Skill Builder 3 - Loopy Loops

This Skill Builder will require you to write several functions in which loops are the focus. So, let's get started!

The class called SkillBuilder3 includes a set of skeleton methods. The requirements for each method are provided below.

String Multiplication

Implement the method with the following signature.

```
public static String repeat(String s, int numOfTimesToRepeat)
```

The method returns a string consisting of the character represented by the parameter `ch` repeated `numOfTimesToRepeat`.

Since Java does not have string arithmatic as in Python, we are forced to write a method to simulate string arithmatic.

For example, invoking the method as,

```
skillBuilder3.repeat("*", 8);
```

results in a string, `"********"`, consisting of eight asterisks.

Invoking the method,

```
skillBuilder3.repeat("%", 10);
```

results in a string, `"%%%%%%%%%%"`, consisting of ten percent (%) characters.

Invoking the method,

```
skillBuilder3.repeat("Hi", 3);
```

results in a string, `"HiHiHi"`, consisting of a string that repeats "Hi" three times.

Left Triangle

Implement the method with the following signature.

```
public static String leftRightTriangle(int height)
```

The method returns a string consisting of asterisks in a left-right triangle pattern with a height given by the parameter `height`. When the method is invoked as follows:

```
SkillBuilder3.leftRightTriangle(5);
```

the result is a string that is equivalent to the following:

```
*\n**\n***\n****\n*****\n
```

If printed,

```
System.out.println(SkillBuilder3.leftRightTriangle(5));
```

The result is:

```
*
**
***
****
*****
```

Invoking the method with a different argument,

```
SkillBuilder3.leftRightTriangle(3)
```

results in,

```
*\n**\n***\n
```

and if printed,

```
System.out.println(SkillBuilder6.leftRightTriangle(3));
```

results in,

```
*
**
***
```

NOTE: you must use the `repeat` method in your implementation of this method.

Right Triangle

Implement the method with the following signature.

```
public static String rightRightTriangle(int height)
```

The method returns a string consisting of asterisks in a right-right triangle pattern with a height given by the parameter `height`. When the method is invoked as follows:

```
skillBuilder3.rightRightTriangle(5);
```

the result is a string that is equivalent to the following:

```
    *\n   **\n  ***\n ****\n*****\n
```

If printed,

```
System.out.println(recitation7.rightRightTriangle(5));
```

The result is:

```
    *
   **
  ***
 ****
*****
```

Invoking the method with a different argument,

```
skillBuilder3.rightRightTriangle(3)
```

results in,

```
  *\n **\n***\n
```

and if printed,

```
System.out.println(skillBuilder3.rightRightTriangle(3));
```

results in

```
  *
 **
***
```

NOTE: you must use the `repeat` method in your implementation of this method.

Let's Draw a Circle - Yay!

Implement the method with the following signature.

```
public static String circle(int radius)
```

The method returns a string of asterisks representing a circle pattern with a radius given by the parameter `radius`. When the method is invoked as follows:

```
skillBuilder3.circle(5)
```

the result is a string as follows,

```
  ******\n ********\n ********\n ********\n**********\n ********\n ********
```

If printed,

```
System.out.println(recitation7.circle(5))
```

the result is,

```
  ******
 ********
 ********
 ********
**********
 ********
 ********
 ********
  ******
```

Invoking the method with a different argument,

```
skillBuilder3.circle(8)
```

the result is a string as follows,

```
"     ******\n   **********\n  ************\n  ************\n *************
```

