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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>______ x = &quot;yes&quot; + Integer.parseInt(&quot;45&quot;);</td>
<td>String</td>
</tr>
<tr>
<td>______ x = 5 &gt; 8 &gt; 6;</td>
<td>error</td>
</tr>
<tr>
<td>______ x = (int) &quot;apple&quot;.charAt(0) - 'a';</td>
<td>int</td>
</tr>
<tr>
<td>______ x = true &amp;&amp; 3 &lt; 4 &amp;&amp; !&quot;yes&quot;.equals(&quot;no&quot;);</td>
<td>boolean</td>
</tr>
<tr>
<td>______ x = (int) Math.random() * 5 - 4.0;</td>
<td>double</td>
</tr>
<tr>
<td>______ x = new char[43]</td>
<td>char[]</td>
</tr>
<tr>
<td>______ x = 4.0 / 3 % 17 == &quot;yes&quot;.equals(&quot;no&quot;)</td>
<td>error</td>
</tr>
</tbody>
</table>

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.

```java
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;
    }
}
```
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.

<table>
<thead>
<tr>
<th>Printed Line</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting main</td>
<td>1</td>
</tr>
<tr>
<td>&quot;The final result is: ______&quot;</td>
<td></td>
</tr>
<tr>
<td>funcOne arguments: a=_<strong><strong>, b=</strong></strong>__</td>
<td></td>
</tr>
<tr>
<td>funcOne returning: __________</td>
<td></td>
</tr>
<tr>
<td>funcTwo argument: num=________</td>
<td></td>
</tr>
<tr>
<td>funcTwo returning: __________</td>
<td></td>
</tr>
<tr>
<td>funcThree argument: x=_<strong><strong>, y=</strong></strong>__</td>
<td></td>
</tr>
<tr>
<td>funcThree returning: __________</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Printed Line</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting main</td>
<td>1</td>
</tr>
<tr>
<td>&quot;The final result is: -5&quot;</td>
<td>8</td>
</tr>
<tr>
<td>funcOne arguments: a=5, b=10</td>
<td>2</td>
</tr>
<tr>
<td>funcOne returning: -5</td>
<td>7</td>
</tr>
<tr>
<td>funcTwo argument: num=15</td>
<td>3</td>
</tr>
<tr>
<td>funcTwo returning: 30</td>
<td>4</td>
</tr>
<tr>
<td>funcThree argument: x=5, y=10</td>
<td>5</td>
</tr>
</tbody>
</table>
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
...
11:30 AM  // these ellipses are not printed literally;
11:45 AM // they are here only for abbreviation.
12:00 PM
12:15 PM
...
11:45 AM

Help write this clock program by filling in the blanks.

```java
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(":").append(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;
}

public static int indexOfInvalidCharacter(String password) {
    for (int i = 0; i < password.length(); i++) {
        char currentChar = password.charAt(i);
        if (currentChar == ' ' || currentChar == '-') {
            return i;
        }
    }
    return -1;
}

public static boolean isGood(String password) {
    return isStrong(password) && isValid(password);
}

private static boolean isValid(String password) {
    int count = 0;
    for (int i = 0; i < password.length(); i++) {
        char currentChar = password.charAt(i);
        if (currentChar == ' ' || currentChar == '-') {
            return false;
        }
        count++;
    }
    return count >= 8;
}
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;
}

public static boolean isGood(String password) {
    return isStrong(password) && indexOfInvalidCharacter(password) == -1
}

/*
* Input: a String representing the password to test
* Output: true if the password is both strong and valid, false otherwise
* A password is good if it is both strong and valid. Use the previous two
*/