Unit 2: Using Objects

Topic 8: Using the Math Class

Name:	

Create a new IntelliJ project for today, LASTNAMEU2T9Lab, or something like that.

Circles!

Here is an incomplete `Circle` class:

```
public class Circle
{
  private double radius;

  public Circle(double radius)
  {
    this.radius = radius;
  }

  /* Returns the area of the circle with given radius
     Area of a circle = pi * r ^ 2
     Use Math.PI for pi and use the pow method
   */
  public double area()
  { /* implement this method! */ }


  /* Returns the circumference of the circle with given radius
     Circumference of a circle = pi * 2 * r
     Use Math.PI for pi
   */
  public double circumference()
  { /* implement this method! */ }


  /* Setter method to update the radius of the circle to newRadius
   */
  public void setRadius(double newRadius)
  { /* implement this method! */ }


  /* Returns a String that includes info about the circle,
     including its radius, area, and circumference on separate lines;
     see example output below (Reminder: the new line escape
     sequence \n allows you to include new lines in Strings!)
     Note that this method should return a string -- it should NOT
     do any printing. This method should call your other methods!
   */
  public String getInfo()
  { /* implement this method! */ }
```

Hint for getInfo

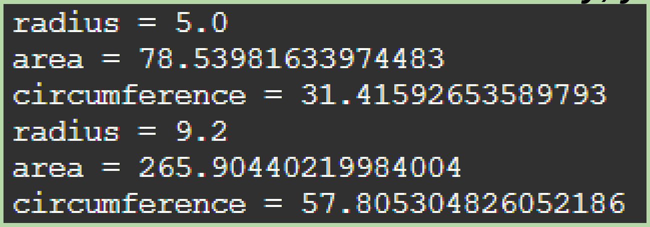
```
}
```

1. Complete the `Circle` class above by completing the three missing methods in IntelliJ.!

2. **TEST** your methods by creating a client/runner class and writing test code to do the following:

- Create a new `Circle` object with a radius of `5.0`
- Print out the string returned by the `getInfo()` method
- *Set* the radius of the circle to a *new* value of `9.2`
- Print out the string returned by the `getInfo()` method again, noticing that all the values reflect the updated radius value

If you wrote your methods and test cases correctly, you should see:



```
radius = 5.0
area = 78.53981633974483
circumference = 31.41592653589793
radius = 9.2
area = 265.90440219984004
circumference = 57.805304826052186
```

Let me check my test code

3. Write *a second* test case of your own.

4. Once you have everything running correctly, copy/paste your code below:

The test code you wrote in steps 2 & 3 above:

Your completed code for the `Circle` class:

Compare

Sam wrote the following code to calculate 6.5 raised to the third power and print the result:

```
double x = 6.5;
double thirdPower = (x ^ 3);
System.out.println(thirdPower);
```

Will this code calculate and print what Sam was hoping? If not, help Sammy **fix** his code so that it does!	*Confirm*

Lab continues on next page

Right Triangles!

1. Create a new `RightTriangle` class.

   a. The class should have two instance variables, `base` and `height`, both of type `double`.

   b. Add a constructor that takes in both base and height as parameters and initializes the instance variables to those values.

   c. Add a `hypotenuse` method that has no parameters and returns a `double`. This method should return the length of the hypotenuse using the Pythagorean Theorem: $a^2 + b^2 = c^2$, where a is `base` and b is `height`. Use $\square = \sqrt{\square^2 + \square^2}$ to find c, the hypotenuse, which gets returned by the method.

Let me confirm the instance variables and constructor

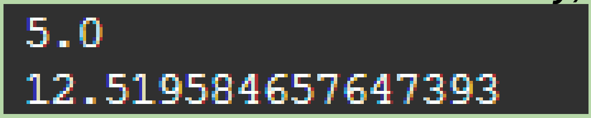
2. **TEST** your class and hypotenuse method by writing test code to do the following:

   ○ Create *two* new `RightTriangle` objects, one with a base = 3 and height = 4, and the other with a base = 6.5 and height = 10.7

   ○ Store the length of each triangle's hypotenuse in its own variable (use the `hypotenuse` method).

   ○ Print each variable to confirm the hypotenuse lengths match the expected output below.

If you wrote your methods and test cases correctly, you should see:



```
5.0
12.519584657647393
```

Let me check my test code

3. Write *a second* test case of your own.

4. Once you have everything running correctly, copy/paste your code below:

The test code you wrote in steps 2 & 3 above:

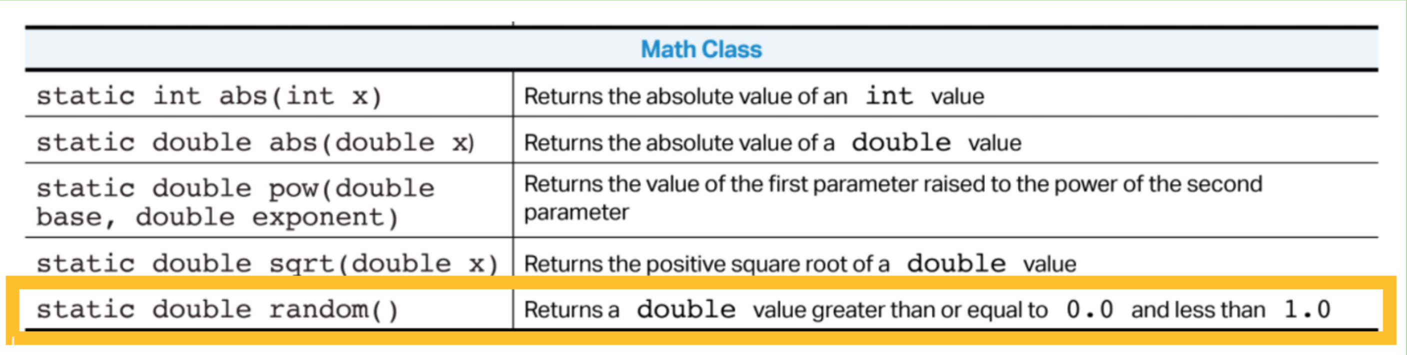
Your completed code for the `RightTriangle` class, including the `hypotenuse` method:

