In this lab, we’ll gain practice working with the two mental models of computation that we’ve talked about so far: the substitutive model of computation and the stack-heap model of computation. We’ll use them to predict the behavior of Java code fragments of increasing complexity.

Directions

You should work on these problems with your lab partner, write down the solutions, and have the instructor or mentors check your work when you are done. You should be done with most, if not all, of the problems by the end of the class period. Please write down both your name and your partner’s name on this submission.

Part 1: The Substitutive Model

For each of the following Java expressions, answer the following questions:

1. What is the type of the expression?
2. What is the exact step-by-step sequence of evaluation steps this expression takes?
3. What is the final value that this expression evaluates to?

Some expressions may give type errors at compile-time. Please identify those and state where the error resides.

Double check your work by writing small programs that print out the results of these expressions. Note that `System.out.println` allows you to print out any value, not just strings!

For reference, the Java operator precedence table can be found on this Java tutorial page: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html.

1. \( 9 - 15 / (6 + 4 \times 2) \)

2. \( 12 / 5 - 2 \times 2 \% 3 \)
3. \[ \text{public static boolean compare (int } x, \text{ int } y) \{ \\
\quad \text{return !(} x \geq y); \\
\} \]

\begin{verbatim}
  \text{compare(5, 2 + 3) || compare(5+1, 2+5)}
\end{verbatim}

4. \[ 8 / (7 + 3.0) \]

5. \[ "hello" + "world" \]

6. \[ "hello " + "plus " + 5 \]

7. \[ "hello " + "times " \times 5 \]
8. $5 + "oh" + 3.0 + true + "hi"

9. $3.0 + true + "oh" + 5 + "hi"

Part 2: The Stack-and-Heap Model  For each of the following Java programs (Mystery1 and Mystery2), give stack-and-heap diagrams describing the state of memory at each point indicated in the program below. The format of your stack-and-heap diagrams should be identical to the format presented in the reading. You should create a completely new diagram for each part (i.e., not re-use older diagrams).

Part 3: Arrays and the Stack-and-Heap Model  As briefly mentioned in the reading, values of variables with non-primitive type, i.e., not int, char, boolean, float, or double, exist on the heap. More specifically, strings and arrays exist on the heap and variables contain references or pointers to those values rather than the actual values themselves. This has important implications when you pass in arrays to functions. In contrast, strings in Java are immutable, i.e., they cannot be modified, so they effectively act like primitive values even though they exist on the heap.

Like Part 2, give stack-and-heap diagrams for each of the points in the programs (Mystery3 and Mystery4) annotated below.

(For these stack-and-heap diagrams, feel free to break up the work—each member of the group can do a separate diagram in parallel. When you are finished, you should present and explain your work to the rest of the group. This serves two purposes: (1) the group learns from you, (2) the group checks your work.)
public class Mystery1 {
    public static double mystery(int x, int y) {
        // Point (C)
        int z = x + y;
        x = y;
        y = z;
        // Point (D)
        return (double) x - z;
    }
    public static void main(String[] args) {
        // Point (A)
        int x = 0;
        int y = 5;
        // Point (B)
        double d1 = mystery(x, y);
        // Point (E)
    }
}
public class Mystery2 {
    // This is (effectively) a global variable declaration
    // You can place it in the heap for now.
    public static int glob = 5;

    public static int mystery(int x) {
        // Points (B) (on first call to mystery)
        glob += x;
        x = glob;
        // Points (D) (on second call to mystery)
        return glob - 5;
    }

    public static void main(String[] args) {
        int x = 0;
        // Point (A)
        int y = mystery(x);
        // Point (C)
        glob = x;
        y = mystery(x)
        // Point (F)
    }
}
public class Mystery3 {
    public static void mystery(int[] arr, int x) {
        x = 0;
        arr[0] = arr[5];
        arr[1] = arr[2];
        arr[3] = arr.length;
    }
}

public static void main(String[] args) {
    int x = 5;
    int[] xs = {0, 1, 2, 3, 4, 5};
    // Point (A)
    mystery(xs, x);
    // Point (B)
}
public class Mystery4 {
    public static void mystery(int[] arr, int n) {
        // Points (B) - (D) (every call to mystery)
        if (n >= 0) {
            arr[n] = n * 2;
            mystery(arr, n-1);
        }
    }

    public static void main(String[] args) {
        int[] arr = new int[3];
        // Point (A)
        mystery(arr, arr.length - 1);
        // Point (E)
    }
}