CMP5332: Lab1 – (Part 2)

Exercise 1 - Sequencing statements

Just like Python, in Java we can write multiple statements one after the other sing the `System.out.prinln()` method.

Task1: Create a new class named `FreshPrince`, and use the `System.out.prinln()` method to print the text below:

```
Yo, this is a story all about how
My life got flipped, turned upside-down
And I'd like to take a minute,
just sit right there
I'll tell you how I became the prince
of a town called Bel-Air
```

Remember that each statement such as `System.out.println("Yo, this is a story all about how");` must end with a semicolon.

Task 2: Add the following variables to your program, use them as part of your code to complete the text of the song:

```
String homeTown = "West Philadelphia";
String place = "On the playground";
String activity = "shootin' some b-ball";

In West Philadelphia, born and raised
On the playground was where I spent most of my days
Chillin' out, maxin', relaxin' all cool
And all shootin' some b-ball outside of the school
```

Task 3: Declare another variable named `newTown` with the value `"Bel-Air"`, and rewrite the part of the program which should use this variable. Make sure to declare and use the variable at the right place in your code!

Note: The method `System.out.print`, is like `System.out.println` except it doesn't print a "newline" character. This means the next time something is printed, it will continue on the same line.

Naming conventions

The names of variables, classes and methods all follow the standard conventions for names in Java:

- Variable names with one letter or one word, like `x`, `y`, `pi`, `letter`, `name`, `age`, `grade`, `place` and `activity`, are lowercase.
- Variable names with multiple words, like `gigaWatts`, `isBlue`, `phoneNumber` and homeTown, start with a lowercase letter and use uppercase letters on every other word.
- Method names follow the same rules as variable names.
- Class names like HelloWorld and `FreshPrince`, begin with an uppercase letter, and use uppercase letters on every new word.
- In Java names are not separated with underscores.

Following these conventions makes it easier for other people to understand your code, because they'll know whether something is a variable or a class. The naming conventions also help you understand Java's built-in names — for example, String and System are classes because they begin with uppercase letters, but int is not a class.

If statements

Java has `if` statements which run part of the code on some logical condition. This works just like in Python — the only difference is syntax. Here's what an `if` statement looks like in Java:

```
if(age >= 18) {
    System.out.println("You're an adult");
}
```

This code checks if a variable named `age` holds a value which is `18` or higher. If so, it prints `"You're an adult"`; otherwise it does nothing. The condition `age >= 18` is in brackets, `()`, and the conditional block is in braces, `{}`.

Technically, the braces are not required when there is only one statement inside. However, you should **always** write the braces anyway, because it makes your code easier to understand, and avoids mistakes and bugs.

To have more branches in an `if` statement, use `else if` and `else`. Again, this is the same as in Python except for syntax. Write the following program in Eclipse:

```
class AgeChecker {
    static void main() {
        double age = 19.5;

        if(age >= 18) {
            System.out.println("You're an adult");
        } else if(age >= 4) {
            System.out.println("You're a child");
        } else if(age >= 2) {
            System.out.println("You're a toddler");
        } else {
            System.out.println("You're a baby");
        }
    }
}
```

Exercise 2

Add a new class `AgeChecker`, compile and run this class to make sure the output is as you expect it to be for the different conditions.

Change some of the conditions, for example:

- A person is considered child if their age is greater than or equal to 5 and less than 12
- A toddler age is equal to or greater than 3 and less than or equals 5, and
- below the age of 3 is considered an infant.
- If the person age is equal to or over 68 then he is a pensioner
- Young adults age is greater than or equals12 and less than 18
- If the age is negative return a message saying that the age is invalid

While loops

To run part of the code repeatedly on some logical condition, use a `while` loop. These are the same as in Python, just with different syntax; for example, this loop prints a sequence of numbers, starting at `n` and doubling until it reaches `1024`:

```
while(n <= 1024) {
    System.out.print(n + ", ");
    n *= 2;
}
```

Just like an `if` statement, there should be brackets around the condition `n <= 1024`, and braces around the loop block.

Add a new class and write the following program in Eclipse:

```
class Doubler {
    static void main() {
        int n = 1;

        while(n <= 1024) {
            System.out.print(n + ", ");
            n *= 2;
        }
    }
}
```

Exercise 3

- Compile and run it to make sure the outputs are as expected
- Set your counter to start at 3 and go up to multiples of fives instead of 2's
- Change the maximum iterations to 2000 and replace the commas with dots

For loops

Java has two kinds of `for` loop:

- The "basics `for` loop" is meant for iterating over a sequence of numbers.
- The "enhanced `for` loop" iterates over a sequence or collection of values; we will see this later when we learn about arrays and collections in Java.

The "basic `for` loop" looks like this:

```
for(int i = 0; i < 10; i++) {
    System.out.print(i + ", ");
}
```

This loop prints the integers from `0` to `9`. There are three important parts:

- Initialisation: declare a variable to hold the current number, which starts as the first number in the sequence. Here, the variable is named `i`, and the first number is `0`.
- Condition: check that the current number is still within the desired range. Here, the condition is `i < 10` because we want to print the numbers up to (but not including) `10`.
- Update: advance to the next number. Here, `i++` increases the value of `i` by `1`.

The following program uses a for loop to print the text of the song "`Ten Green Bottles`". Write it in Eclipse:

```
class TenGreenBottles {
    static void main() {
        for(int n = 10; n > 0; n--) {
            System.out.println(n + " green bottles, hanging on the wall");
            System.out.println(n + " green bottles, hanging on the wall");
            System.out.print("And if one green bottle ");
            System.out.println("should accidentally fall");
            System.out.print("There'll be " + (n-1) + " green bottles, ");
            System.out.println("hanging on the wall");
            System.out.println();
        }
    }
}
```

The `n--` operation decreases the value of `n` by `1`, and the empty `println` statement prints a blank line between verses of the song. Compile and run the program to see the output.

Exercise 4 - Putting it together

The `TenGreenBottles` program would be better if it printed "1 green bottle" instead of "1 green bottles". To fix it, you should:

- Declare a variable named word.
- Use an `if/else` statement with the correct condition to decide whether to assign word the value `"bottle"` or `"bottles"`.
- Rewrite the `println` statements to use the variable word **everywhere** the word bottle needs to be printed in the text.
- You will need to handle the cases for `n` and `n-1` separately.