Compare

Exploration continues on next page

RaNdOM nUmBErS

From the AP Exam Java Reference Sheet:



Math Class	
`static int abs(int x)`	Returns the absolute value of an `int` value
`static double abs(double x)`	Returns the absolute value of a `double` value
`static double pow(double base, double exponent)`	Returns the value of the first parameter raised to the power of the second parameter
`static double sqrt(double x)`	Returns the positive square root of a `double` value
`static double random()`	Returns a `double` value greater than or equal to `0.0` and less than `1.0`

`Math.random()` takes no parameters and returns a `double` value *greater than or equal* to 0.0 and *strictly less than* 1.0. In other words, exactly 0.0 *is* possible, but exactly 1.0 is *not* -- although 0.9999… is

Free Style! Write some code to use the `random()` method a few times and print the results (you should see that it produces a different random decimal each time it is called).

Copy/paste the code you wrote:

Can you figure out how to use the `random()` method to generate a random number between 1.0 and 10.0 (*excluding* exactly 10.0)?

See if you can work this out ***before*** *reading below:*

Below are examples of how to generate a random `double` within a certain range:

```
// RANGES OF DOUBLES
// Ex 1: generate a random double between 0.0 and 1.0 (excluding exactly 1.0)
double randomNum = Math.random();

// Ex 2: generate a random double between 0.0 and 10.0 (excluding exactly 10.0)
double randomNum = Math.random() * 10;

// Ex 3: generate a random double between 0.0 and 20.0 (excluding exactly 20.0)
double randomNum = Math.random() * 20;

// Ex 4: generate a random double between 5.0 and 20.0 (excluding exactly 20.0)
```

```
double randomNum = Math.random() * 15 + 5;

// Ex 5: generate a random double between 8.0 and 35.0 (excluding exactly 35.0)
double randomNum = Math.random() * 27 + 8;

// Ex 6: generate a random double between 15.0 and 60.0 (excluding exactly 60.0)
double randomNum = Math.random() * 45 + 15;
```

Revisit your answer to the previous question; after seeing the 5 examples above, modify your answer (if needed, otherwise leave blank!):

Using the examples above, figure out how to write an expression to generate a random double between 25.0 and 100.0, *exclusive* of 100.0.

Confirm

```
// RANGES OF INTS
note the use of (int) casting AFTER the multiplication takes place!

// Example 6: generate a random int between 0 and 9, inclusive (including 9)
int randomNum = (int) (Math.random() * 10);

// Example 7: generate a random int between 1 and 10, inclusive (including 10)
int randomNum = (int) (Math.random() * 10) + 1;

// Example 8: generate a random int between 5 and 10, inclusive (including 10)
int randomNum = (int) (Math.random() * 6) + 5;

// Example 9: generate a random int between 8 and 20, inclusive (including 20)
int randomNum = (int) (Math.random() * 13) + 8;

// Example 10: generate a random int between 75 and 99, inclusive (including 99)
int randomNum = (int) (Math.random() * 25) + 75;
```

Study the examples above; focus on 6 - 10 which are used to produce a random *integer* (the most common use!)

What *relationship* do you notice between the desired random range (e.g. “between 8 and 20”) and the numbers used to produce the random number?

When you think you have the relationship figured out, complete the line of code below to generate and store a random `integer` from `50` to `85`, *inclusive*, in `randomNum`:

```
int randomNum =
```

TEST! We haven't formally discussed **iteration** (loops) yet, but below is a `for` loop in Java that you can use to help test to see if your expression produces a random number in the desired random range. Copy/paste the code below in your IDE, replacing your code above in the space:

```
for (int i = 0; i < 100; i++) {
  int randomNum = /* put your code here to generate a random number between 50 and 85 */
  System.out.println(randomNum);
}
```

This will loop **100** times, printing a random integer each time; it's not a perfect way to test randomness, because you can't ever guarantee certain numbers will get selected, but it's a decent way to get a feel for the types of numbers your code produces.

You should be looking to make sure:

- **50** is the ***lowest*** possible number that ***ever*** shows up
- **85** is the ***highest*** possible number that ***ever*** shows up
- All other values fall between these

With 100 loops, you have a pretty good chance of seeing both "border" numbers (50 and 85); if not, just run it again until you do! Feel free to change 100 to another number, or change up how the printing is formatted.

→ After testing your code with the loop, ***confirm the relationship & code*** ←

After checking the answer above, choose which "version" of the formula you like best (A and B or min and max), and type it below:

This is a super important formula that you need to memorize!

