For additional information see Section 10.9 of our Textbook.

2.1.1 Fields

None - interfaces do not declare instance fields.

2.1.2 Methods

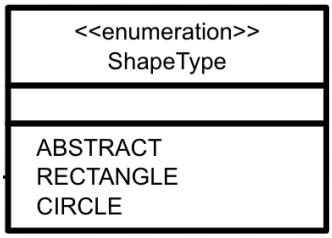
`+width() : double`
Defines the header of a method that returns the width of a 2D Boundable object.

`+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 ShapeType.java - Enumeration

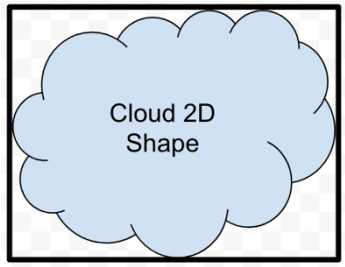


This is a Java enumeration with three values including `ABSTRACT`, `RECTANGLE`, and `CIRCLE`. Write the enumeration in its own file called ShapeType.java.

For additional information see Section 8.9 of our Textbook. Enumerated Types were covered in CPS141.

2.3 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.

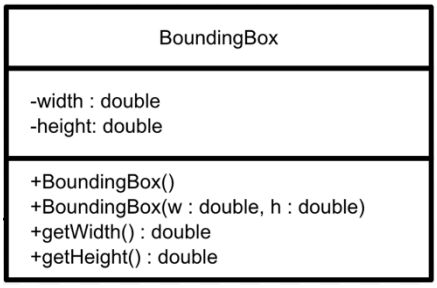


Bounding Box

2.3.1 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



2.3.2 Constructors

`+BoundingBox()`
A no-arg constructor that sets the width and height fields to 0.0.

`+BoundingBox(w : double, h : 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.

2.3.3 Methods

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

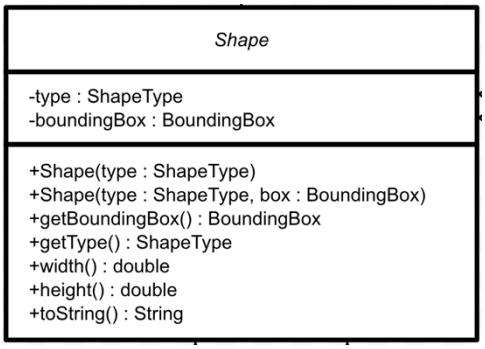
`+getWidth() : double`
Returns the value of the width field.

`+getHeight() : double`
Returns the value of the height field.

For additional information review Chapter 6 from CPS141 on how to create a simple class. This is a “Plaine Ole’ Java Object” (POJO).

2.4 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.



For additional information see Section 10.8 of our Textbook, on how to write an Abstract Class.

2.4.1 Class Header

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

2.4.2 Fields

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

`-type : ShapeType`
The type of the current shape set to one of the ShapeType enumeration constants.

`-boundingBox : BoundingBox`
An instance of the `BoundingBox` class that will represent the 2D "bounding box" of the shape.

2.4.3 Constructors

`+Shape(type : ShapeType)`
This constructor takes a ShapeType object as an argument.

Initialize the `type` instance field with the `type` parameter.

Initialize the `boundingBox` field with a default `BoundingBox` instance using `new BoundingBox()`.

`+Shape(type : ShapeType, boundingBox : BoundingBox)`
This constructor takes a ShapeType object and a BoundingBox object as arguments.

Initialize the `type` instance field with the `type` parameter.

Initialize the `boundingBox` field with the `boundingBox` parameter.

2.4.4 Methods

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

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

`+getType() : ShapeType`
Returns the value of the `type` field.

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

`+height() : double`
Returns the value of 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:

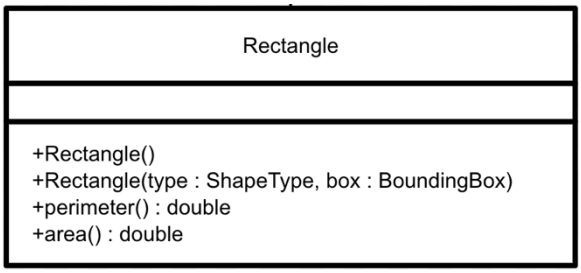
```
RECTANGLE
  width:          24.69
  height:         51.00
  area:        1,259.31
  perimeter:     151.38
```

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

2.5 Rectangle.java - Concrete Class

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

For additional information see Sections 10.1 to 10.5 of our Textbook on Inheritance, which was covered in CPS141.



