CSE21: Lab #5 – Methods Overloading

Overview

You will declare your own methods for the first time in this lab and fill in the details. You can figure out what the declaration should be by the way we are invoking such methods in *main*. We will walk you through the steps to do this lab so try to follow the directions and build on each subsequent step.

Recall the different parts to declaration methods:

- Name
- Input Parameters (type and name)
- Return value (type)

We invoke or call the method declared by using Name (input1, input2 …) so on and if the method does not take in any arguments then we use empty parentheses (). Try to answer the questions as you encounter them instead of saving them till the end since that will help you solve each step before moving onto the next exercise before you are ready.

Getting started

After starting Eclipse, create a new project called Lab 21_5. Import into your project Lab21_5.java.

(Exercise) Fill-in – Lab21_5.java

We provide a full solution sample to sumAll in lecture and in this lab to get you started. Basic method only takes in one argument: max and it will return the sum of all the numbers starting at 1 up to and including max.

```
public static long sumAll(int max) {
      long subTotal = 0;
      for (int i = 1; i <= max; i++) {
            subTotal += i;
            System.out.println("sumAll " + i + " value " + subTotal);
      }
      return subTotal;
}
```

Q1. What is the return type of *sumAll()* ?

We store the temporary result into the variable subTotal and print out each time we do the calculation. This way we can trace how many times the loop is executed every time we invoke sumAll. If we look in main we see the following calls to sumAll:

```
System.out.println("sumAll output for 5 is " + sumAll(5));
System.out.println("sumAll output for 10 is " + sumAll(10));
System.out.println("sumAll output for 20 is " + sumAll(20));
System.out.println("sumAll output for 15 is " + sumAll(15));
```

We use parameters 5, 10, 20 and 15 in sequence to illustrate the redundant calculations being done. We can use the same method name with different input arguments to solve the same problem but in a more efficient way. We do that by introducing an array pointer variable in addition to max.

```
public static long sumAll(long[] arr, int max) {
      for (int i = 1; i <= max ; i++) {
            if (arr[i] == 0) {
                  arr[i] = arr[i-1] + i;
                  System.out.println("sumAll[" + i + "] value " + arr[i]);
            }
      }
      return arr[max];
}
```

We will store all the calculations in the array that is given and those are usually called partial results. Go ahead and run the Lab21_6.java and you will see the output matching the expected output for sumAll.

Q2. What relation must be true between length of *arr* and *max* for this method to work correctly?

Example runs:

```
sumAll 1 value 1
sumAll 2 value 3
sumAll 3 value 6
sumAll 4 value 10
sumAll 5 value 15
sumAll output for 5 is 15
sumAll 1 value 1
sumAll 2 value 3
sumAll 3 value 6
sumAll 4 value 10
sumAll 5 value 15
sumAll 6 value 21
sumAll 7 value 28
sumAll 8 value 36
sumAll 9 value 45
sumAll 10 value 55
sumAll output for 10 is 55
sumAll 1 value 1
sumAll 2 value 3
sumAll 3 value 6
sumAll 4 value 10
sumAll 5 value 15
sumAll 6 value 21
sumAll 7 value 28
sumAll 8 value 36
sumAll 9 value 45
sumAll 10 value 55
sumAll 11 value 66
```

```
sumAll 12 value 78
sumAll 13 value 91
sumAll 14 value 105
sumAll 15 value 120
sumAll 16 value 136
sumAll 17 value 153
sumAll 18 value 171
sumAll 19 value 190
sumAll 20 value 210
sumAll output for 20 is 210
sumAll 1 value 1
sumAll 2 value 3
sumAll 3 value 6
sumAll 4 value 10
sumAll 5 value 15
sumAll 6 value 21
sumAll 7 value 28
sumAll 8 value 36
sumAll 9 value 45
sumAll 10 value 55
sumAll 11 value 66
sumAll 12 value 78
sumAll 13 value 91
sumAll 14 value 105
sumAll 15 value 120
sumAll output for 15 is 120

sumAll[1] value 1
sumAll[2] value 3
sumAll[3] value 6
sumAll[4] value 10
sumAll[5] value 15
sumAll output for 5 is 15
sumAll[6] value 21
sumAll[7] value 28
sumAll[8] value 36
sumAll[9] value 45
sumAll[10] value 55
sumAll output for 10 is 55
sumAll[11] value 66
sumAll[12] value 78
sumAll[13] value 91
sumAll[14] value 105
sumAll[15] value 120
sumAll[16] value 136
sumAll[17] value 153
sumAll[18] value 171
sumAll[19] value 190
sumAll[20] value 210
sumAll output for 20 is 210
sumAll output for 15 is 120
```

Exercise 1: Implement squareSum

Similarly to sumAll, squareSum will take in one parameter: *max* and figures out the sum of all the squares starting 1 and up to including max^2. For example if max is 5 then this method returns 55 by doing 1+4+9+16+25.