Now use your formula to write the expressions below:

Generate and store a random `integer` from `0` to `6`, *inclusive*, in `randomNum`:

```
int randomNum =
```

Generate and store a random `integer` from `1` to `6`, *inclusive*, in `randomNum`:

```
int randomNum =
```

Generate and store a random `integer` from `25` to `30`, *inclusive*, in `randomNum`:

```
int randomNum =
```

Generate and store a random `integer` from `100` to `200`, *inclusive*, in `randomNum`:

```
int randomNum =
```

Generate and store a random `integer` from `94` to `132`, *inclusive*, in `randomNum`:

```
int randomNum =
```

Generate and store a random `integer` from `1` to `999`, *inclusive*, in `randomNum`:

```
int randomNum =
```

Generate and store a random `integer` from `-15` to `50`, *inclusive*, in `randomNum`:

```
int randomNum =
```

Confirm your answers

Below are two lines of code that are intended to generate and print two random integers between 2 and 10, inclusive of 10.

Will they both work? If not, why not? *If you aren't sure, copy/paste and run the code several times in IntelliJ **before** you answer!*

```
int rand1 = (int) (Math.random() * 9) + 2;
int rand2 = (int) Math.random() * 9 + 2;
System.out.println(rand1);
System.out.println(rand2);
```

check

LUCKY NUMBERS!

Below is an incomplete `LuckyNumbers` class:

```
public class LuckyNumbers
{
  /* No instance variables */

  /* "empty" constructor with no parameters */
  public LuckyNumbers() { }
```

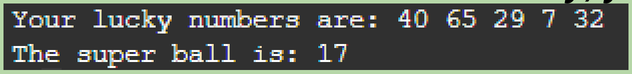
```
   /* Generates a random number between min and max, inclusive,
      and returns that random number
    */
   public int randomIntegerBetween(int min, int max) {
      /* implement this method!  */
   }

   /* Generates and returns a String containing lucky numbers!

      For this lucky number game, there are 5 balls randomly drawn,
      each between 1 and 65, and one "super ball" between 1 and 30.

      In this game, the same number CAN appear more than once.

      The returned String should have the 6 numbers listed (they do not
      need to be in ascending order): 5 "lucky numbers" between 1 and 65,
      and the last one, the "super ball," between 1 and 30.
      See samples below.

      This method should call your randomIntegerBetween method above multiple times --
      don't rewrite the same code over and over to generate multiple random numbers,
      use your method!
     */
   public String getLuckyNumbers() {
     /* implement this method!  */
   }
}
```

1. Implement the `LuckyNumbers` class above in IntelliJ by completing the two missing methods. ***Please be sure to read both method descriptions carefully!***

2. **TEST** your class and methods by writing test code to do the following:

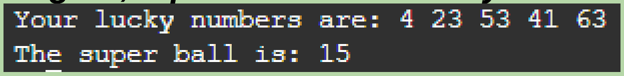
- Create a `LuckyNumbers` object (using the default, no argument constructor).
- Call the `getLuckyNumbers` method and store the returned String in a variable.
- Print the variable to make sure it looks something like the following:

If you wrote your methods and test code correctly, you should see:



```
Your lucky numbers are: 40 65 29 7 32
The super ball is: 17
```

Run again, it produces new lucky numbers!



```
Your lucky numbers are: 4 23 53 41 63
The super ball is: 15
```

Let me check my test code

Note! *Each "lucky number" should be between 1 and 65, and the "super ball" should be between 1 and 30.*

Copy/paste your code for your tested and completed `LuckyNumbers` class below:

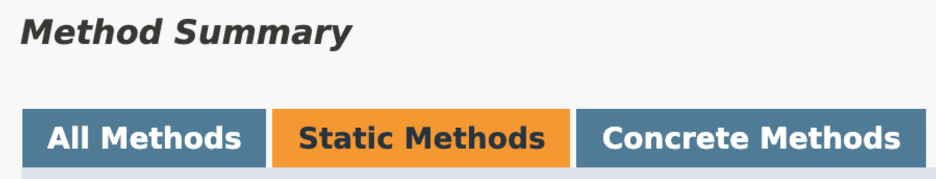
Compare

Lab concludes on the next page!

Exploring the Math Class

Locate and explore the Java API docs for the `Math` class (it's part of the **java.lang** package, along with `String` and `System`). Scroll to and click on the `Math` class in the bottom left.

Find **one or two** other static methods that sound interesting to you and click on them to learn more!



Here are a few that you might want to check out:

- Trigonometry methods: `Math.sin()`, `Math.cos()`, `Math.tan()`
- Conversion methods: `Math.toDegrees()`, `Math.toRadians()`
- Other math methods: `Math.round()`, `Math.floor()`, `Math.ceil()`
- Max/min methods: `Math.max()`, `Math.min()`

NOTE! *Only* the five `Math` methods listed on the Java Quick Reference are tested on the AP Exam, so whatever method(s) you explore now are *bonus!* But as a software engineer, you should be familiar with using documentation to learn what functionality is available to you!

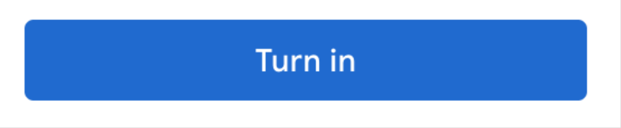
Write some code to test out the method(s) you explored.

Which `Math` method(s) did you learn about?	