2.5.1 Fields

This class does not require any fields.

2.5.2 Constructors

`+Rectangle()`

The no-arg constructor for `Rectangle`. Directly call the superclass constructor that accepts a ShapeType using the following an argument.

```
ShapeType.RECTANGLE
```

`+Rectangle(type : ShapeType, boundingBox : BoundingBox)`

This constructor takes a ShapeType object and a BoundingBox object as arguments.

Pass both parameters to the superclass, super constructor.

2.5.3 Methods

`+perimeter() : double`

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

$$= 2(\quad + h)$$

Where:

$= $ `this.width()`

$h =$ `this.height()`

`+area() : double`

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

$$= \quad h$$

Where:

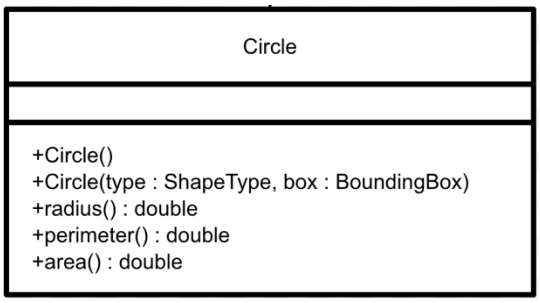
$= $ `this.width()`

$h =$ `this.height()`

2.6 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.

For additional information see Sections 10.1 to 10.5 of our Textbook on Inheritance, which was covered in CPS141.



2.6.1 Fields
This class does not require any fields.

2.6.2 Constructors
`+Circle()`
The no-arg constructor for `Circle`. Directly call the superclass constructor that accepts a ShapeType using the following an argument.

`ShapeType.`*`CIRCLE`*

`+Circle(type : ShapeType, boundingBox : BoundingBox)`
This constructor takes a ShapeType object and a BoundingBox object as arguments.

Pass both parameters to the superclass, super constructor.

2.6.3 Methods
`+radius() : double`
Returns the radius of the `Circle` object based on the following equation.

$$= \frac{}{2}$$

Where:
= `this.width()`

For this class, we are assuming that both width and height are the same so the diameter of the circle can be represented by either value.

`+perimeter() : double`
Returns the perimeter of the Circle based on the following equation.

$$= 2$$

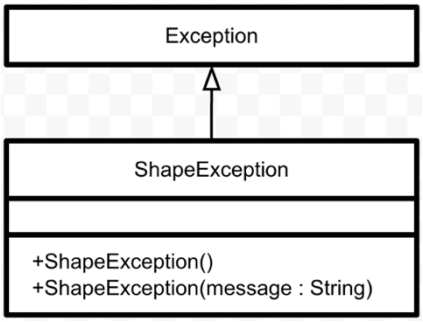
Where:
= `this.radius()`

`+area() : double`
Returns the area of the Circle based on the following equation.

$$= {}^{2}$$

Where:
= `this.radius()`

2.7 ShapeException

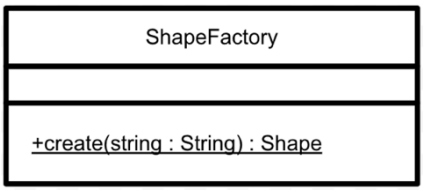


This class will implement our own custom Exception class that we can use if there are any errors related to handling Shape objects. This will be a very simple class definition consisting of only the two constructors shown in the UML diagram. Inside the constructors, simply call the appropriate super constructor.

This class will inherit directly from the Exception class.

For additional information see Section 11.1 of our Textbook on Exceptions.

2.8 ShapeFactory



This is a "helper" or "utility" class that has a single static method that converts a string into the corresponding `Shape` object. The `String` object expected by the create method will consist of three comma separated values including the `Shape` type, the width of the bounding box and the height of the bounding box in that order.

