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	Solution
`______ x = "yes" + Integer.parseInt("45");`	`String`
`______ x = 5 > 8 > 6;`	error
`______ x = (int) "apple".charAt(0) - 'a';`	`int`
`______ x = true && 3 < 4 && !"yes".equals("no");`	`boolean`
`______ x = (int) Math.random() * 5 - 4.0;`	`double`
`______ x = new char[43]`	`char[]`
`______ x = 4.0 / 3 % 17 == "yes".equals("no")`	error

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. If the answer is a `String` or `char`, make sure to write the literal with the appropriate quotation marks (" and ', respectively).

```
System.out.println(Integer.parseInt("3") / Double.parseDouble("4"));
```

ANSWER: 0.75

```
int[] x = {1, 2, 3};
x[x.length - 1] = x[x.length - x[0]] - x[1];
System.out.println(x[2]);
```

ANSWER: 1

```
int[] arr = {1, 2, 3};
System.out.println(arr[3]);
```

ANSWER: runtime error

```
System.out.println("10" + 20);
```

ANSWER: 1020

```
double[] numbers = {4.1, 0, -13.1};
double x = 0;
for (int i = 0; i < numbers.length; i++) {
    x = x * numbers[i];
}
System.out.println(x);
```

ANSWER: 0

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 = 10;
        int y = 5;
        int z = funcOne(y, x);
        System.out.println("The final result is: " + z);
    }

    public static int funcOne(int a, int b) {
        System.out.println("funcOne arguments: a=" + a + ", b=" + b);
        int result1 = a + b;
        int result2 = funcTwo(result1);
        int result3 = funcThree(a, b);
        int finalResult = result2 - result3;
        System.out.println("funcOne returning: " + finalResult);
        return finalResult;
    }

    public static int funcTwo(int num) {
        System.out.println("funcTwo argument: num=" + num);
        int result = num * 2;
        System.out.println("funcTwo returning: " + result);
        return result;
```

```
    }

    public static int funcThree(int x, int y) {
        System.out.println("funcThree arguments: x=" + x + ", y=" + y);
        int result1 = x * 3;
        int result2 = y * 2;
        int finalResult = result1 + result2;
        System.out.println("funcThree returning: " + finalResult);
        return finalResult;
    }
}
```

When the program is run with `java TracingExercise`, seven 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. The order for the first line is marked for you.

Printed Line	Order
`Starting main`	1
`"The final result is: ______"`	
`funcOne arguments: a=_____, b=______`	
`funcOne returning: ________`	
`funcTwo argument: num=________`	
`funcTwo returning: ________`	
`funcThree argument: x=______, y=______`	
`funcThree returning: ________`	

Printed Line	Order
`Starting main`	1
`"The final result is: -5"`	8
`funcOne arguments: a=5, b=10`	2
`funcOne returning: -5`	7
`funcTwo argument: num=15`	3
`funcTwo returning: 30`	4
`funcThree argument: x=5, y=10`	5

Printed Line	Order
`funcThree returning: 35`	6

Fill in the blanks

The following program is supposed to print out every fifteen minutes of the day, starting with `12:00 AM`, then `12:15 AM`, then `12:30 AM` and so on. Times in the second half of the day should be printed using the 12-hour clock, meaning that the full set of printed times should include 96 timestamps and should look exactly like the following:

```
12:00 AM
12:15 AM
12:30 AM
12:45 AM
1:00 AM
...       // these ellipses are not printed literally;
11:30 AM  // they are here only for abbreviation.
11:45 AM
12:00 PM
12:15 PM
...
11:45 AM
```

Help write this clock program by filling in the blanks.

```
public class Clock {
    public static void main(String[] args) {
        for (int hour = 0; hour < 24; hour++) {
            for (int minute = 0; minute < 60; minute += 15) {
                if (hour == 0 || hour == 12) {
                    System.out.print("12");
                } else {
                    System.out.print(hour % 12);
                }
                if (minute == 0) {
                    System.out.print(":00");
                } else {
                    System.out.print(":" + minute);
                }
                if (hour < 12) {
                    System.out.println(" AM");
```

```
                } else {
                    System.out.println(" PM");
                }
            }

        }
    }
}
```

Coding!

A password can be represented by a `String`. A password is said to be **strong** if it contains an uppercase letter (a `char` between `'A'` and `'Z'`), a lowercase letter (a `char` between `'a'` and `'z'`), and a digit (a `char` between `'0'` and `'9'`). A password is said to be **valid** if it does not contain any forbidden characters: `' '` (a space) or `'-'` (a hyphen). Finally, a password is **good** if it is both strong and valid. Write the following three functions: `isStrong`, `indexOfInvalidCharacter`, and `isGood`. 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 String representing the password to test
 * Output: true if the password is considered "strong"
 * and false otherwise.
 *
 * A password is "strong" if it contains an uppercase
 * letter, and a lowercase letter, and a digit character.
 */
public static boolean isStrong(String password) {
    boolean hasLower = false;
    boolean hasUpper = false;
    boolean hasDigit = false;

    for (int i = 0; i < password.length(); i++) {
        char currentChar = password.charAt(i);
        if (currentChar >= 'a' && currentChar <= 'z') {
            hasLower = true;
        } else if (currentChar >= 'A' && currentChar <= 'Z') {
            hasUpper = true;
        } else if (currentChar >= '0' && currentChar <= '9') {
            hasDigit = true;
        }
```

```
    }

    return hasLower && hasUpper && hasDigit;
}


/*
 * Input: a String representing the password to test
 * Output: the index of the position of the *first* invalid character,
 * or -1 if the String contains no invalid characters.
 *
 * A password is "valid" if it does not contain
 * either of these two characters: ' ' or '-'.
 */
public static boolean indexOfInvalidCharacter(String password) {
    for (int i = 0; i < password.length(); i++) {
        char currentChar = password.charAt(i);
        if (currentChar == ' ' || currentChar == '-') {
            return i;
        }
    }

    return -1;
}


/*
 * Input: a String representing the password to test
 * Output: true if the password is both strong and valid, false otherwis
 *
 * A password is good if it is both strong and valid. Use the previous t
 */
public static boolean isGood(String password) {
    return isStrong(password) && indexOfInvalidCharacter(password) == -1
}
```