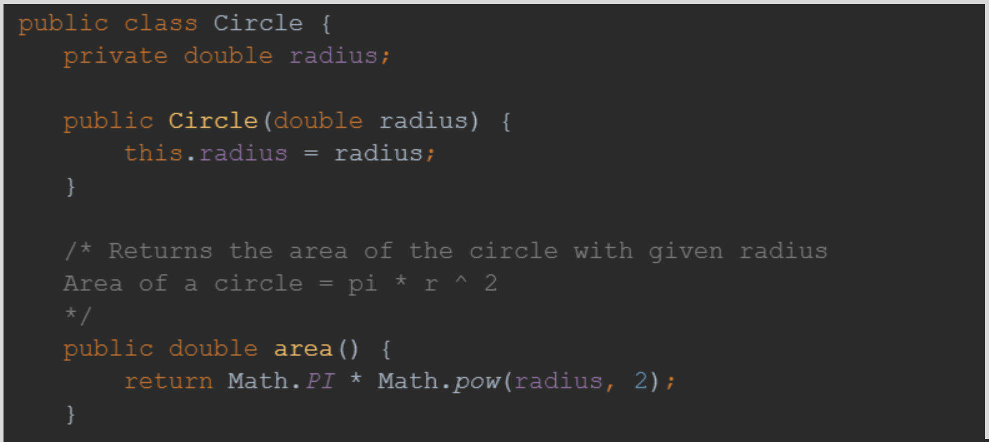
Copy/paste the code you wrote to test out the new method(s):

Done!

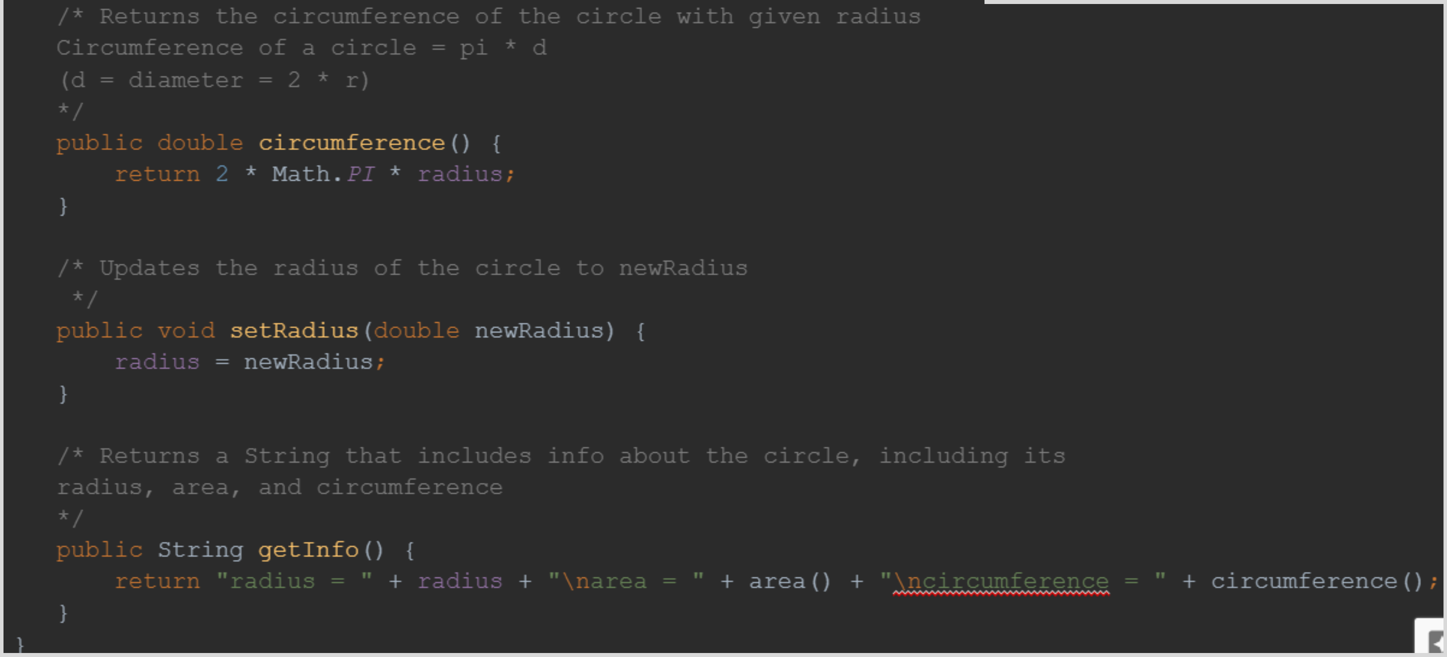
Submit in Google Classroom



Sample Solution (back)



```
public class Circle {
   private double radius;

   public Circle(double radius) {
       this.radius = radius;
   }

   /* Returns the area of the circle with given radius
   Area of a circle = pi * r ^ 2
   */
   public double area() {
       return Math.PI * Math.pow(radius, 2);
   }
```



```
   /* Returns the circumference of the circle with given radius
   Circumference of a circle = pi * d
   (d = diameter = 2 * r)
   */
   public double circumference() {
       return 2 * Math.PI * radius;
   }

   /* Updates the radius of the circle to newRadius
    */
   public void setRadius(double newRadius) {
       radius = newRadius;
   }

   /* Returns a String that includes info about the circle, including its
   radius, area, and circumference
   */
   public String getInfo() {
       return "radius = " + radius + "\narea = " + area() + "\ncircumference = " + circumference();
   }
}
```

Solution (back)

The **^** is **NOT** how you do exponents in Java! This is actually a totally different thing in Java (look up "caret operator Java" if you're curious what it does, although you don't need to know it for this class).

```
double x = 6.5;
double thirdPower = (x ^ 3);   // does NOT do exponentiation!!!
System.out.println(thirdPower);
```

This is how Sammy can fix his code:

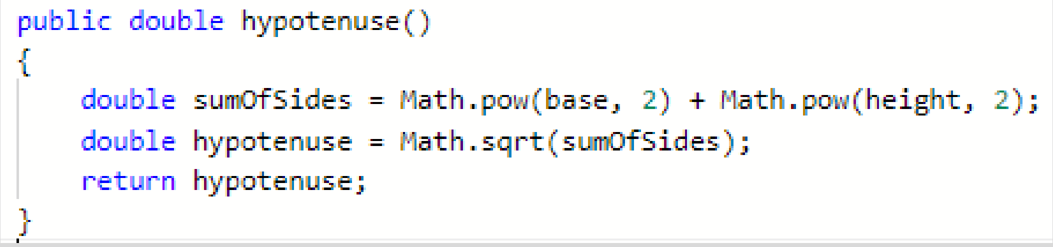
```
double x = 6.5;
double thirdPower = Math.pow(x, 3);
System.out.println(thirdPower);
```

Note: it should print out: 274.625

Sample Solution (back)

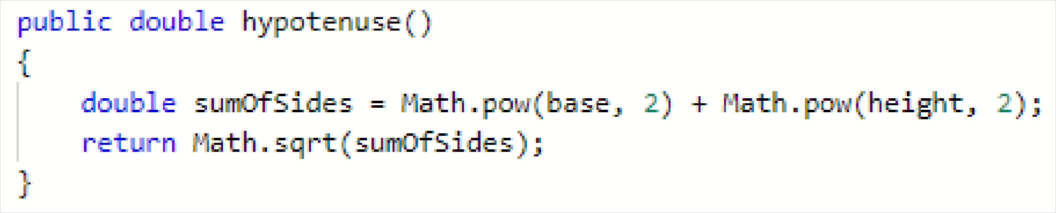
Below is one possible solution for the hypotenuse method:

In this case, “hypotenuse” is side *c* in this equation: $□ = \sqrt{□^2 + □^2}$



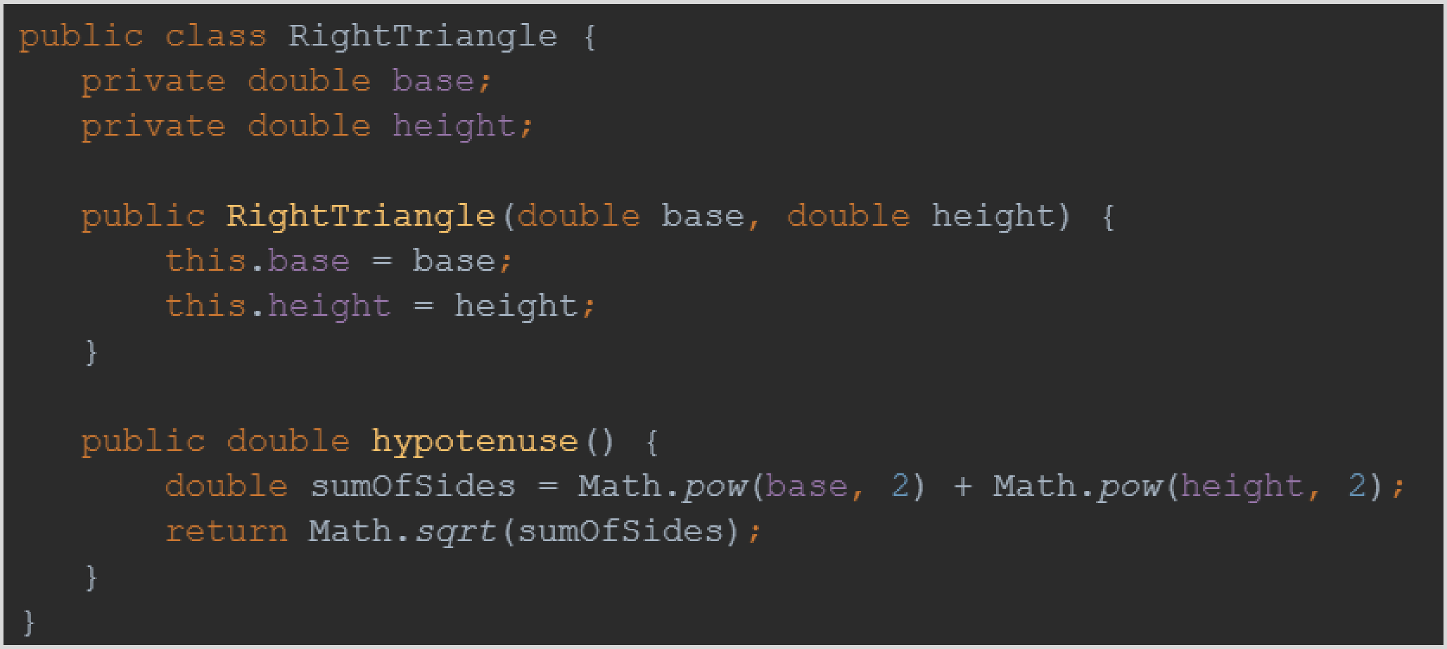
```
public double hypotenuse()
{
    double sumOfSides = Math.pow(base, 2) + Math.pow(height, 2);
    double hypotenuse = Math.sqrt(sumOfSides);
    return hypotenuse;
}
```

OR you could return the square root directly:



```
public double hypotenuse()
{
    double sumOfSides = Math.pow(base, 2) + Math.pow(height, 2);
    return Math.sqrt(sumOfSides);
}
```