For example, the String: "CIRCLE,2.3,2.3"

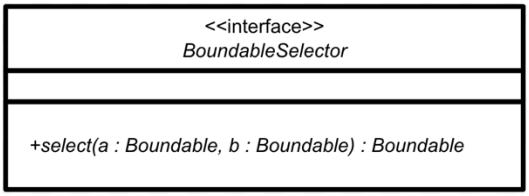
Would produce a `Circle` object with a bounding box, 2.3 wide by 2.3 tall. In this case, since the minimum dimension is 2.3, the

This class method will be used in the `Main` program to help create the array of objects loaded from the shapes database file.

Please see the partially completed ShapeFactory class provided with the assignment. There are several "TODO:" comments to complete.

For additional information see Sections 9.5 to 9.7 of our Textbook. Also review the lectures on handling CSV files.

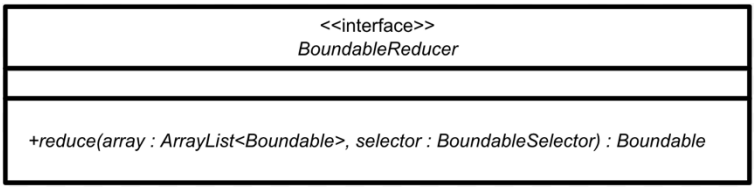
2.9 BoundableSelector.java - Interface



This is a functional interface (See Section 10.11) that defines a single method to "select" one `Boundable` object or the other by returning the selected `Boundable` object. This interface will be used in the main program to create lambda expressions to choose the `Boundable` with the larger `width`, `height`, `perimeter`, or `area`.

2.9.1 Fields
None - interfaces do not declare instance fields.

2.9.2 Methods
`+select(a : Boundable, b : Boundable) : Boundable`
Defines the header of a method that returns either "a" or "b" based on some criterion defined in the class or Lambda expression that implements it.

2.10 BoundableReducer.java - Interface


This is a functional interface (See Section 10.11) that defines a single method to "reduce" an `ArrayList` of Boundable objects to a single `Boundable` object based on a `BoundableSelector object` passed as the second argument. For example, if we wanted to find the Shape (or other `Boundable` object) with the smallest width in an `ArrayList` of `Shapes`, we can define a `BoundableSelector` that returns the `Boundable` with the smaller width. We then give both the `ArrayList` and the `BoundableSelector` for the smaller `width` to the `BoundableReducer`, which will then find the Shape with the smallest `width`.

We will be able to reuse the `BoundableReducer` to find the smallest (or largest) `area`, smallest (or largest) `perimeter`, and smallest (or largest) `height` by simply passing in different `BoundableSelector` objects written appropriately for each task.

This is a kind of "plug and play" system where we plug-in different `BoundableSelector` objects to achieve different results using the same basic code in the `BoundableReducer`.

2.10.1 Fields
None - interfaces do not declare instance fields.

2.10.2 Methods
`+reduce(array : ArrayList<Boundable>, selector : BoundableSelector) : Boundable`

Defines the header of a method that returns the `Boundable` instances from the array that meets the criterion defined by the given `BoundableSelector`. For example, a class or Lambda expression that implements this interface could find which `Boundable` object in the `ArrayList` has the smallest area, etc.

2.11 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.