You will need to implement both versions of the method just like sumAll.

One version only takes one parameter and the other version takes an array as an additional argument. Declare the methods as appropriate in the space given in the

file Lab21_5.java. Uncomment the invocation of *squareSum()* methods inside *main* to get the output that matches the expected output.

Q3. What is the return type of *squareSum()* methods?

Q4. What is the use of init(arr) call before squareSum(arr, 5)?

Example runs:

```
squareSum 1 value 1
squareSum 2 value 5
squareSum 3 value 14
squareSum 4 value 30
squareSum 5 value 55
squareSum output for 5 is 55
squareSum 1 value 1
squareSum 2 value 5
squareSum 3 value 14
squareSum 4 value 30
squareSum 5 value 55
squareSum 6 value 91
squareSum 7 value 140
squareSum 8 value 204
squareSum 9 value 285
squareSum 10 value 385
squareSum output for 10 is 385
squareSum 1 value 1
squareSum 2 value 5
squareSum 3 value 14
squareSum 4 value 30
squareSum 5 value 55
squareSum 6 value 91
squareSum 7 value 140
squareSum 8 value 204
squareSum 9 value 285
squareSum 10 value 385
squareSum 11 value 506
squareSum 12 value 650
squareSum 13 value 819
squareSum 14 value 1015
squareSum 15 value 1240
squareSum 16 value 1496
squareSum 17 value 1785
squareSum 18 value 2109
squareSum 19 value 2470
squareSum 20 value 2870
squareSum output for 20 is 2870
squareSum 1 value 1
squareSum 2 value 5
squareSum 3 value 14
squareSum 4 value 30
squareSum 5 value 55
squareSum 6 value 91
squareSum 7 value 140
squareSum 8 value 204
squareSum 9 value 285
squareSum 10 value 385
squareSum 11 value 506
squareSum 12 value 650
squareSum 13 value 819
squareSum 14 value 1015
```

```
squareSum 15 value 1240
squareSum output for 15 is 1240

squareSum[1] value 1
squareSum[2] value 5
squareSum[3] value 14
squareSum[4] value 30
squareSum[5] value 55
squareSum output for 5 is 55
squareSum[6] value 91
squareSum[7] value 140
squareSum[8] value 204
squareSum[9] value 285
squareSum[10] value 385
squareSum output for 10 is 385
squareSum[11] value 506
squareSum[12] value 650
squareSum[13] value 819
squareSum[14] value 1015
squareSum[15] value 1240
squareSum[16] value 1496
squareSum[17] value 1785
squareSum[18] value 2109
squareSum[19] value 2470
squareSum[20] value 2870
squareSum output for 20 is 2870
squareSum output for 15 is 1240
```

Exercise 2: Implement Fibonacci

This method also takes in the parameter *max* and returns the Fibonacci number corresponding to the position of *max*. Fibonacci sequence consists of numbers that are the sum of the previous two numbers. Starting from the position 0, we get the sequence.

Position	0	1	2	3	4	5	6	7	8
Fib Num	0	1	1	2	3	5	8	13	21

So for position 6, it takes value of position 5 and 4 together to add them up.
So this method would return 8 by doing 3+5.
You will need to implement both versions of the method just like sumAll.
One version only takes one parameter and the other version takes an array as an additional argument. Declare the methods as appropriate in the space given in the file Lab21_5.java. Uncomment the invocation of *fib()* methods inside *main* to get the output that matches the expected output.

Q5. What is the return type of *fib()* methods?

Q6. Why does Fibonacci output start at 2?