Your complete class should look something like:



```
public class RightTriangle {
   private double base;
   private double height;

   public RightTriangle(double base, double height) {
       this.base = base;
       this.height = height;
   }

   public double hypotenuse() {
       double sumOfSides = Math.pow(base, 2) + Math.pow(height, 2);
       return Math.sqrt(sumOfSides);
   }
}
```

Solution (back)

Relationship:

To generate a random **`int`** between **A** and **B**, inclusive (*including* **B**)

```
randomNum = (int) (Math.random() * (B - A + 1)) + A;
```

Alternatively, referring to A as **min** and B as **max**:

```
randomNum = (int) (Math.random() * (max - min + 1)) + min;
```

So the code you can use to generate a random integer between **50 and 85**, inclusive is:

```
randomNum = (int) (Math.random() * (85 - 50 + 1)) + 50;
```

Or more simply:

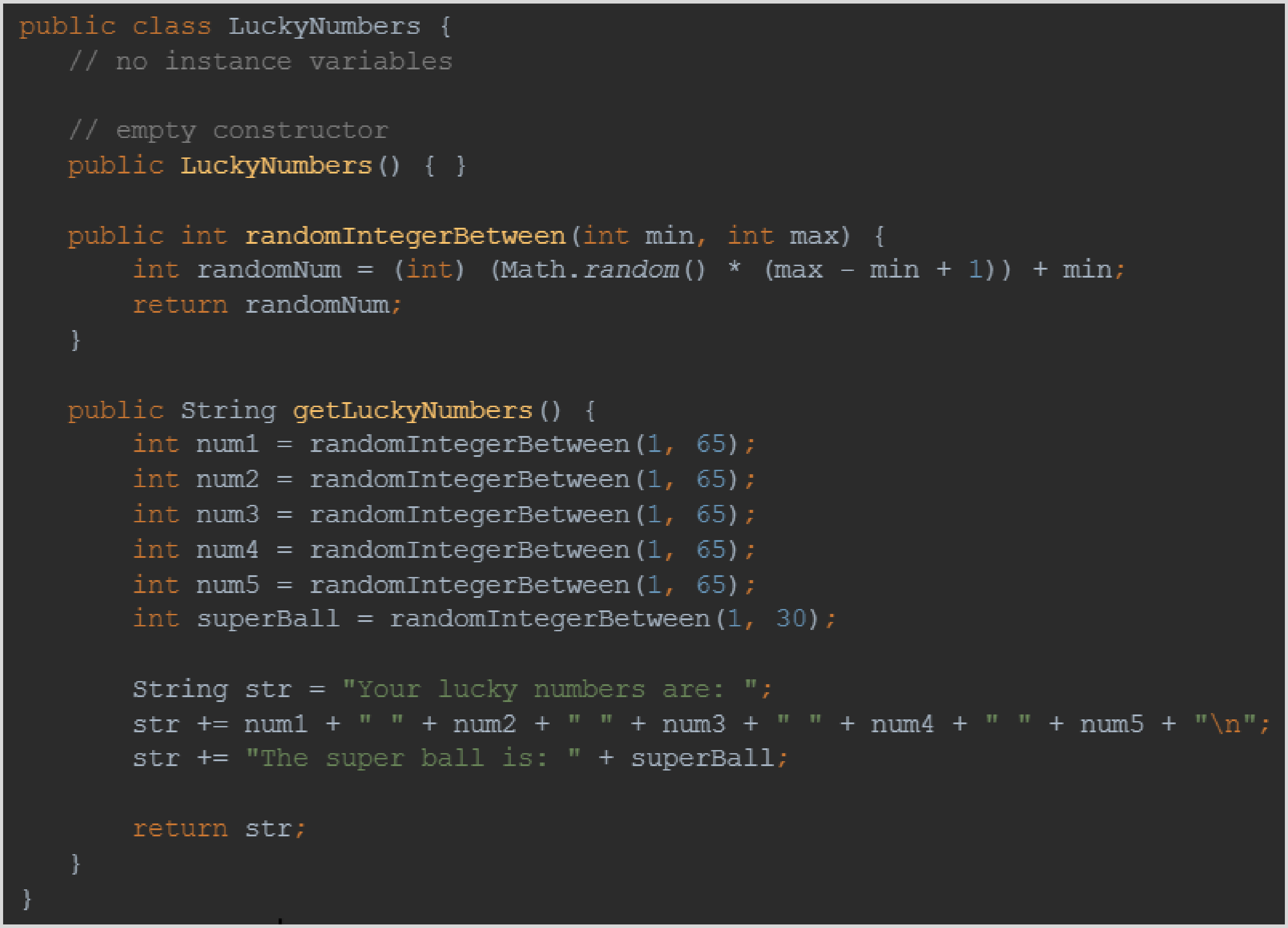
```
randomNum = (int) (Math.random() * 36) + 50;
```

Solutions (back)

```
//Generate a random int between A and B, inclusive (including B)

randomNum = (int) (Math.random() * (max - min + 1)) + min;
```