2.12 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 – Summer 2023
 * Date: <the date you start goes here>
 * Instructor: Adam Divelbiss
 * Assignment: Program01
 * Purpose: <write the purpose of your program here>
 */
```

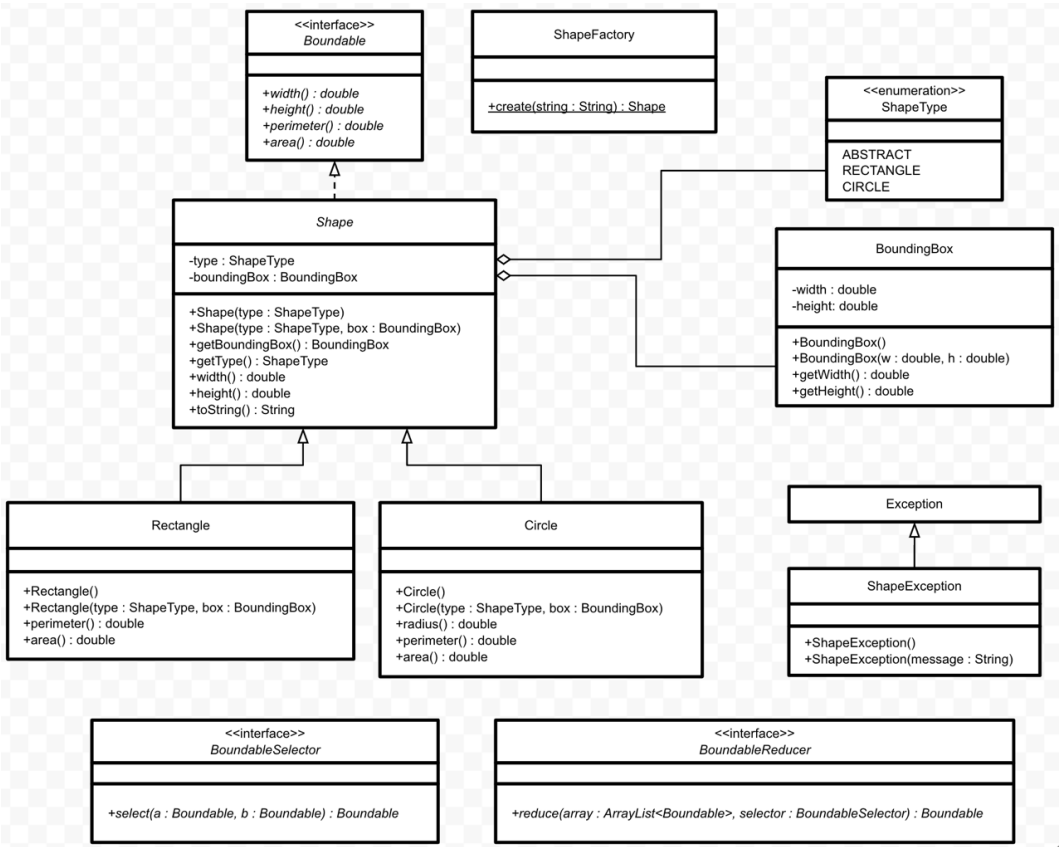
Replace the "<" ">" angle brackets and text inside them with your information.

All class, interface, enumeration, and method headers, 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.

2.13 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.

- Boundable.java
- ShapeType.java
- BoundingBox.java
- Shape.java
- Rectangle.java
- Circle.java
- ShapeException.java
- ShapeFactory.java – Updated with your changes.
- BoundableReducer.java
- BoundableSelector.java
- Main.java - Updated with your changes.

The due date/time is:
Saturday, July 22, 2023, at 11:59PM

4 Sample Output

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

```
Shape Analyzer 1.0

  Invalid width value < 0: -0.8615351439689705
  Invalid width value: BADVALUE
  Invalid height value: BADVALUE
  Invalid width value: BADVALUE
  Invalid height value: BADVALUE
  Invalid width value < 0: -0.3122872453450647
  Invalid width value: BADVALUE
  Invalid height value: BADVALUE
  Invalid width value < 0: -0.7279107498306163
  Invalid width value < 0: -0.36446432086154434
  Invalid height value: BADVALUE

<<< ADDITIONAL INVALID VALUES NOT SHOWN >>>

  Invalid width value: BADVALUE
  Invalid width value: BADVALUE
  Invalid width value: BADVALUE

Number of valid shapes: 9,695

Largest width:
CIRCLE
  width:          98.00
  height:         98.00
  area:        7,542.41
  perimeter:     307.86

Largest height:
CIRCLE
  width:          98.00
  height:         98.00
  area:        7,542.41
  perimeter:     307.86

Largest area:
RECTANGLE
  width:          96.81
  height:         97.35
  area:        9,424.06
  perimeter:     388.31

Largest perimeter:
RECTANGLE
  width:          96.81
  height:         97.35
  area:        9,424.06
  perimeter:     388.31

End report
```

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, interfaces, and enumerations should have a valid, meaningful documentation comment including a purpose or brief description statement of the class.

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.	10
4	Internal Operation: Written to meet all requirements for internal operation as specified as specified or implied by the problem statement.	50
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-Summer 2023), and purpose/description at the top of each file.	5
7	Documentation: All methods, classes, interfaces, and enumerations 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**