Example runs:

```
Fibonacci 2 value 1
Fibonacci 3 value 2
Fibonacci 4 value 3
Fibonacci 5 value 5
```

```
Fibonacci Number 5 is 5
Fibonacci 2 value 1
Fibonacci 3 value 2
Fibonacci 4 value 3
Fibonacci 5 value 5
Fibonacci 6 value 8
Fibonacci 7 value 13
Fibonacci 8 value 21
Fibonacci 9 value 34
Fibonacci 10 value 55
Fibonacci Number 10 is 55
Fibonacci 2 value 1
Fibonacci 3 value 2
Fibonacci 4 value 3
Fibonacci 5 value 5
Fibonacci 6 value 8
Fibonacci 7 value 13
Fibonacci 8 value 21
Fibonacci 9 value 34
Fibonacci 10 value 55
Fibonacci 11 value 89
Fibonacci 12 value 144
Fibonacci 13 value 233
Fibonacci 14 value 377
Fibonacci 15 value 610
Fibonacci 16 value 987
Fibonacci 17 value 1597
Fibonacci 18 value 2584
Fibonacci 19 value 4181
Fibonacci 20 value 6765
Fibonacci Number 20 is 6765
Fibonacci 2 value 1
Fibonacci 3 value 2
Fibonacci 4 value 3
Fibonacci 5 value 5
Fibonacci 6 value 8
Fibonacci 7 value 13
Fibonacci 8 value 21
Fibonacci 9 value 34
Fibonacci 10 value 55
Fibonacci 11 value 89
Fibonacci 12 value 144
Fibonacci 13 value 233
Fibonacci 14 value 377
Fibonacci 15 value 610
Fibonacci Number 15 is 610

Fibonacci[2] value 1
Fibonacci[3] value 2
Fibonacci[4] value 3
Fibonacci[5] value 5
Fibonacci Number 5 is 5
Fibonacci[6] value 8
Fibonacci[7] value 13
Fibonacci[8] value 21
Fibonacci[9] value 34
Fibonacci[10] value 55
Fibonacci Number 10 is 55
Fibonacci[11] value 89
Fibonacci[12] value 144
Fibonacci[13] value 233
Fibonacci[14] value 377
```

```
Fibonacci[15] value 610
Fibonacci[16] value 987
Fibonacci[17] value 1597
Fibonacci[18] value 2584
Fibonacci[19] value 4181
Fibonacci[20] value 6765
Fibonacci Number 20 is 6765
Fibonacci Number 15 is 610
```

Exercise 3: Implement Factorial

This method also takes in the parameter *max* and returns the factorial of it.
For example if max is 5 then it returns 120 by doing 1*2*3*4*5.
You will need to implement both versions of the method just like sumAll.
One version only takes one parameter and the other version takes an array as an additional argument. Declare the methods as appropriate in the space given in the file Lab21_5.java. Uncomment the invocation of *factorial()* methods inside *main* to get the output that matches the expected output.

Q7. What is the return type of *factorial()* methods?

Q8. Why does Factorial output start at 2?

Example runs:

```
Factorial 2 value 2
Factorial 3 value 6
Factorial 4 value 24
Factorial 5 value 120
Factorial of 5 is 120
Factorial 2 value 2
Factorial 3 value 6
Factorial 4 value 24
Factorial 5 value 120
Factorial 6 value 720
Factorial 7 value 5040
Factorial 8 value 40320
Factorial 9 value 362880
Factorial 10 value 3628800
Factorial of 10 is 3628800
Factorial 2 value 2
Factorial 3 value 6
Factorial 4 value 24
Factorial 5 value 120
Factorial 6 value 720
Factorial 7 value 5040
Factorial 8 value 40320
Factorial 9 value 362880
Factorial 10 value 3628800
Factorial 11 value 39916800
Factorial 12 value 479001600
Factorial 13 value 6227020800
Factorial 14 value 87178291200
Factorial 15 value 1307674368000
Factorial 16 value 20922789888000
Factorial 17 value 355687428096000
Factorial 18 value 6402373705728000
Factorial 19 value 121645100408832000
Factorial 20 value 2432902008176640000
Factorial of 20 is 2432902008176640000
```

```
Factorial 2 value 2
Factorial 3 value 6
Factorial 4 value 24
Factorial 5 value 120
Factorial 6 value 720
Factorial 7 value 5040
Factorial 8 value 40320
Factorial 9 value 362880
Factorial 10 value 3628800
Factorial 11 value 39916800
Factorial 12 value 479001600
Factorial 13 value 6227020800
Factorial 14 value 87178291200
Factorial 15 value 1307674368000
Factorial of 15 is 1307674368000

Factorial[2] value 2
Factorial[3] value 6
Factorial[4] value 24
Factorial[5] value 120
Factorial of 5 is 120
Factorial[6] value 720
Factorial[7] value 5040
Factorial[8] value 40320
Factorial[9] value 362880
Factorial[10] value 3628800
Factorial of 10 is 3628800
Factorial[11] value 39916800
Factorial[12] value 479001600
Factorial[13] value 6227020800
Factorial[14] value 87178291200
Factorial[15] value 1307674368000
Factorial[16] value 20922789888000
Factorial[17] value 355687428096000
Factorial[18] value 6402373705728000
Factorial[19] value 121645100408832000
Factorial[20] value 2432902008176640000
Factorial of 20 is 2432902008176640000
Factorial of 15 is 1307674368000
```

Q9. Why is `long[] arr = new long[MAXSIZE];` declared as array of longs?
(Could it be array of int instead?)

Q10. Change MAXSIZE to 10, does everything still work?
If not, how can you fix it?

What to hand in

When you are done with this lab assignment, you are ready to submit your work. Make sure you have done the following before you press Submit:

- Include answers to questions (Q1-Q10)
- Attach Lab21_5.java
- List of Collaborators