**Approximate Pi**

Open BlueJ, and create a new BlueJ project titled **Lab24-ApproximatePi** in your CS 1 folder.

Create a new class, open it (double click), delete BlueJ's starter code, and **type** in our code skeleton.

A method is, essentially, a named block of code. We run a method by calling it (typing in its name and supplying it the information it needs to run - the method's parameters). Methods got OUTSIDE OF OTHER METHODS BUT INSIDE THE CLASS, and have the following parts:

public static String sampleMethod(int a)

{

//something

}

|  |  |
| --- | --- |
| public static | access modifier (for now, this will always be public static). |
| String | the return type. The **type** of data this method will return (to where it's called) when it's finished running. |
| sampleMethod | method name. Naming convention is the same as variables. |
| int a | method's parameter. Placeholder variable for the information passed in to the method. |

As always, more information is available on our [website.](http://bit.ly/mrbunnCS)

**Before each problem, insert a COMMENT with the problem number.**

1. Complete the method: public static boolean isPositive(int a), that returns true if parameter a is positive. [Here](https://youtu.be/LXZBN2-Wls4) is how you should organize your method calls for testing.
2. Complete the method: public static boolean isOdd(int a), that returns true if the value of the parameter a is odd.
3. Complete the method: public static int getMax(int a, int b), that returns the larger of the two numbers (return the value if they're the same).
4. (Riddle) 1 = A N of H
5. Complete the method: public static int rollDice(), that returns a random number between 1 and 6, representing the roll of a die.
6. Complete the method: public static int rollDice(int numFaces), that returns a random number between 1 and numFaces, representing a roll of a die with a non-standard number of faces.

/\* When two methods have the same name but different parameter lists, they are referred to as being "overloaded". The best example is String's substring() methods (same method name, has a two-parameter version and a one-parameter version) \*/

1. Complete the method: public static String capitalizeFirst(String name) that returns the supplied String with the first letter capitalized. You can assume the name parameter will already be lower case. //mansi => Mansi, null => null
2. Complete the method: public static double circleArea(double radius), that returns the area of a circle with the supplied radius. Circum = 2\*pi\*r ==pi\*d(d=2\*r), Area = pi\* r\*r
3. Complete the method: public static boolean isMultiple(int a, int b), that returns true if b is a multiple of a*.*
4. (Riddle) 4 = Y in one P T
5. Complete the method: public static String convertTime(double time, boolean isMinutes), that converts seconds to minutes and vice versa. The boolean parameter isMinutes will be supplied as true if time is in minutes. The method should return a String representation like the examples below (use String concatenation):

convertTime(3.0, true) >>> 3.0 minutes is 180.0 seconds

convertTime(180.0, false) >>> 180.0 seconds is 3.0 minutes

1. Complete the method: public static void printDiagonal(String s), that prints the String parameter in the following fashion (shown below with the String s passed in as "hello"). **Note: this method has a** void **return type, meaning it does not return any data (e.g. an integer or a boolean). You can't print what this method returns, but you CAN simply call the method (it already contains print statements).**

h

e

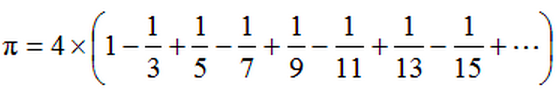
l

l

o

**Approximate Pi**

Write a method that returns an **approximation** of the value of pi. The value of pi can be approximated by calculating the result of the following 'infinite' series (a sequence of numbers that continues forever):



Your method should take a single parameter, an integer called iterations, that specifies the number of ***iterations*** (number of loops) that should be performed in the calculation. If iterationswas supplied as 4, only the first four terms in the series would be used in the calculation (1 – 1/3 + 1/5 – 1/7).

Your method should return the value of the calculation as a double. For large numbers of iterations, your method should return a value that is very close to pi. Sample method call (with iterations of ten million):

approximatePi(10\_000\_000) >>> 3.1415925535897915

/\* you can separate groups of zeros in a large integer with an underscore (not commas, commas indicate another parameter) \*/

**(Advanced) Triangle numbers**

A sequence of 'triangle numbers' is generated by adding all the natural numbers up to and including that number. The 7th triangle number would be 1+2+3+4+5+6+7 (== 28). The first 10 triangle numbers would be:

1, 3, 6, 10, 15, 21, 28, 36, 45, 55

If we list the **factors** of the first 7 triangle numbers:

1: 1

3: 1, 3

6: 1, 2, 3, 6

10: 1, 2, 5, 10

15: 1, 3, 5, 15

21: 1, 3, 7, 21

28: 1, 2, 4, 7, 14, 28

We can see that 28 is the first triangle number to have over 5 factors.

**What is the value of the first triangle number to have over 500 divisors?**

//Note – if your program is taking a long time to run, you need to think of ways you can optimize your solution. A great solution should take only a few seconds, a good solution will take under a minute. You can see the run time of your program by adding the following code:

double start = System.currentTimeMillis();

/\*

\* Code goes here

\*/

double end = System.currentTimeMillis();

System.out.println("Run time = " + (end - start) + "ms");