Generate and store a random `integer` from `0` to `6`, *inclusive*, in `randomNum`:
`int randomNum = (int) (Math.random() * 7) + 0` → **`(int) (Math.random() * 7)`**
Generate and store a random `integer` from `1` to `6`, *inclusive*, in `randomNum`:
`int randomNum = (int) (Math.random() * 6) + 1`
Generate and store a random `integer` from `25` to `30`, *inclusive*, in `randomNum`:
`int randomNum = (int) (Math.random() * 6) + 25;`
Generate and store a random `integer` from `100` to `200`, *inclusive*, in `randomNum`:
`int randomNum = (int) (Math.random() * 101) + 100;`
Generate and store a random `integer` from `94` to `132`, *inclusive*, in `randomNum`:
`int randomNum = (int) (Math.random() * 39) + 94;`
Generate and store a random `integer` from `1` to `999`, *inclusive*, in `randomNum`:
`int randomNum = (int) (Math.random() * 999) + 1;`
Generate and store a random `integer` from `-15` to `50`, *inclusive*, in `randomNum`:
`int randomNum = (int) (Math.random() * 66) - 15;`

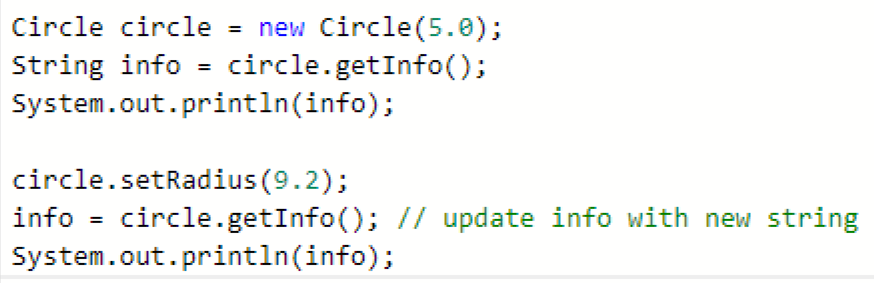
Solution (back)



```java
public class LuckyNumbers {
   // no instance variables

   // empty constructor
   public LuckyNumbers() { }

   public int randomIntegerBetween(int min, int max) {
       int randomNum = (int) (Math.random() * (max - min + 1)) + min;
       return randomNum;
   }

   public String getLuckyNumbers() {
       int num1 = randomIntegerBetween(1, 65);
       int num2 = randomIntegerBetween(1, 65);
       int num3 = randomIntegerBetween(1, 65);
       int num4 = randomIntegerBetween(1, 65);
       int num5 = randomIntegerBetween(1, 65);
       int superBall = randomIntegerBetween(1, 30);

       String str = "Your lucky numbers are: ";
       str += num1 + " " + num2 + " " + num3 + " " + num4 + " " + num5 + "\n";
       str += "The super ball is: " + superBall;

       return str;
   }
}
```

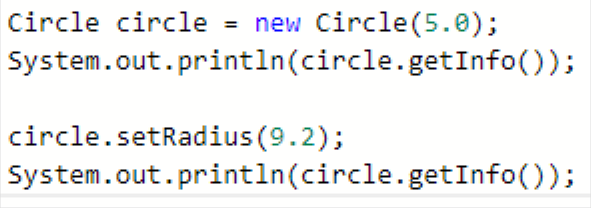
Test Code (back)

This is the test code that produces the output shown:



```
Circle circle = new Circle(5.0);
String info = circle.getInfo();
System.out.println(info);

circle.setRadius(9.2);
info = circle.getInfo(); // update info with new string
System.out.println(info);
```

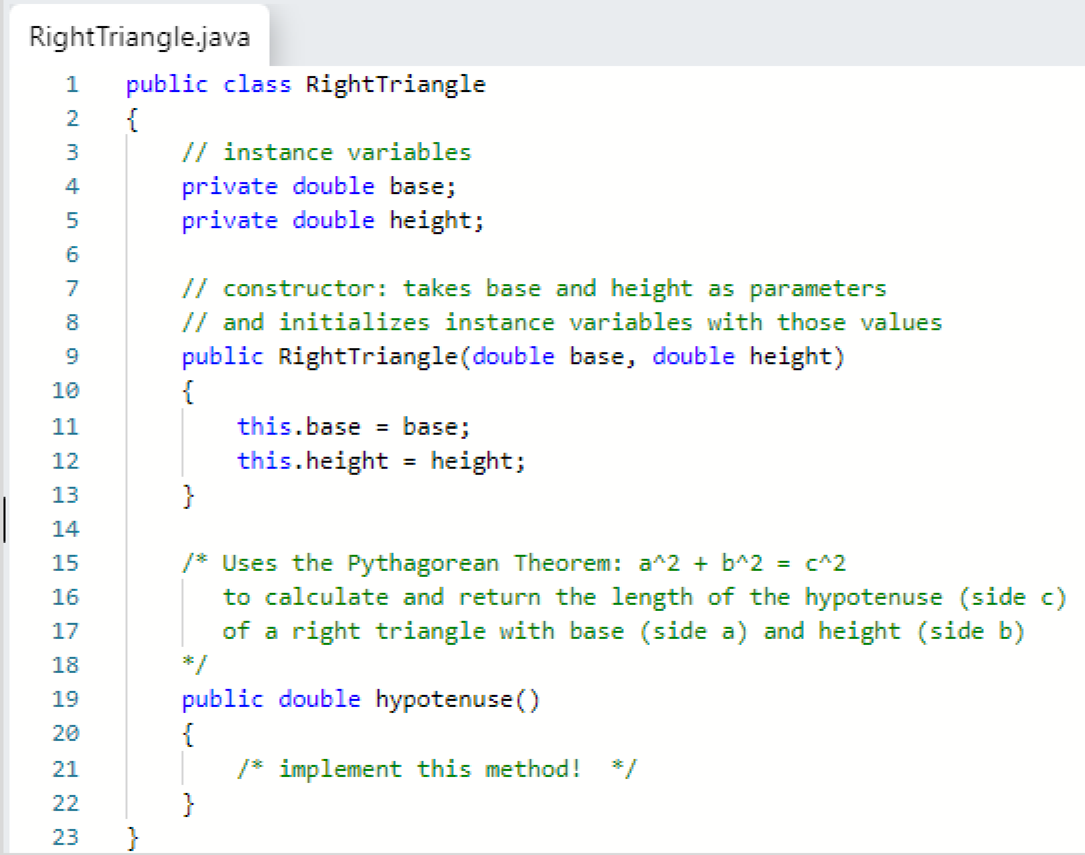
Note that you didn’t need to store the info string as a variable first, you could have just printed it directly like this:



```
Circle circle = new Circle(5.0);
System.out.println(circle.getInfo());

circle.setRadius(9.2);
System.out.println(circle.getInfo());
```

