Lab Assignment 4: Working with the class String

Course: Advanced Programming (EE423)
Institution: IGEE, Boumerdes University

November 4, 2022

1 Objectives

Within this lab, you will learn about the String class, which is a provided by Java. With this class you easily manipulate chains of characters. It has a set of predefined functions that you can use to solve many problems related to stings.

2 Assignment

2.1 Creating strings

```
String name = "Fluffy";
String name = new String("Fluffy");
```

Both give you a reference variable called `name` pointing to the String object `"Fluffy"`. However, they are subtly different! The former says to use the **string pool** normally. The second says "No, JVM. I really don't want you to use the string pool. Please create a new object for me even though it is less efficient."

What is a string pool?

In some production applications, strings can use up 25–40 percent of the memory in the entire program. Java realizes that many strings repeat in the program and solves this issue by reusing common ones. The string pool, also known as the intern pool, is a location in the Java virtual machine (JVM) that collects all these strings. The string pool contains literal values that appear in your program. For example, `"name"` is a literal and therefore goes into the string pool. `myObject.t oString()` is a string but not a literal, so it does not go into the string pool. Strings not in the string pool are garbage collected just like any other object.

2.2 Concatenation

```
System.out.println(1 + 2); // 3
System.out.println("a" + "b" + 3); // ab3
System.out.println(1 + 2 + "c"); // 3c
```

As you can see, if both operands are numeric, + means numeric addition. If either operand is a String, + means concatenation.

```
int three = 3;
String four = "4";
System.out.println(1 + 2 + three + four); //64
```

2.3 Accessing elements

A string is a sequence of characters and Java counts from 0 when indexed. This is how each character in the string `"animals"` is indexed

```
-------------------------
| a | n | i | m | a | l |
-------------------------
| 0 | 1 | 2 | 3 | 4 | 5 |
-------------------------
```

2.4 Basic methods of the class String

- `length()` returns the number of characters in the String.

  ```
  String string = "animals";
  System.out.println(string.length()); // 7
  ```

- `charAt()` lets you query the string to find out what character is at a specific index.

  ```
  String string = "animals";
  System.out.println(string.charAt(0)); // a
  System.out.println(string.charAt(6)); // s
  System.out.println(string.charAt(7)); // throws exception
  ```

- `indexOf()`looks at the characters in the string and finds the first index that matches the desired value.

  ```
  String string = "animals";
  System.out.println(string.indexOf('a')); // 0
  System.out.println(string.indexOf("al")); // 4
  System.out.println(string.indexOf('a', 4)); // 4
  System.out.println(string.indexOf("al", 5)); // -1
  ```

- `substring()` looks for characters in a string. It returns parts of the string. The first parameter is the index to start with for the returned string. As usual, this is a zero-based index. There is an optional second parameter, which is the end index you want to stop at.

```
String str = "animals";
System.out.println(str.substring(3)); // mals
System.out.println(str.substring(str.indexOf('m'))); // mals
System.out.println(str.substring(3, 4)); // m
System.out.println(str.substring(3, 7)); // mals
```

- `equals()` method checks whether two String objects contain exactly the same characters in the same order. The equalsIgnoreCase() method checks whether two String objects contain the same characters with the exception that it will convert the characters' case if needed.

```
System.out.println("abc".equals("ABC")); // false
System.out.println("ABC".equals("ABC")); // true
System.out.println("abc".equalsIgnoreCase("ABC")); // true
```

- `startsWith()` and `endsWith()`

```
System.out.println("abc".startsWith("a")); // true
System.out.println("abc".startsWith("A")); // false
System.out.println("abc".endsWith("c")); // true
System.out.println("abc".endsWith("a")); // false
```

- `contains()` looks for matches in the String.

```
System.out.println("abc".contains("b")); // true
System.out.println("abc".contains("B")); // false
```

- `replace()` method does a simple search and replace on the string.

```
System.out.println("abcabc".replace('a', 'A')); // AbcAbc
System.out.println("abcabc".replace("a", "A")); // AbcAbc
```

- `trim()` removes whitespace from the beginning and end of a String.

```
System.out.println("abc".trim()); // abc
System.out.println("\t a b c\n".trim()); // a b c
```

- `compareTo()` compares two strings using the alphabetical order

```
"a".compareTo("b") // -1
"a".compareTo("a") // 0
"b".compareTo("a") // 1
"bob".compareTo("alex") < 0 // false
"bob".compareTo("alex") > 0 // true
```

2.5 Method Chaining

```
String start = "AniMaL ";
String trimmed = start.trim(); // "AniMaL"
String lowercase = trimmed.toLowerCase(); // "animal"
String result = lowercase.replace('a', 'A'); // "Animal"
System.out.println(result);
```

The previous code is equivalent to this:

```
String result = "AniMaL ".trim().toLowerCase().replace('a', 'A');
System.out.println(result);
```

2.6 Testing for equality of strings using equals()

You have to be very careful when testing for equality of strings in Java!

```
String x = "Hello World";
String y = "Hello World";
System.out.println(x == y); // true
```

The JVM created only one literal in memory. `x` and `y` both point to the same location in memory; therefore, the statement outputs `true`. It gets even trickier. Consider this code:

```
String x = "Hello World";
String z = " Hello World".trim();
System.out.println(x == z); // false

String x = new String("Hello World");
String y = "Hello World";
System.out.println(x == y); // false

String x = "Hello World";
String z = " Hello World".trim();
System.out.println(x.equals(z)); // true
```

The lesson is to never use `==` to compare String objects and use `equals()` instead.

2.7 Exercise 1: Palindrome

Write a function `public String findPalindrome(String s)` that returns the longest palindrome in a string. You can use the function `s.toLowerCase()` to convert the string to lower case.

- Return `"The string is too short"` if the string is less than 3 characters.

- Return `"No palindrome found"` if there is no palindrome of at least 3 characters,
- Return `"Only alphabetical characters are allowed"` in case there is a different character than space or alphabetical characters.
- In case there are many possible palindromes with the highest length, return the leftmost one.

```
findPalindrome("ok")  // The string is too short
findPalindrome("ok!") // Only alphabetical characters are allowed
findPalindrome("coucou") // No palindrome found
findPalindrome("good morning Madam") // Madam
findPalindrome("nurses run"); // nurses run (notice the spaces)
findPalindrome("madam good morning Madam") // madam
findPalindrome("room llevell   morning refer") // om llevell   mo
```

2.8 Exercise 2: Sorting

Write a function `String [] sortArray(String [] a)` that takes as input an array of strings and sorts it. The passed array must remain unchanged. The returned array must be sorted.

2.9 Exercise 3: Longest chain

Write a function `int[] longestIncrementalChain(String s, int n)` that finds the longest incremental chain of numbers. You can increase any number using n until n becomes 0. For instance,

```
longestIncrementalChain("5,0,1,7", 0);         // (0,1,7)
longestIncrementalChain("5,0,1,7", 9);         // (5,0+5,1+4,7)
longestIncrementalChain("10,1,9,4,10,0,6,7",0);// (1,4,6,7)
longestIncrementalChain("8,7,6,5,5",0);        // (5,5)
longestIncrementalChain("8,7,6,5,5",5);        // (7,6+1,5+2,5+2)
```

- use `split(",")` to get an array of numbers
- use `int i = Integer.parseInt("-56");` to parse a string into an int

3 Conclusion

After finishing this assignment, you should now know how to use the String class and most of its predefined functions.