Programming 1

Week 08– `String` Class

The `String` Class

- Java provides 8 primitive data types. Java has no primitive data type that holds a series of characters.
- They are called “primitive” because they are not created from classes.
- The `String` class from the Java standard library is used for this purpose.
- In order to be useful, a variable must be created to reference a `String` object.

```
String name;
```

  - Notice the `S` in `String` is upper case.
  - By convention, class names should always begin with an uppercase character.

Primitive versus Reference Variables (1 of 2)

- Primitive variables actually contain the value that they have been assigned.

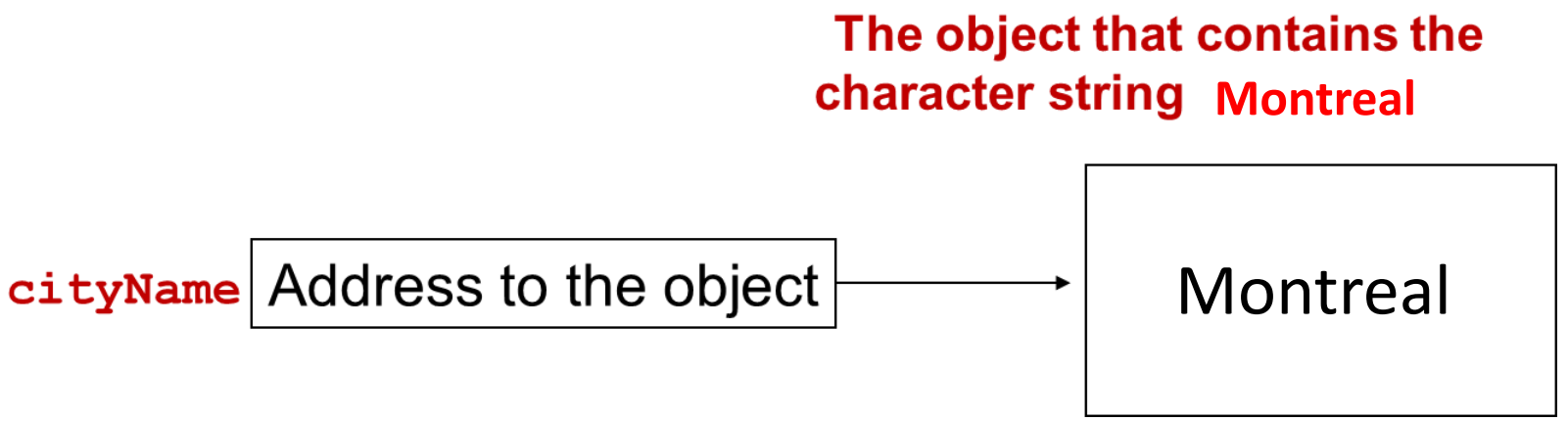
```
number = 25;
```

- The value 25 will be stored in the memory location associated with the variable `number`.
- Objects are not stored in variables, however, objects are **referenced** by variables.

Primitive versus Reference Variables (2 of 2)

- When a variable references an object, it contains the memory address of the object's location.
- Then it is said that the variable **references** the object.

```
String cityName = "Montreal";
```



`String` Objects

- A variable can be assigned a String literal.

```
String value = "Hello";
```

- `Strings` are the only objects that can be created in this way.
- A variable can be created using the **new** keyword.

```
String value = new String("Hello");
```

- This is the method that all other objects must use when they are created.