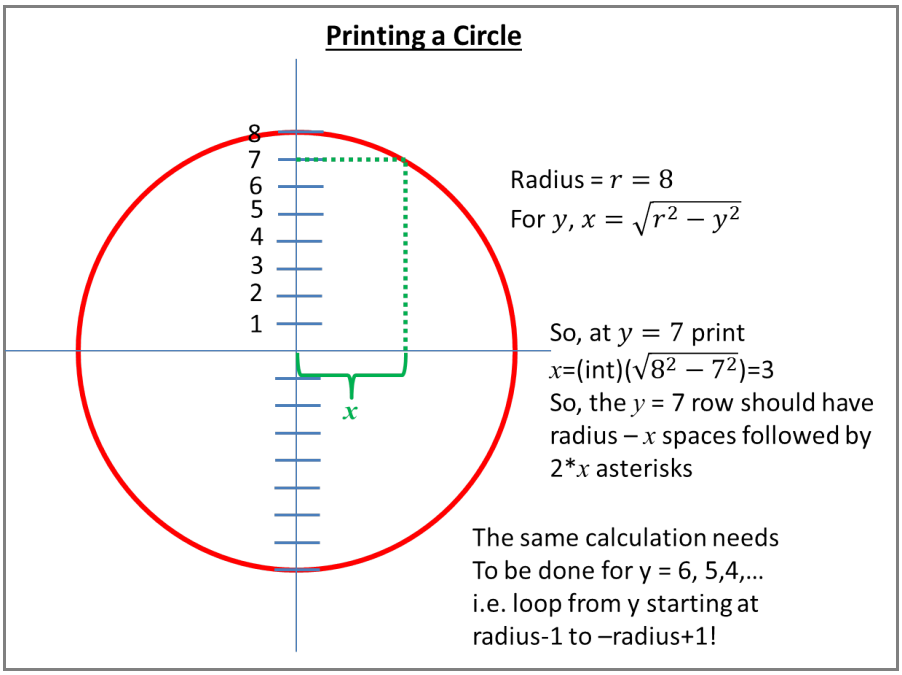
If printed,

```
System.out.println(skillBuilder3.circle(8))
```

the result is,

```
     ******
   **********
  ************
  ************
 **************
 **************
 **************
****************
 **************
 **************
 **************
  ************
  ************
   **********
     ******
```

An explanation of how to calculate the number of asterisks to print on each line is provided in the illustration below.



Note that you do not need to calculate x for $y = r$ or for $y = -r$ since for both these values, $x = 0$. So, you can use a for-loop that goes from `radius-1` to `-radius+1`, inclusively or `radius-1` to `-radius`, not including `-radius`.

NOTE: you must use the `repeat` method in your implementation of this method.

Sum of All Divisors of a Number

Implement the method with the following signature.

```
public static long sumOfDivisors(long number)
```

The method returns a long integer that is the sum of all the divisors of the parameter `number` but does not include `number`.

For example, invoking the method as follows,

```
skillBuilder3.sumOfDivisors(10)
```

results in,

```
8
```

because the divisors of 10 (not including 10) are 1, 2, 5, and their sum is `1 + 2 + 5 = 8`.

Another example,

```
skillBuilder3.sumOfDivisors(56)
```

results in,

```
64
```

because the divisors of 56 (not including 56) are 1, 2, 4, 7, 8, 14, 28 and their sum is `1 + 2 + 4 + 7 + 8 + 14 + 28 = 64`.

NOTE: you must implement this method using a `for` loop!

Perfect Numbers

Let's use the method designed above (`sumOfAllDivisors`) to determine if a number is perfect. A *perfect number* is a positive integer equal to the sum of its positive divisors, not including the number itself.

Implement the method with the following signature.

```
public static boolean isPerfect(long number)
```

The method returns `true` if the parameter `number` is a perfect number; otherwise `false`.

For example, invoking the method as follows,

```
skillBuilder3.isPerfect(10)
```

results in, `false` because the divisors of 10 (not including 10) are 1, 2, 5, and their sum is `1 + 2 + 5 = 8` is not equal to `10`.

Another example,

```
skillBuilder3.sumOfDivisors(6)
```

results in `true` because the divisors of 6 (not including 6) are 1, 2, 3, and their sum is `1 + 2 + 3 = 6` is equal to `6`.

Sum of All Divisors of a Number Using a While loop

Implement the `sumOfDivsors` method using a `while` loop. Implement this requirement in the method with the following signature,

```
public static long sumOfDivisorsUsingWhile(long number)
```

NOTE: you must implement this method using a `while` loop!