Lab Partner 1 Omar Ashurbayov (oashurbayov@hawk.iit.edu)
Lab Partner 2 Newfal Haddi (nhaddi@hawk.iit.edu)
Lab Partner 3 Jesus Chavez Leanos (jchavezleanos@hawk.iit.edu)

CS 116 Spring 2023 Lab #02

Due: **Wednesday, February 1, 2023 at 11:59 PM CST**
Points: **20**

Instructions:

1. Use this document template to report your answers and create separate java files for your classes. Enter all lab partner names at the top of first page.
2. You don't need to finish your lab work during the corresponding lab session.
3. ZIP your Java files and lab report into a single file. Name the file as follows:

```
LastName_FirstName_CS116_Lab2_Report.zip
```

4. Submit the final document to Blackboard Assignments section before the due date. No late submissions will be accepted.
5. ALL lab partners need to submit a report, even if it is the same document.

Objectives:

1. (7 points) Write and test user-defined class.
2. (5 points) Demonstrate the ability write user-defined classes (and read corresponding UML class diagrams) with copy constructors and equals() methods,
3. (8 points) Write a user-defined class that involves conditions.

Problem 1 [7 points]:

Write a user-defined Java class called `BoxVolumeCalculator` that could be used to calculate the volume of a box given length, height, and width.

Your class should:

- NOT have a `main` method,
- Should have four fields (use correct data types and **remember about information hiding**):
  - `length`,
  - `width`,
  - `height`, and
  - `volume`,
- A no-argument (non-parameterized) constructor method that resets all four fields to zero,
- Three mutator / setter methods:
  - `setLength(      )` that will set the `length` field to a new value passed as argument (regardless of whether it is valid or not),

    - `setWidth(      )` that will set the `width` field to a new value passed as argument (regardless of whether it is valid or not),
    - `setHeight(      )` that will set the `height` field to a new value passed as argument (regardless of whether it is valid or not),
    - Use public access modifiers for all four mutator / setter methods,
- Four accessor / getter methods:
    - `getLength()` that will return the value of the `length` field,
    - `getWidth()` that will return the value of the `width` field,
    - `getHeight()` that will return the value of the `height` field,
    - `getVolume()` that will return the value of the `volume` field,
    - Use public access modifiers for all four accessor/ getter methods,
- a `calculateVolume()` method which:
    - will calculate box volume based on current values of `length`, `height`, and `width` fields and update the `volume` field accordingly.

If you implemented your `BoxVolumeCalculator` class correctly and compile/run the following program:

`LabProblemTest` **program:**

```
public class LabProblemTest {

      public static void main(String args[]){
            //  Instantiate  the  BoxVolumeCalculator  class  object
(calling constructor)
            BoxVolumeCalculator        myCalculator        =        new
BoxVolumeCalculator();

      // Show field values before using setter methods to change them
            System.out.println("Initial dimensions:");
            System.out.println("Length: " + myCalculator.getLength() +
"  |  Width:  "  +  myCalculator.getWidth()  +  "  |  Height:  "  +
myCalculator.getHeight());

            // Set box dimensions using mutator / setter methods

            myCalculator.setLength(2.0);
            myCalculator.setWidth(3.0);
            myCalculator.setHeight(4.0);

            // Show field values before using setter methods to change
them
            System.out.println("New dimensions:");
            System.out.println("Length: " + myCalculator.getLength() +
"  |  Width:  "  +  myCalculator.getWidth()  +  "  |  Height:  "  +
myCalculator.getHeight());

            // Re-calculate box volume
            myCalculator.calculateVolume();

            // Show the current box volume
            System.out.println("Calculated      volume:      "      +
myCalculator.getVolume());
```

```
        }
}
```

