Practice Exam 1

Types

Choose the type for the variable that would allow the line to compile, or write "compilation error" if there is an error in the expression that makes its type undefined.

Statement	Data Type of x or Error Type
`______ x = "apple" + 123;`	String
`______ x = 7 < 5 and 3 > 2;`	Error
`______ x = true;`	boolean
`______ x = "true";`	String
`______ x = "true && 2 < 4";`	String
`______ x = true || 3 < 4 && "yes".equals("no");`	boolean
`______ x = (int) (4.0 * 5);`	int
`______ x = new double[10];`	double[]

For each statement, indicate whether it will result in a compilation error, runtime error, or specify the data type of variable x.

Values

Write the value that gets printed, or write "runtime error" if there is an error during the execution of these lines of the program.

```
String str1 = "Hello";
String str2 = "World";
System.out.println(str1 + " " + str2);
```

ANSWER: Hello World

```
int[] numbers = {5, 10, 15, 20};
int result = 0;
for (int i = 0; i < numbers.length; i++) {
    result += numbers[i] / 2;
```

```
}
System.out.println(result);
```

ANSWER: 24

```
int[] numbers = {5, 10, 15, 20};
int result = 0;
for (int i = 0; i < numbers.length; i++) {
    result += numbers[i / 2];
}
System.out.println(result);
```

ANSWER: 30

```
System.out.println((char) ('A' + 2));
```

ANSWER: 'C' (67 would be acceptable although it's not what is printed)

```
System.out.println("A" + 2);
```

ANSWER: A2

```
double[] numbers = {4.1, 0, -13.1};
numbers[(int) numbers[2]] = numbers[2];
System.out.println(numbers[0]);
```

ANSWER: Error (`(int) numbers[2]` is -13, which is not a valid index.)

Tracing

Here's a class that features a few functions.

```
public class TracingExercise {
    public static void main(String[] args) {
        System.out.println("Starting main");
        int x = 7;
        int y = 4;
        int z = functionA(x, y);
        System.out.println("The final result is: " + z);
    }
```

```
    public static int functionA(int a, int b) {
        System.out.println("functionA arguments: a=" + a + ", b=" + b);
        int result1 = a * 2;
        int result2 = functionC(result1, b);
        int finalResult = functionC(result2, 9);
        System.out.println("functionA returning: " + finalResult);
        return finalResult;
    }

    public static int functionB(int num) {
        System.out.println("functionB argument: num=" + num);
        int result = num - 3;
        System.out.println("functionB returning: " + result);
        return result;
    }

    public static int functionC(int x, int y) {
        System.out.println("functionC arguments: x=" + x + ", y=" + y);
        int t = x / 2;
        int u = y % 3;
        int v = -9;

        if (u == 0) {
            v = functionB(t);
        } else {
            v = t + u;
        }

        System.out.println("functionC returning: " + v);
        return v;
    }
}
```

When the program is run with `java TracingExercise`, **ten** lines are printed. For each of the following lines, fill in the blanks to show what values the variables have when they are printed out. Also, mark the order in which they are printed. Some lines are printed more than once, and so there are multiple rows in the table for those lines. The order for the first line is marked for you.

Printed Line	Order
`Starting main`	0

Printed Line	Order
`functionA arguments: a=7, b=4`	1
`functionA returning: 1`	8
`functionB argument: num=4`	5
`functionB returning: 1`	6
`functionC arguments: x=14, y=4`	2
`functionC returning: 8`	3
`functionC arguments: x=8, y=9`	4
`functionC returning: 1`	7
`The final result is: 1`	9

Debugging

Swiper the Fox loves apples. One day, Swiper stumbles upon n picnic tables all conveniently in a line, all of which have some number of apples on them (so lucky)! Swiper also has an irresistible urge to swipe (steal) items, but since he is feeling quite hungry, he decides to eat half the apples at each table as well!

Swiper is also feeling especially mischievous, so along with eating the apples, he is also going to reverse the order of the tables.