Confirm (back)

Here is how the instance variables and constructor should be set up:



```java
public class RightTriangle
{
    // instance variables
    private double base;
    private double height;

    // constructor: takes base and height as parameters
    // and initializes instance variables with those values
    public RightTriangle(double base, double height)
    {
        this.base = base;
        this.height = height;
    }

    /* Uses the Pythagorean Theorem: a^2 + b^2 = c^2
       to calculate and return the length of the hypotenuse (side c)
       of a right triangle with base (side a) and height (side b)
    */
    public double hypotenuse()
    {
        /* implement this method!  */
    }
}
```

Confirm (back)

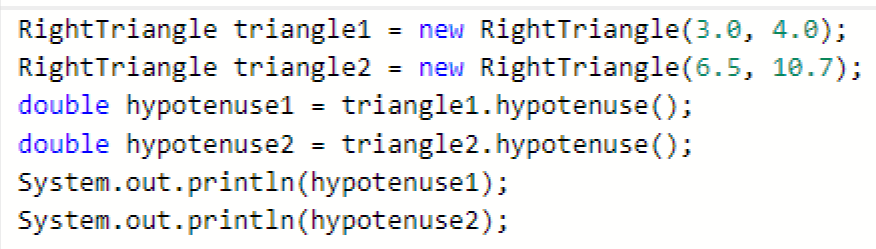
Revisit your answer to the previous question; after seeing the 5 examples above, modify your answer (if needed, otherwise leave blank!):

```
Math.random() * 9 + 1
```

Using the examples above, figure out how to write an expression to generate a random double between 25.0 and 100.0, *exclusive* of 100.0.

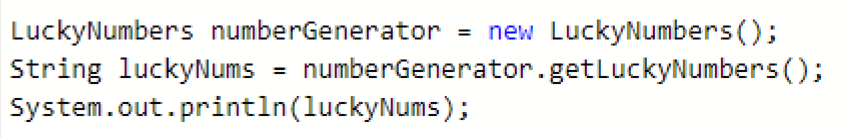
```
Math.random() * 75 + 25
```

Test Code (back)



```
RightTriangle triangle1 = new RightTriangle(3.0, 4.0);
RightTriangle triangle2 = new RightTriangle(6.5, 10.7);
double hypotenuse1 = triangle1.hypotenuse();
double hypotenuse2 = triangle2.hypotenuse();
System.out.println(hypotenuse1);
System.out.println(hypotenuse2);
```

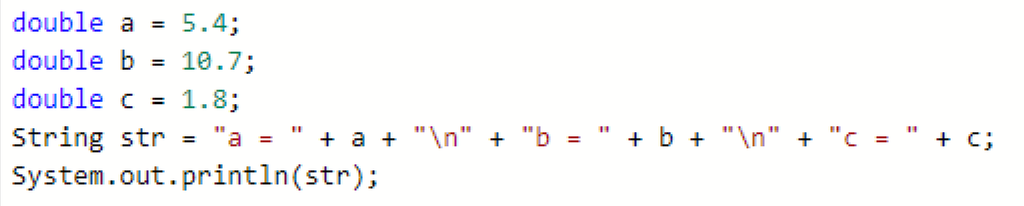
Test Code (back)



```
LuckyNumbers numberGenerator = new LuckyNumbers();
String luckyNums = numberGenerator.getLuckyNumbers();
System.out.println(luckyNums);
```

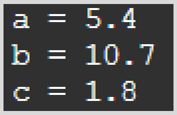
Hint (back)

This code:



```
double a = 5.4;
double b = 10.7;
double c = 1.8;
String str = "a = " + a + "\n" + "b = " + b + "\n" + "c = " + c;
System.out.println(str);
```

WIll produce a string where a, b, and c are separated by new lines:



```
a = 5.4
b = 10.7
c = 1.8
```

Answer (back)

Below are two lines of code that are intended to generate two random integers between 2 and 10, inclusive of 10.

Will they both work? If not, why not? *If you aren't sure, run them in IntelliJ before you answer!*

```
int rand1 = (int) (Math.random() * 9) + 2;
int rand2 = (int) Math.random() * 9 + 2;
System.out.println(rand1);
System.out.println(rand2);
```

They will **NOT** both work!

The **first line is correct** and `rand1` will be a correctly generated random number

The **second is not correct** as it will **ALWAYS PRODUCE 2, every time!** This is because the `(int)` casting happens on the `Math.random()` result ***before*** multiplying by 9, and since `Math.random()` returns a value less than 1.0, it will *always* truncate to 0 when cast to an `int`, and *then* 0 * 9 is 0, and finally 0 + 2 is 2, so `rand2` *will **always** be 2 -- NOT a random number!*

Oh, the importance of parentheses!