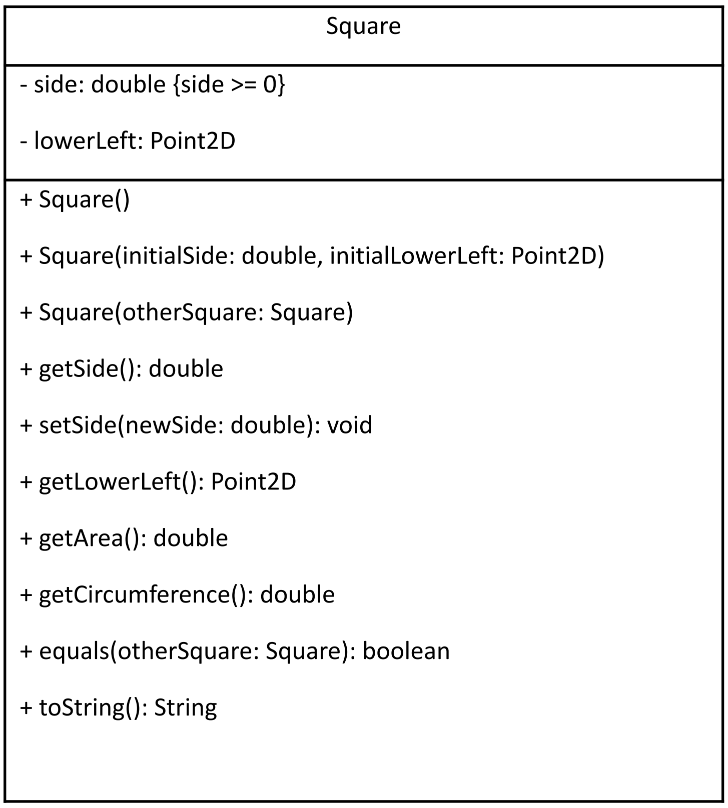
You should see the following output:

```
Initial dimensions:
Length: 0.0 | Width: 0.0 | Height: 0.0
New dimensions:
Length: 2.0 | Width: 3.0 | Height: 4.0
Calculated volume: 24.0
```

In the program above, note how the calculator object is instantiated and how its methods are being used to achieve the goal.

Problem 2 [5 points]:

You are given the following UML class diagram for a class called `Square`:



Square

- side: double {side >= 0}
- lowerLeft: Point2D

+ Square()
+ Square(initialSide: double, initialLowerLeft: Point2D)
+ Square(otherSquare: Square)
+ getSide(): double
+ setSide(newSide: double): void
+ getLowerLeft(): Point2D
+ getArea(): double
+ getCircumference(): double
+ equals(otherSquare: Square): boolean
+ toString(): String

Notes:

- `Point2D` is a class representing a single point in 2D space (see code below),
- `lowerLeft` represents Cartesian coordinates of lower left corner of your square,

- non-default non-parameterized constructor should set the `side` to 0 and `lowerLeft` to (0, 0),
- in non-default parameterized constructor:
  - if lower left corner `Point2D` reference is `null`, assume the coordinates are (0, 0),
  - when `initialSide` is invalid, set it to 0,
- `Square(otherSquare: Square)` is so called copy constructor. It is a special constructor creates (in this case initializes attributes) an object (a new `Square` object in this case) using another object of the same Java class (existing `Square` type object `otherSquare`). Think: creating a physical duplicate (distinct object) of your IIT ID with the same information (by copying data from one to the other). There is a couple things to consider:
  - when `otherSquare` is a `null` reference, fall back on the non-default non-parameterized constructor behavior,
  - note that one of class `Square` attributes/fields is going to be an object. Copying it **does NOT** mean simply copying its reference (shallow copy). It means creating another object of the same type with same attribute values (deep copy),
- `equals(otherSquare: Square)` is an object comparison method. Using the == operator when applied to object references will tell you whether both references are referring to the same object. If you want to check if two distinct objects of the same type are the same data/information-wise, you will need to use equals() method (see how it is used for String objects) and compare each individual attribute/field value (**that includes object comparison if one of the attributes/fields is an object**). If all match, two compared objects are the same. If `otherSquare` is a `null` reference, return `false`,
- `toString()` should display a similar message to:
  - `Square: lower left corner: (2, 3) | side: 2.0 | area: 4.0 | circumference: 8.0`

