Lab 3 : Programming with Decisions and Loops

Topics covered mixed variable types in expressions, Boolean logic, the `&&`,`||` and `%` operators, desk checking, `if`/`else` statements, `while` loops

---

Important!

One of the most important parts of programming is checking to see if your code performs in the way you expect it to. For each of the programming exercises in this week's lab, perform a **desk check** on your answer using various inputs to test the program's behaviour.

Remember to try normal, abnormal and boundary values for the input and, most importantly, make sure you do this on paper! Just running the code with the desired inputs does not tell you whether the internal workings of your program are performing as expected.

Drawing a **state diagram** of the program can also allow you to see flaws in your logic, and will help you choose which values to test in your desk check.

There are some operators you need for this lab:

`%` The modulus operator is a binary operator that returns the remainder when the first operand ("operand" = "thing being operated on") is divided by the second. Here's an example:

```
1  int x = 20;
2  int y = 4;
3  if (x % y == 0) {
4     System.out.println(y + " is a factor of " + x);
5  }
```

The modulus operator gives the remainder when one integer is divided by another, and is often used to check whether a number is even.

`&&` This is the logical "and" operator: used to join two Boolean expressions, the resulting expression is true exactly when both the two smaller expressions are true. For example, if A is true and B is false, then A `&&` B is false. If A and B are both true, then A `&&` B is true.

`||` This is the logical "or" operator: used to join two Boolean expressions, the resulting expression is true if either, or both, of the two smaller expressions is true.

Exercise 1: Given the following variable declarations and initialisations, evaluate the expressions below and indicate the data type of the expression.

```
1  int count = 8;
2  double size = 12.0;
3  int index = 12;
4  boolean flag = false;
5  String str1 = "Hello";
6  String str2 = "World";
```

`size / count` _____

`index / count` _____

`index <= size` _____

`count != 8` _____

`size + index / count` _____

`!(index == size)` _____

`count > 8 || !flag` _____

`str1 + str2` _____

`index % 5 <= 3` _____

`(count - 2 > index % 7) && (flag || size < 20)` _____

`str1.length() + size` _____

`flag || (count == index) || str1.equals(str2)` _____

`str1.charAt(3) == str2.charAt(3)` _____

Exercise 2: Write a program that reads an integer number from the user and tests to see if the number supplied is both *even* and lies in the range 20 – 200 inclusive, or is both *odd* and negative. For example, '5' fails this test because while it's odd, it's not negative. '22' passes the test because it's both even and in the right range.

Draw a state diagram for this program.

Casting is the act of treating a variable (or result of an expression) as another data type temporarily. The syntax is given by `(<type>)(<expression>)`, where `<type>` is the data type you wish to convert to, and `<expression>` is the variable or expression you wish to temporarily convert.

Note that if `<expression>` is just a single variable, it does not need to have brackets around it.

Remember: when you cast an expression to a certain data type, this does not affect the variables in memory. It only treats the variable (or result of an expression) as another data type for the duration of that statement. For example:

```
1   int x = 5;
2   int y = 2;
3
4   // This will output 2, as we are performing integer division
5   System.out.println(x / y);
6
7   // Will output 2.5, as the x is cast to double for the duration
8   // of the calculation
9   System.out.println((double)x / y);
10
11  // Will output 2.0, as the division is still integer division, but
12  // the result is cast to a double
13  System.out.println((double)(x / y));
```

Exercise 3: Write a program that determines whether an input integer is even — but you can't use the modulus operator `%`: for this one you'll have to use *casting*. There are several ways to do this.

It may be helpful to draw a diagram of the logic part of this exercise to see if you can figure out how to do it.

For the next five exercises, you will be creating and modifying a program called `NumberCrunch`, which reads in numbers from the user and then outputs information about those numbers.

Exercise 4: Part 'A' Create a `NumberCrunch` class, and make it read in **up to** three integers from the user. As soon as the user inputs a negative number, the program should stop trying to read numbers. Display to the user how many positive numbers were read in.

Exercise 5: Part 'B' If the user has not entered any positive numbers, `NumberCrunch` should tell the user to input at least one positive number.

```
> java NumberCrunch
   Please enter up to three positive numbers:
   -1
   You have not entered any positive numbers. Please input at least one positive number.
```

Exercise 6: Part 'C' If the user has entered exactly two positive numbers, `NumberCrunch` should print out the product of the two numbers, as well as their relationship (equal, greater than, less than).

```
> java NumberCrunch
   Please enter up to three positive numbers:
   2 6 -1
   You have entered 2 positive numbers.
   Their product is 12, and 2 is less than 6.
> java NumberCrunch
   Please enter up to three positive numbers:
   10 10 -1
   You have entered 2 positive numbers.
   Their product is 100, and they are equal.
```

Exercise 7: Part 'D' If the user has entered exactly three positive numbers, `NumberCrunch` should print out the largest of the three numbers.

```
> java NumberCrunch
   Please enter up to three positive numbers:
   10 30 20 -1
   You have entered 3 positive numbers.
   The largest is 30.
```

Exercise 8: Part 'E' If the user has entered exactly one positive number, `NumberCrunch` should print out all factors of that number. Remember: y is a factor of x if, when you divide x by y, there's no remainder. You should use the *modulus* operator `%` for this.

You will have to use a loop for this exercise.

```
> java NumberCrunch
   Please enter up to three positive numbers:
   12 -1
   You have entered 1 positive number.
   The factors of 12 are: 1, 2, 3, 4, 6, 12.
```

Exercise 9: Write a program that reads in numbers (not just integers) from the user until they enter zero. The program should then output the sum and the average of the numbers entered (excluding the zero).

Extensions

Extension: Modify your `NumberCrunch` program so that when the user inputs a single number, the program prints out all **prime** factors of that number. Prime factorisation is not as simple as regular factorisation!

Extension: Modify your `NumberCrunch` program so not all the functionality is built in the `main` method. Write three methods, that will begin like this:

```
public static void printFactors(int a) {
    // ... your code here
}
```

```
public static void printProductAndRelationship(int a, int b) {
    // ... your code here
}
```

```
public static void printMax(int a, int b, int c) {
    // ... your code here
}
```

Your new `main` method will *call* these other methods depending on whether there are 1, 2 or 3 integer numbers to deal with. For example you could do this:

```
if (numPositiveInputs == 2) {
    printProductAndRelationship(a, b);
    // assuming you already had got values for a and b
}
```