Unit 2: Using Objects

Topic 2 Lab 1: Overloaded Constructors & Getters/Setters

Name:	

1. Create a new IntelliJ project and name it LASTNAMEU2T1Lab1 (or whatever you want).

2. Create a new `Rectangle` class and copy/paste **this code**.

3. Create a `RectangleRunner` class and add a main method.

How many *constructors* does the `Rectangle` class have? How can you *tell* just by looking at a method that it's a constructor?	*Confirm*

4. In the `RectangleRunner` class, create three `Rectangle` objects to represent three different plots of farmland, described below. Name the variables `plot1, plot2, plot3`

- `plot1`: a plot with a length of 150 and width of 200 -- use the constructor with two parameters
- `plot2`: a square plot with side length 125 -- use the constructor with one parameter
- `plot3`: a plot that has "default" values -- use the constructor with *no* parameters

5. Then, using appropriate `Rectangle` methods, write code in your runner class to print out the length, width, and area of each rectangle, like the following; do **not** modify the Rectangle class. You will need to use the "getter" methods (note that "getter" methods *return* values). **Using "getter" methods is generally how a client directly accesses (reads) the values of an object's instance variables (i.e. its stored data).**

```
Plot 1 length: 150
Plot 1 width: 200
Plot 1 area: 30000

Plot 2 length: 125
Plot 2 width: 125
Plot 2 area: 15625

Plot 3 length: 100
Plot 3 width: 50
Plot 3 area: 5000
```

After you type it up yourself, you can compare your code with this sample if you want.

Why does `plot2` have length and width *both* set to 125?	

Why does `plot3` have length and width set to 100 and 50?	*check*

6. *Under* the print statements you typed above, write code to **update** all three plot *widths* to be 75. For `plot2` (the square plot), *also* update its *length* to 75 to keep it a square. You will need to use the "setter" methods (note that "setter" methods are **void** methods).

confirm

7. Copy all the print statements you typed for step 5 and paste them below the code you just typed, so that you can see the *updated* values printed! **Using "setter" methods is generally how a client directly updates the values of an object's instance variables (i.e. its stored data).**

confirm

8. Once you see how getters and setters are used, you can remove all the print statements (leave the code from step 6).

9. Now, assume you need to purchase grass seed to plant grass on all three plots. Write some code to calculate the total area of all three plots, and print it out:

```
These three plots requires 24375 square feet of seed
```

confirm

Copy/paste your entire `RectangleRunner` class code below:

sample

Lab continues on next page

10. Two of these code segments *will* successfully create objects from the Rectangle class but one *will not*. Which one will not, and why? (*hint*: inspect the constructors… it has to do with data types!)

```
 I.   int len = 65;
      int wid = len + 10;
      Rectangle rect = new Rectangle(len, wid);

II.   Rectangle rect = new Rectangle(10.0, 30.0);

III.  Rectangle rect = new Rectangle(0);
```

My answer:

11. TEST! Comment out your code in RectangleRunner and test your answer above by copying/pasting/running the following code segment:

```
int len = 65;
int wid = len + 10;
Rectangle rect1 = new Rectangle(len, wid);
Rectangle rect2 = new Rectangle(10.0, 30.0);
Rectangle rect3 = new Rectangle(0);
```

After running the code, was your answer above correct?

What does the error indicate? Is it a compiler/syntax (red squiggly) error or a runtime error?

Confirm

Lab continues on next page

Boxes!

Below is a `Box` class. Note that the method `allSidesLongerThan(int side)` utilizes **Boolean** logic; in Java, the `||` operator is the **OR** operator (we will talk more about Booleans in Unit 3).

```
public class Box
{
  private double length;
  private double width;
  private double height;

  public Box(double length, double width, double height)
  {
    this.length = length;
    this.width = width;
    this.height = height;
  }

  public Box(double side)
  {
    length = side;
    width = side;
    height = side;
  }

  public double volume()
  {
    return length * width * height;
  }

  public boolean anySideLongerThan(int side)
  {
    if (length > side || width > side || height > side)
    {
      return true;
    }
    else
    {
      return false;
    }
  }

  public void printDimensions()
  {
    System.out.println("L = " + length + ", W = " + width + ", H = " +
height);
  }
}
```

13. How many *constructors* does the `Box` class have?

Confirm

14. Predict, using just your brains!
Determine which of the following statements *will* create a new `Box` object, and which will *not,* causing compiler errors (*hint:* look carefully at the `Box` constructor's parameters)

A) `Box box = new Box(5.0, 4.5, 7.2);`
B) `Box box = new Box(5, 4, 7);`
C) `Box cube = new Box(15.0);`
D) `Box cube = new Box(15);`
E) `Box box = new Box(2.5, 6.7);`
F) `Box box = new Box();`

PREDICTIONS
Will it create a Box object, or will an error occur? If an error will occur, why?

