**Lab 25 – PiCalculator**

Open BlueJ, and create a new BlueJ project titled **Lab25-PiCalculator** in your CS\LABS folder.

Create a new class and start with this code:

//Name:

import java.util.\*;

public class PracticeProblems

{

public static void main(String[] args)

{

}

}

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. |

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

1. Complete the method: public static boolean isOdd(int a), that returns true if the value of the parameter is odd.

**What should you put in your main method to test/run this method?** Add this code:

out.println(5 + " is Odd? >>> " + isOdd(5));

out.println(6 + " is Odd? >>> " + isOdd(6));

1. Complete the method: public static boolean isSingleDigit(int a), that returns true if the int parameter is only 1 digit.
2. 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). (You are basically writing your own version of Math.max(). Don’t call Math.max(), write your own version. You know… for fun!)
3. (Riddle) 1 = A N of H
4. Complete the method: public static int rollDice(), that returns a random number between 1 and 6, representing the roll of a die.
5. 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.

**NOTE:** Even though the method in problem 6 has the same name as the method in problem 5, they have different signatures. You will write TWO methods called rollDice. The method in problem 5 has no parameters, while the method in problem 6 requires an integer parameter. Even though the methods have the same name, Java still knows which version of the method you want to use based on what parameters you pass in.

1. Complete the method: public static String capitalizeOnlyFirst(String name) that returns the supplied String with the first letter capitalized and all other letters lowercase.
2. Complete the method: public static double circleArea(double radius), that returns the area of a circle with the supplied radius.
3. Complete the method: public static boolean isMultiple(int a, int b), that returns true if b is a multiple of a*.* (Example: 35 is a multiple of 5.)
4. (Riddle) 4 = Y in one P T
5. Complete the method: public static boolean isPrime(long num), that returns true if the supplied parameter is a prime number.

//**long** is a data type. A long is essentially an int that can hold much larger values

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"):

h

e

l

l

o

1. Complete “Worksheet - Methods #1” worksheet.

**Pi Calculator**

Create a new class and start with this code:

//Name:

import java.util.\*;

public class PiCalculator

{

public static void main(String[] args)

{

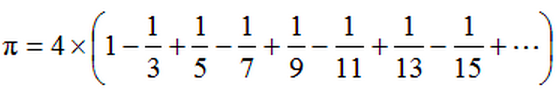
System.out.println(“Series of 10 >>> “ + approximatePi(10));

System.out.println(“Series of 1000 >>> “ + approximatePi(1000));

}

}

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 precision, that specifies how many **iterations** (number of loops) should be performed in the calculation. If precisionwas passed in 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. Sample output (based on method calls in skeleton code):

Series of 10 >>> 3.0418396189294032

Series of 1000 >>> 3.140592653839794

**(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");