Pratical Exercices N° 2 - Debriefing

Nom	Prenom
BADAROU	Mikdaam

Exercice 1

1. Considering the following code

```
var s = "toto";
System.out.println(s.length());
```

   - The variable s has the `String` type.
   - The compiler know that the variable s have `length()` method because it know that the type of `s` is `String`.

2. The following code :

```
var s1 = "toto";
var s2 = s1;
var s3 = new String(s1);

System.out.println(s1 == s2);
System.out.println(s1 == s3);
```

   displays :

```
user:~/Exo1/bin$ java Main
true
false
```

   - The first comparison return `true` because the `s2` and `s1` contains the reference of `"toto"` string.
   - The second comparison return `false` because the reference of `new String(s1)` is different of the reference `s1`

3. To compares the contains of two strings, we have to use the `equals()` method.

```
var s4 = "toto";
var s5 = new String(s4);

System.out.println(s4.equals(s5));
```

4. The following code

```
var s6 = "toto";
var s7 = "toto";
System.out.println(s6 == s7);
```

display

```
true
```

> The comparison return `true` because the JVM create an object `"toto"` in the heap and assign the reference of this object to `s6` and `s7` variable So both variables contains the reference of `"toto"` object.

5. The `String` has to be immutable to ensure the integrity of the contains of the string. In addition, the `String` object is used in other context of the program.

6. The following code

```
var s8 = "hello";
s8.toUpperCase();
System.out.println(s8);
```

display

```
hello
```

> The `toUpperCase()` methode return a string, so the return has to be assigned to a variable.

Exercice 2 - Morse Stop.

1. Write a code using the `+` operator to concatenate

```java
public class Morse {
    public static void main(String[] args) {
        int len = args.length;
        if(len == 0) {
            System.err.println("Usage: java Morse <arg1> <arg2> ....");
            return;
        }
        var output = "";
        for(String arg: args) {
            output += arg + " Stop. ";
        }
        System.out.println(output);
    }
}
```

2. The `StringBuilder` object allow us to build a string with better performance.
    - The `append(String)` method return a `StringBuilder` object to allow us the chaining of method call like :

```java
  var myString = new
StringBuilder().append("Hello").append("World").toSting();
```

3. Rewrite the previous code using `StringBuilder`

```java
public class Morse {
    public static void main(String[] args) {
        int len = args.length;
        if(len == 0) {
            System.err.println("Usage: java Morse <arg1> <arg2> ....");
            return;
        }
        var output = new StringBuilder();
        for(String arg: args) {
            output.append(arg)
                .append(" Stop. ");
        }
        System.out.println(output);
    }
}
```

Using the `Sringbuilder` is better than the concatenation beacause the `+` operator allocate the momory for each concatenation.

4. We can use `' '` instead of `" "` because if an operand of `+` operateur is a string, the result of the operation is a string.
    - We can deduce that the JVM replace the `+` with a built-in method which allocate once the memory if the operation is on one line.
5. We can deduce that the JVM `+` operator allocate memory each time on the loop.
    - We can use a `StringBuilder` in case of a loop and the `+` operator incase of inline code
    - The usage of `+` operator in the `append()` method is bad beacuse the `+` operator allocate a memory each time there is an operand

Exercice 3 - Regex pattern

1. The `Pattern` class and it method `compile()` allow us to create a regex pattern from a string. The resulting pattern can be used to match a string against the regular expression.
    - The `Macther` class is used to find the match of the pattern in a string.
2. Write a program who match the command line arguments

```java
import java.util.regex.Pattern;

public class Regex {
    public static void main(String[] args) {
        if(args.length == 0) {
            System.out.println("Usage: java Regex <arg1> <arg2> ...");
            return;
        }

        Pattern number = Pattern.compile("\\d+");

        for(String argument: args) {
            if(number.matcher(argument).matches()) {
                System.out.println(argument + " is a number.");
            }
        }
    }
}
```

3. Edit the previous code to macthes the argument who contains at leat one digit

```java
import java.util.regex.Pattern;

public class Regex {
    public static void main(String[] args) {
        if(args.length == 0) {
            System.out.println("Usage: java Regex <arg1> <arg2> ...");
            return;
        }

        for(String argument: args) {
            if(Pattern.matches("\\D*\\d+$", argument)) {
                System.out.print(argument + " ");
            }
        }
    }
}
```

4. Write a method who matches an IPv4 adress

```java
import java.util.regex.Pattern;

public class Regex {
    public static void main(String[] args) {
        ...
        System.out.println("IP? : " + args[0]);
        System.out.println("====================");
        var ip_parts = splitIPAdress(args[0]);
        for (int i = 0; i < ip_parts.length; i++) {
            System.out.println("Part "+ i +" => " + (ip_parts[i] & 0xFF));
        }
    }

    public static byte[] splitIPAdress(String ip) {
        var pattern = Pattern.compile("((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)(\\.
(?!$)|$))");
        var matcher = pattern.matcher(ip);
        boolean isIP = Pattern.matches("^((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)(\\.
(?!$)|$)){4}$", ip);
        var result = new byte[4];
        if(!isIP) {
            throw new IllegalArgumentException("Not a valid IPv4 address");
        }

        int i = 0;
        while (matcher.find()) {
            var part = i < 3
                ? Integer.parseInt(matcher.group().substring(0,
matcher.group().length() - 1))
                : Integer.parseInt(matcher.group());
            result[i] = (byte) part;
            i++;
        }
        return result;
    }
}
```