A)
B)
C)
D)
E)
F)

TEST!

15. Create a new `Box` class and copy/paste the `Box` class code from above.

16. Create a new `BoxRunner` class with a main method, and copy/paste the following into it; fix the indentations to look nice 😎

```
Box box = new Box(5.0, 4.5, 7.2);
Box box = new Box(5, 4, 7);
Box cube = new Box(15.0);
Box cube = new Box(15);
Box box = new Box(2.5, 6.7);
Box box = new Box();
```

17. ***You can't declare more than one variable with the same name*****,** so rename all the `box` variables as `box1`, `box2`, `box3`, `box4` and the cube variables as `cube1`, `cube2`.

What should my code look like?

18. Run the code to see which lines produce errors! Read the error messages; do they make sense to you? *Comment out those lines of code* and try running it again -- once you have commented out all broken lines, the program should run without error. **Confirm your predictions by comparing the incorrect (commented out) lines to your predictions!**

19. Were your predictions correct?
If not, what mistake(s) did you make?

Confirm

20. Assume a `Box` object `newBox` has been properly declared and initialized as part of a client class as follows:

```
Box newBox = new Box(6, 10, 8);
```

Which of the following statements, labeled A - G, are *valid* and

Categorize each statement by putting an X in the appropriate column:

	Valid (will	Will lead to

will compile/execute without error, and which *will lead to errors*?

```
double larger = newBox.anySideLongerThan(10); // A

boolean big = newBox.anySideLongerThan(7.5); // B

boolean x = newBox.anySideLongerThan(5); // C

String dim = newBox.printDimensions(); // D

System.out.println(newBox.printDimensions()); // E

newBox.printDimensions(); // F

double volume = newBox.volume(21.8); // G
```

	work):	an error:
A		
B		
C		
D		
E		
F		
G		

TEST!

21. In your `BoxRunner` class, copy and paste this: `Box newBox = new Box(6, 10, 8);`

22. Then, copy and paste the code above. Run it to see what happens! If any of your predictions was incorrect, be sure to read the explanations:

Explanations

23. The developer of the `Box` class wants to add the following method, `printVolume()`, to the `Box` class that prints the volume. The developer wants to use the already existing `volume()` method in the new `printVolume()` method.

```
public void printVolume() {
  System.out.println("Volume = " + ________);
}
```

Which one of the following could the developer type in the space?

A) `volume`
B) `volume()`
C) `volume(length, width, height)`
D) `Box.volume()`
E) `None of the above`

ANSWER:

24. You guessed it! TEST by copying/pasting the `printVolume()` method to your `Box` class and completing it as you believe it should be written, based on your answer to 23. Then, add a line of code in your BoxRunner class to test the new method on your `newBox` object.

Does it work? If so, your answer to 23 above was correct! If not, figure out your mistake!

Confirm answer and test code

Time to write your first class! 😎

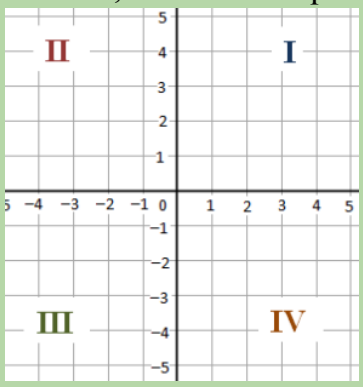
Have fun with this and challenge yourself! This is the first time you are writing a complete class, and it may seem crazy at first, but don't fret -- we will be talking in much more detail in the coming weeks about how to design classes. *But see what you can come up with here!*

Review how the `Rectangle` class was set up!