`Point2D` class

`public class Point2D {`
 	
`private final double x;`
`private final double y;`

```
	public Point2D(){
		this.x = 0.0d;
		this.y = 0.0d;
	}

	public Point2D(double initialX, double initialY){
		this.x = initialX;
		this.y = initialY;
	}

	public Point2D(Point2D otherPoint){
		if (otherPoint != null){
			this.x = otherPoint.getX();
			this.y = otherPoint.getY();
		} else {
			this.x = 0.0d;
			this.y = 0.0d;
		}
	}

	public double getX(){
		return this.x;
	}

	public double getY(){
		return this.y;
	}

	public boolean equals(Point2D otherPoint){
		if (otherPoint != null){
			if    (this.x    ==    otherPoint.getX()    &&    this.y    ==
otherPoint.getY()){
				return true;
			} else {
				return false;
			}
		} else {
			return false;
```

```
                }

        }


        public String toString(){

                return "(" + this.x + ", " + this.y + ")";

        }

}
```

Your task is to implement this class in Java. You can test your class using the following app code:

`SquareApp` class

```
public class SquareApp {

        public static void main(String[] args) {

                Square s1 = new Square();

                Square s2 = new Square(2.0, new Point2D(1.0, 1.0));

                Square s3 = null;

                Square s4 = new Square(s2);

                Square s5 = new Square(3.0, new Point2D(3.0, 2.0));

                Square s6 = s2;

                Square s7 = new Square(s3);


                System.out.println("s1: " + s1);

                System.out.println("s2: " + s2);

                System.out.println("s3: " + s3);

                System.out.println("s4: " + s4);

                System.out.println("s5: " + s5);

                System.out.println("s6: " + s6);

                System.out.println("s7: " + s7);

                System.out.println();


                System.out.println("s1 refers to the same object as s2: " + (s1 ==
s2));

                System.out.println("s2 refers to the same object as s3: " + (s2 ==
s3));

                System.out.println("s4 refers to the same object as s2: " + (s4 ==
s2));

                System.out.println("s6 refers to the same object as s2: " + (s6 ==
s2));

                System.out.println("s7 refers to the same object as s1: " + (s7 ==
s1));
```

```
               System.out.println();


               System.out.println("object referred to by s1 represents the same square as object referred to by s2: " + (s1.equals(s2)));

               System.out.println("object referred to by s2 represents the same square as object referred to by s3: " + (s2.equals(s3)));

               System.out.println("object referred to by s4 represents the same square as object referred to by s2: " + (s4.equals(s2)));

               System.out.println("object referred to by s6 represents the same square as object referred to by s2: " + (s6.equals(s2)));

               System.out.println("object referred to by s7 represents the same square as object referred to by s1: " + (s7.equals(s1)));

       }

}
```

If you coded everything correctly, the output of this app should be as below:

`SquareApp` class output

```
s1: Square: lower left corner (0.0, 0.0) | side : 0.0 | area : 0.0 |
circumference: 0.0
s2: Square: lower left corner (1.0, 1.0) | side : 2.0 | area : 4.0 |
circumference: 8.0
s3: null
s4: Square: lower left corner (1.0, 1.0) | side : 2.0 | area : 4.0 |
circumference: 8.0
s5: Square: lower left corner (3.0, 2.0) | side : 3.0 | area : 9.0 |
circumference: 12.0
s6: Square: lower left corner (1.0, 1.0) | side : 2.0 | area : 4.0 |
circumference: 8.0
s7: Square: lower left corner (0.0, 0.0) | side : 0.0 | area : 0.0 |
circumference: 0.0

s1 refers to the same object as s2: false
s2 refers to the same object as s3: false
s4 refers to the same object as s2: false
s6 refers to the same object as s2: true
s7 refers to the same object as s1: false

object referred to by s1 represents the same square as object referred to by s2:
false
object referred to by s2 represents the same square as object referred to by s3:
false
object referred to by s4 represents the same square as object referred to by s2:
true
object referred to by s6 represents the same square as object referred to by s2:
true
object referred to by s7 represents the same square as object referred to by s1:
true
```

Problem 3 [8 points]:

Write a user-defined class called `MyCalculator` that will **evaluate (calculate)** simple arithmetic expressions of the following form:

A operator B

where:

- ■ *A* and *B* are the two operands of the operation. Each is a **SINGLE DIGIT positive integer** (0-9), and
- ■ *operator* is one of the following arithmetic operators: + (addition), - (subtraction), * (multiplication), / (division), ^ (exponentiation: “A to the power of B”).