For instance, an example of the picnic tables could look as follows where each integer element is the number of apples on that table.

```
int[] tables = {0, 2, 4, 6, 8, 10};
int eatenApples = reverseAndSwipe(tables); // returns 15 to represent th
for (int i = 0; i < tables.length; i++) {
    System.out.print(tables[i] + ", ");
}
// prints 5, 4, 3, 2, 1, 0,
```

Below is the code implementation of Swiper's behavior. However, there are several bugs in the code. Your job is to determine the lines that contain bugs in the code, summarize the issue, and provide a fix that should be made to accurately reflect Swiper's actions! `reserveAndSwipe()` should return the number of apples Swiper has eaten.

Note: You may assume there are an even number of apples on each table for the sake of integer division.

```
1.    public static int[] reverseAndSwipe(int[] tables) {
2.      int eaten = tables.length;
3.      for (int i = 0; i <= tables.length; i++) {
4.          int leftTableApples = tables[i];
5.          int rightTableApples = tables[tables.length - i];
6.          eaten += leftTableApples / 2;
7.          if (leftTableApples != rightTableApples) {
8.              eaten += rightTableApples / 2;
9.              tables[i] = rightTableApples / 2;
10.         }
11.
12.         tables[tables.length - i - 1] = leftTableApples / 2;
13.     }
14.     if (tables.length % 2 == 1) {
15.         eaten += tables[tables.length / 2] / 2;
16.         tables[tables.length / 2] /= 2;
17.     }
18.     eaten;
19.   }
```

Line Number	Brief Description	Fix
1	wrong return type	`int`
2	wrong initial value for `eaten`	`int eaten = 0;`
3	only need to do the swaps for the first half of the array	`i <= tables.length / 2`
5	off by one error	`tables[tables.length - i - 1]`
18	missing `return`	`return eaten;`

Coding!

A business' daily sales revenues for a week can be represented as a `double[]` ("an array of doubles"). The length of the array can vary between 2 and 7, since the business may not be open every day of the week. A sales week is **typical** if the first and last days have the top two highest revenues in the week. A sales week is **stable** if there is no day with revenue more than $100 lower than the average revenue for the week. (If the average revenue for a week is $840 and there is a day with $700 in revenue, the week would not be stable.) Write the following three functions: `isTypical`, `greatestInstability`, and `isAtypicalOrUnstable`. Note the

function headers & signatures for each that describe how they should behave: your functions need to return values of the correct types to receive credit!

```
/*
 * Input: a double[] representing the week's revenue to analyze
 * Output: true if the week is considered "typical"
 * and false otherwise.
 *
 * A week's revenue is "typical" if the first and last reported days
 * have the top two highest revenues in the week.
 */
public static boolean isTypical(double[] revenues) {
    double smallerDayOfFirstAndLast;
    if (revenues[0] >= revenues[revenues.length - 1]) {
        smallerDayOfFirstAndLast = revenues[revenues.length - 1];
    } else {
        smallerDayOfFirstAndLast = revenues[0];
    }

    for (int i = 0; i < revenues.length; i++) {
        if (revenues[i] > smallerDayOfFirstAndLast) {
            return false;
        }
    }
    return true;
}
```

```
/*
 * Input: a double[] representing the week's revenue to analyze
 * Output: the largest difference between any day's revenue and
 * the week's average revenue.
 *
 * For a given week {800, 600, 700, 900, 1000}, the average revenue is $
 * For each day, the difference between revenue and average is:
 *                  {0,  -200,-100, 100,  200}
 * So we would return -200.
 */
public static double greatestInstability(double[] revenues) {
    double sum = 0;
    for (int i = 0; i < revenues.length; i++) {
        sum += revenues[i];
    }
```

```
    double average = sum / revenues.length;
    double largestNegativeDeviation = Double.POSITIVE_INFINITY;
    for (int i = 0; i < revenues.length; i++) {
        if (revenues[i] - average < largestNegativeDeviation) {
            largestNegativeDeviation = revenues[i] - average;
        }
    }
    return largestNegativeDeviation;
}


/*
 * Input: a double[] representing the week's revenue to analyze
 * Output: true if the week is not typical or if the if the week has a
 * day with revenue at least $100 lower than the average weekly revenue,
 * or both.
 *
 * Use the previous two functions you wrote to complete this one in **on
 */
public static boolean isAtypicalOrUnstable(double[] revenues) {
    return !isTypical(revenues) || greatestInstability(revenues) <= -100
}
```