Now create and write a complete `Point` class that has:

- two `int` instance variables named `x` and `y` (be sure to make them **`private`**!)
- a constructor that accepts and sets both instance variables
  - If you also name your parameters x and y, be sure to use the `this` keyword!
- a constructor that accepts one value and sets *both* x and y to the *same* value
- a constructor that has no parameters and sets x and y to 0 (the origin)
- a "getter" method for x, name it `getX` (it should return an int)
- a "getter" method for y, name it `getY` (it should return an int)
- a "setter" method for x, name it `setX` (it should be a void method and have an int parameter)
  - name the parameter `newX` (as is typical)
- a "setter" method for y, name it `setY` (it should be a void method and have an int parameter)
  - name the parameter `newY` (as is typical)
- a method named `coordinate` that has no parameters and returns the x and y value as a String coordinate in this format: "(5, 8)"
- a method named `quadrant` that has no parameters and returns as a String the quadrant the point is in, either "I", "II", "III", "IV", "origin", or "on an axis"
  - **TIP!** For this, you might want to use `AND` logic, which we haven't learned yet but in Java the "and operator" is this: `&&`
    - For example: `if (x < 0 && y > 0)`
    - Another example: `if (x == 0 && y < 0)`
  - **TIP 2:** using an if statement with multiple else if branches and an else could also be helpful!

As a reminder, here are the quadrants:



Write the complete `Point` class!

TESTING: When you are ready to test your class, copy/paste the following code into a new `PointTester` class:

```
public class PointTester {
    public static void main(String[] args) {
        System.out.println("-- TESTING CONSTRUCTORS AND GETTER METHODS --");
        Point p1 = new Point(10, 5);
        System.out.println(p1.getX());
        System.out.println(p1.getY());

        Point p2 = new Point(12);
        System.out.println(p2.getX());
        System.out.println(p2.getY());

        Point p3 = new Point();
        System.out.println(p3.getX());
        System.out.println(p3.getY());

        System.out.println("-- TESTING SETTER, COORDINATE, QUADRANT METHODS
--");
        System.out.println(p1.coordinate());
        System.out.println(p1.quadrant());
        p1.setX(-8);
        System.out.println(p1.coordinate());
        System.out.println(p1.quadrant());
        p1.setY(-14);
        System.out.println(p1.coordinate());
        System.out.println(p1.quadrant());
        p1.setX(7);
        System.out.println(p1.coordinate());
        System.out.println(p1.quadrant());
        p1.setX(0);
        System.out.println(p1.coordinate());
        System.out.println(p1.quadrant());
        p1.setY(0);
        System.out.println(p1.coordinate());
        System.out.println(p1.quadrant());
        p1.setX(13);
        System.out.println(p1.coordinate());
        System.out.println(p1.quadrant());
    }
}
```

RUN IT AND SEE IF YOUR CODE PRODUCES THE FOLLOWING OUTPUT EXACTLY:

```
-- TESTING CONSTRUCTORS AND GETTER METHODS --
10
5
12
12
0
0
```

```
-- TESTING SETTER, COORDINATE, QUADRANT METHODS --
(10, 5)
I
(-8, 5)
II
(-8, -14)
III
(7, -14)
IV
(0, -14)
on an axis
(0, 0)
origin
(13, 0)
on an axis
```

Copy/paste your `Point` class below:

sample solution

Done!

Submit in Google Classroom:

Turn in

Confirm code (back)

Your code should look something like this.
Note the use of the "getLength" and "getWidth" getter methods, which *return* **int** values

```
public class RectangleRunner {
    public static void main(String[] args) {
        Rectangle plot1 = new Rectangle(150, 200);
        Rectangle plot2 = new Rectangle(125);
        Rectangle plot3 = new Rectangle();

        System.out.println("Plot 1 length: " + plot1.getLength());
        System.out.println("Plot 1 width: " + plot1.getWidth());
        System.out.println("Plot 1 area: " + plot1.calculateArea());
        System.out.println();
        System.out.println("Plot 2 length: " + plot2.getLength());
        System.out.println("Plot 2 width: " + plot2.getWidth());
        System.out.println("Plot 2 area: " + plot2.calculateArea());
        System.out.println();
        System.out.println("Plot 3 length: " + plot3.getLength());
        System.out.println("Plot 3 width: " + plot3.getWidth());
        System.out.println("Plot 3 area: " + plot3.calculateArea());
    }
}
```

Confirm (back)

You could do it like this:

```
int plot1Area = plot1.calculateArea();
int plot2Area = plot2.calculateArea();
int plot3Area = plot3.calculateArea();
int seedRequired = plot1Area + plot2Area + plot3Area;
System.out.println("These three plots requires " + seedRequired + " square feet of seed");
```