Here’s a couple sample expressions:

*6+3, 1-9, 0*8, 3/3, 7^4*

Your `MyCalculator` class should:

- ■ have a `main` method that will:
  - ■ obtain the arithmetic expression from the user, **[0.5 pts]**
  - ■ use one of the `String` class methods to remove/replace all spaces from the user input (for example: “`3   +  4`” should become “`3+4`”), **[0.5 pts]**
  - ■ call `evaluateExpression()` method and pass the expression to it. This method will return the result of expression evaluation (assuming the expression in question is valid and can be evaluated. If not, the following message should be displayed on screen: "`ERROR: Impossible to evaluate this expression.`"). **[1 pt]**

    HINTS:
    - ■ refresh your understanding of wrapper classes and autoboxing and unboxing concepts,
    - ■ read `Double` wrapper class documentation,
    - ■ use `Double.NaN` constant,
    - ■ consider using `isNaN()` method (of `Double` class).

- ■ have a `evaluateExpression()` method that will:
  - ■ accept a single parameter / argument: expression, **[0.5 pts]**
  - ■ perform some validation / sanity checks (things to consider: Strings are objects that are accessed via reference variables; what is a special reference variable value that could lead to issues? How many characters should be included in the expression after all spaces are removed?). NOTE: you can ignore checking if A and B are actually numbers **for now**, but please see what happens when you enter for example “`[*y`” as your expression. Think how would you address this issue. **[1 pts]**
  - ■ use a Java `switch` structure (this is essential: **don’t** use if-elseif-else structures instead) to evaluate the expression result depending on the operator, **[3 pts]**
    - ◆ If the operator is not on the list of legal operators, your program should display the following message on screen: "`ERROR: Unknown operator.`"
  - ■ return the result of the operation: **[0.5 pts]**
    - ◆ numerical value if the expression can be evaluated,

- `Double.NaN` value if the expression cannot be evaluated.

Here's a couple program output / interactions samples:

```
Format: A operator B, where A and B are positive SINGLE
DIGIT integers
Allowed operators: + - * / ^
Sample expressions: 6+3, 1-9, 0*8, 3/3, 7^4
***********************************************
Enter your arithmetic expression: 3   * 4
User expression: 3   * 4
User expression after removing spaces: 3*4
3*4=12.0
Format: A operator B, where A and B are positive SINGLE
DIGIT integers
Allowed operators: + - * / ^
Sample expressions: 6+3, 1-9, 0*8, 3/3, 7^4
***********************************************
Enter your arithmetic expression: 2^ 10
User expression: 2^ 10
User expression after removing spaces: 2^10
ERROR: This expression is either too long or too short.
ERROR: Impossible to evaluate this expression.

Format: A operator B, where A and B are positive SINGLE
DIGIT integers
Allowed operators: + - * / ^
Sample expressions: 6+3, 1-9, 0*8, 3/3, 7^4
***********************************************
Enter your arithmetic expression: 4-9
User expression: 4-9
User expression after removing spaces: 4-9
4-9=-5.0

Format: A operator B, where A and B are positive SINGLE
DIGIT integers
Allowed operators: + - * / ^
Sample expressions: 6+3, 1-9, 0*8, 3/3, 7^4
***********************************************
Enter your arithmetic expression: 3v5
User expression: 3v5
User expression after removing spaces: 3v5
ERROR: Unknown operator.
ERROR: Impossible to evaluate this expression.
```