Or like this:

```
int seedRequired = plot1.calculateArea() + plot2.calculateArea() + plot3.calculateArea();
System.out.println("These three plots requires " + seedRequired + " square feet of seed");
```

Confirm (back)

How many *constructors* does the `Rectangle` class have? How can you *tell* just by looking at a method that it's a constructor?	**3** constructors (so this class has "**overloaded**" constructors because there is more than one!) You can tell a method is a constructor because it has the exact same name as the class (in this case, Rectangle):

```java
// constructor to create a Rectangle object with a particular width and length
public Rectangle(int rectLength, int rectWidth) {
    length = rectLength;
    width = rectWidth;
}

// constructor to create a Rectangle object with equal width and length (a square)
public Rectangle(int side) {
    length = side;
    width = side;
}

// "no-argument" constructor which creates a Rectangle object with default values
public Rectangle() {
    length = 100;
    width = 50;
}
```

Check (back)

Why does plot2 have length and width *both* set to 125?
 	
Why does plot3 have length and width set to 100 and 50?
	Creating plot 2 looks like this:
	`Rectangle plot2 = new Rectangle(125);`
	using one parameter, which corresponds to the one-parameter constructor in the Rectangle class:
	`public Rectangle(int side) { length = side; width = side; }`
	In this constructor, both the length and width instance variables get set to the value passed in, which was 125
	--- 	
	Creating plot 3 looks like this:
	`Rectangle plot3 = new Rectangle();`
	using no parameters, which corresponds to the no-parameter constructor in the Rectangle class:
	`public Rectangle() { length = 100; width = 50; }`
	In this constructor, the length is set to a "default" value of 100 and the width is set to a "default" value of 50

Confirm (back)

Use the "setter" methods -- note that those methods are "void" methods (so we just call them):

```
System.out.println("Plot 3 width: " + plot3.getWidth());
System.out.println("Plot 3 area: " + plot3.calculateArea());

plot1.setWidth(75);
plot2.setWidth(75);
plot3.setWidth(75);
plot2.setLength(75); // keep plot2 a square!
```

Confirm (back)

Your code (yes, it's long!) should look something like this:

```
Rectangle plot1 = new Rectangle(150, 200);
Rectangle plot2 = new Rectangle(125);
Rectangle plot3 = new Rectangle();

System.out.println("Plot 1 length: " + plot1.getLength());
System.out.println("Plot 1 width: " + plot1.getWidth());
System.out.println("Plot 1 area: " + plot1.calculateArea());
System.out.println();
System.out.println("Plot 2 length: " + plot2.getLength());
System.out.println("Plot 2 width: " + plot2.getWidth());
System.out.println("Plot 2 area: " + plot2.calculateArea());
System.out.println();
System.out.println("Plot 3 length: " + plot3.getLength());
System.out.println("Plot 3 width: " + plot3.getWidth());
System.out.println("Plot 3 area: " + plot3.calculateArea());

plot1.setWidth(75);
plot2.setWidth(75);
plot3.setWidth(75);
plot2.setLength(75); // keep plot2 a square!

System.out.println("Plot 1 length: " + plot1.getLength());
System.out.println("Plot 1 width: " + plot1.getWidth());
System.out.println("Plot 1 area: " + plot1.calculateArea());
System.out.println();
System.out.println("Plot 2 length: " + plot2.getLength());
System.out.println("Plot 2 width: " + plot2.getWidth());
System.out.println("Plot 2 area: " + plot2.calculateArea());
System.out.println();
System.out.println("Plot 3 length: " + plot3.getLength());
System.out.println("Plot 3 width: " + plot3.getWidth());
System.out.println("Plot 3 area: " + plot3.calculateArea());
```

And when you run it, you should see the original values first printed, **THEN** the updated values!

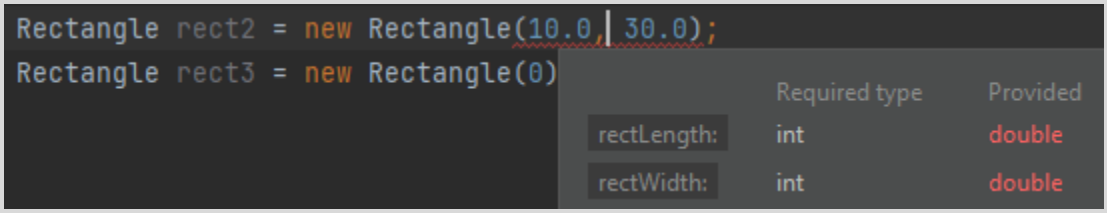
```
Plot 1 length: 150
Plot 1 width: 75
Plot 1 area: 11250

Plot 2 length: 75
Plot 2 width: 75
Plot 2 area: 5625

Plot 3 length: 100
Plot 3 width: 75
Plot 3 area: 7500
```

Sample (back)

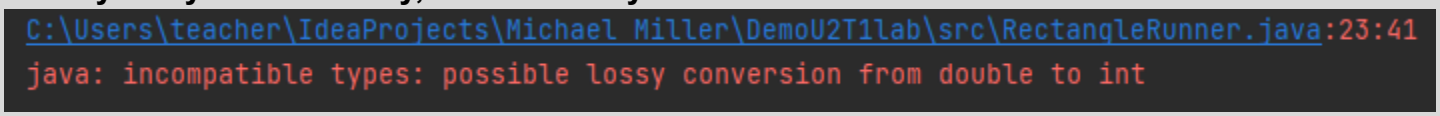
```
public class RectangleRunner {
    public static void main(String[] args) {
        Rectangle plot1 = new Rectangle(150, 200);
        Rectangle plot2 = new Rectangle(125);
        Rectangle plot3 = new Rectangle();

        // update instance variables
        plot1.setWidth(75);
        plot2.setWidth(75);
        plot3.setWidth(75);
        plot2.setLength(75);

        // calculate and print total area
        int plot1Area = plot1.calculateArea();
        int plot2Area = plot2.calculateArea();
        int plot3Area = plot3.calculateArea();
        int seedRequired = plot1Area + plot2Area + plot3Area;
        System.out.println("These three plots requires " + seedRequired + " square feet of seed");
    }
}
```

Answer (back)

`II` will NOT compile, and there is a **compile-time/syntax** error that shows you:



When you try to run it away, we see Java yell:



That is because the constructor `public Rectangle(int rectLength, int rectWidth)` is expecting two ***integers***, and the arguments being passed in are both *doubles*, so the compiler gives an error.

```
 I.   int len = 65;
      int wid = len + 10;
      Rectangle rect = new Rectangle(len, wid);

II.   Rectangle rect = new Rectangle(10.0, 30.0); // will NOT compile

III.  Rectangle rect = new Rectangle(0);
```

`I` **will** work because `len` = 65 (an integer) and `wid` = 75 (an integer), so the types match this constructor: `public Rectangle(int rectLength, int rectWidth)`

`III` **will** also work because `new Rectangle(0)` is a valid use of the single-parameter “square” constructor; even though passing 0 will give you a Rectangle with length and width being 0, it is still valid to do this.

Confirm (back)

Two constructors (so this class has "**overloaded**" constructors because there is more than one); *reminder that a constructor has the same name as the **class** itself:*

`public Box` matches `public class Box`

One constructor has three parameters, and one has one parameter:

```
public class Box
{
  private double length;
  private double width;
  private double height;

  public Box(double l, double w, double h)
  {
    length = l;
    width = w;
    height = h;
  }

  public Box(double s)
  {
    length = s;
    width = s;
    height = s;
  }

  public double volume()
  {
    return length * width * height;
```

Check (back)

Note that the variables are all renamed to avoid duplicate variable names (which isn't allowed in Java!):

BoxRunner.java

```java
public class BoxRunner
{
    public static void main(String[] args)
    {
        Box box1 = new Box(5.0, 4.5, 7.2);
        Box box2 = new Box(5, 4, 7);
        Box cube1 = new Box(15.0);
        Box cube2 = new Box(15);
        Box box3 = new Box(2.5, 6.7);
        Box box4 = new Box();
    }
}
```

Confirm (back)

Determine if each of the following statements *will* create a new `Box` object *without* causing a compiler error:

A) `Box box = new Box(5.0, 4.5, 7.2);`

B) `Box box = new Box(5, 4, 7);`

C) `Box cube = new Box(15.0);`

D) `Box cube = new Box(15);`

E) `Box box = new Box(2.5, 6.7);`

F) `Box box = new Box();`

Will it work?
If an error will occur, why?

A) **Yes**, it correctly uses the `public Box(double l, double w, double h)` constructor

B) **Yes,** even though the constructor asks for `doubles`, Java auto converts `int` to `double` and so it uses the `public Box(double l, double w, double h)` constructor

C) **Yes**, it correctly uses the `public Box(double s)` constructor

D) **Yes**, it uses the `public Box(double s)` constructor in the same was as explained in B

E) **No**, an error will occur because there is no constructor with two `double` parameters.

F) **No**, an error will occur because there is no constructor with no parameters (unlike the Rectangle class from earlier, which *did* have a no-parameter constructor)

Explanations (back)

A. `double larger = newBox.anySideLongerThan(10);` **WILL LEAD TO ERROR!**

The `anySideLongerThan(int)` method returns a **boolean** value, and this statement is attempting to store the return value as a `double`. This will result in a compiler error.

B. `boolean big = newBox.anySideLongerThan(7.5);` **WILL LEAD TO ERROR!**

The `anySideLongerThan(int)` method accepts an `int` value, and this statement is attempting to pass a `double` value in the parameter. Although Java is able to convert an `int` parameter to a `double` (as you saw in the previous problem), it **cannot** convert from a `double` to an `int`. This will result in a compiler error.

C. `boolean x = newBox.anySideLongerThan(5);` **VALID (WILL WORK)!**

The `anySideLongerThan(int)` method accepts an `int` value, and an `int` value is being passed, and the method returns a `boolean` value, and `x` is properly declared as `boolean`.

D. `String dim = newBox.printDimensions();` **WILL LEAD TO ERROR!**

The `printDimensions()` method is a `void` method (no return value), and this statement attempts to store a return value in the `String` variable `dim`, which you cannot do if the method returns no value! This will result in a compiler error.

E. `System.out.println(newBox.printDimensions());` **WILL LEAD TO ERROR!**

The `printDimensions()` method is a `void` method (no return value), and this statement seems to assume the method returns a `String` return value, which is not true; therefore, trying to print the result of the method is not appropriate. This will result in a compiler error.

F. `newBox.printDimensions();` **VALID (WILL WORK)!**

The `printDimensions()` method is a `void` method (no return value), and `void` methods are intended to be invoked by simply calling the method, as is done here.

G. `double volume = newBox.volume(21.8);` **WILL LEAD TO ERROR!**

The `volume()` method's signature does not include a parameter, and this statement is inappropriately providing a parameter. This will result in a compiler error. Note that if the `21.8` was removed, the statement *would* then be valid: `double volume = newBox.volume();`

Answer (back)

The developer of the `Box` class wants to add the following method, `printVolume()`, to the `Box` class that prints the volume for the client. The developer wants to use the `double()` method in the new `printVolume()` method.

```
public void printVolume()
{
  System.out.println("Volume = " + _________);
}
```

Which *one* of the following should the developer type in the space?

A) `volume`
B) **`volume()`**
C) `volume(length, width, height)`
D) `Box.volume()`
E) `None of the above`

The answer is B

`volume()` is how the developer would call the class' `volume` method as part of a print statement.

See below for the full statement.

Test code statement in your BoxRunner class:

```
newBox.printVolume(); // note that printVolume is a VOID method!
```

Sample solution (back)

```java
public class Point {

    // instance variables
    private int x;
    private int y;

    // constructor that takes x and y
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    // constructor that takes one value and sets x and y to that value
    public Point(int num) {
        x = num;
        y = num;
    }

    // constructor that has no parameters and sets x and y to 0
    public Point() {
        x = 0;
        y = 0;
    }
```

```java

    // "getter" method for x instance variable
    public int getX() {
        return x;
    }

    // "getter" method for y instance variable
    public int getY() {
        return y;
    }

    // "setter" method for x instance variable
    public void setX(int newX) {
        x = newX;
    }

```

// continued on the next page

```
    // "setter" method for y instance variable
    public void setY(int newY) {
        y = newY;
    }

    // method that returns coordinate String
    public String coordinate() {
        return "(" + x + ", " + y + ")";
    }

```

```
    // method that returns quadrant of point
    public String quadrant() {
        String quadrant;
        if (x > 0 && y > 0) {
            quadrant = "I";
        } else if (x < 0 && y > 0) {
            quadrant = "II";
        } else if (x < 0 && y < 0) {
            quadrant = "III";
        } else if (x > 0 && y < 0) {
            quadrant = "IV";
        } else if (x == 0 && y == 0) {
            quadrant = "origin";
        } else {
            quadrant = "on an axis";
        }
        return quadrant;
    }
}
```