`String` Methods

- The `String` class contains many methods that help with the manipulation of `String` objects.
- Many of the methods of a `String` object can create new versions of the object.
- To call a method means to execute it. The general form of a method call is as follows:

```
referenceVariable.method(arguments. . .)
```

- `referenceVariable` is the name of a variable that references an object,
- method is the name of a method,
- and `arguments`. . . is zero or more arguments that are passed to the method. If no arguments are passed to the method, a set of empty parentheses must follow the name of the method

length()Method

- One of those methods is the `length` method.

```
stringSize = value.length();
```

- `String` method `length` determines the number of characters in a string including spaces.
- The `String` class's length method returns an int value. This means that the method sends an int value back to the statement that called it. This value can be stored in a variable, displayed on the screen, or used in calculations.

```
•“Java II".length()       equals 7
•"3.14".length()          equals 4
•" ".length()             equals 1
•"".length()              equals 0
```

String Index

- Each character inside a string has an index.
- The index of the first character is 0.
- In the example, the length of the string "Sunny Day" is 9, that means there are 9 characters inside of the string. The index is in range [0, 8].

- In general, if a string has a length of N, the index is in the range [0, N - 1]. Or we say, [0, length() - 1].

String	"Sunny Day"								
Character in the string	'S'	'u'	'n'	'n'	'y'	' '	'D'	'a'	'y'
Position of the character in the string	0	1	2	3	4	5	6	7	8

charAt() Method

- By giving an index, we can know what is stored at that position in a string. `String` method `charAt` returns the character at a specific position in the `String`.

- We use .charAt() method to extract a character from the string.

```
String str = "Today is a good day";
System.out.println("The first character is \"" + str.charAt(0) + "\"");
```

- charAt() will return a char value. You cannot give a double or negative value as an index
- The index of the last element is .length() – 1, instead of .length()

```
String str = "Today is a good day";
System.out.println(str.charAt(str.length()));// Error, index out of bounds
System.out.println(str.charAt(str.length() - 1));// Correct, prints 'y'
```

Substring() Method (1 of 4)

- The `String` class provides methods to extract substrings in a `String` object.
  - The `substring` method returns a substring beginning at a start location and an optional ending location.

```
String fullName = "Maria Susan Smith";
String lastName = fullName.substring(14);
System.out.println("The full name is "
                   + fullName);
System.out.println("The last name is "
                   + lastName);
```

`Substring()` Method (2 of 4)

- There are two ways to extract a substring from a string:
  - `str.substring(startIdx, endIdx)`
  - `str.substring(startIdx)`
- `str.substring(startIdx, endIdx)` needs two indexes, the starting index and the ending index.
- The starting position is included, while the ending position is excluded, i.e.: [startIdx, endIdx).
- "Today is a good day".substring(0, 5) will return you "Today". The index of 'y' is 4, we put 5 because we want to include it in the substring.

Substring() Method (3 of 4)

```
String str = "Today is a good day";
```

- If you want to extract a substring "is a good day", the substring starts at 6, and end at ".length() – 1". However, the ".length() – 1" is not included, if you want to include it, you should use the range .substring(6, .length()).

- If you only give one index in the .substring(startIdx), that means extract a substring, starting from that index, until the end of the string.

```
String str = "Today is a good day";
String subStr = str.substring(6);
// same as str.substring(6, str.length()), but much easier
System.out.println("The length of the substring is " + subStr.length());
```

Substring() Method (4 of 4)

- Since .substring() returns a string, you can chain it with other methods in the String class.

```
String str = "Today is a good day";
System.out.println("The length of the substring is "
                   + str.substring(0, 5).length());
```

```
String str = "Today is a good day";
System.out.println("The first letter of the second substring is "
                   + str.substring(6, 8).charAt(0));
```

`indexOf( )&& lastIndexOf( ) Methods`

- The `String` class also provides methods that will locate the position of a substring.
  - `indexOf()`
    - returns the first location of a substring or character in the calling `String` Object.
  - `lastIndexOf()`
    - returns the last location of a substring or character in the calling `String` Object.

- Method `indexOf` locates the first occurrence of a character in a `String`. If the method finds the character, it returns the character's index in the `String`—otherwise, it returns `-1`.
- A second version of `indexOf` takes two integer arguments—the character and the starting index at which the search of the `String` should begin.

`indexOf( )Method`

- We can extract a string by using the substring(), this is because the startIdx and the endIdx for each part is always fixed.

- But it is not always the case, let's say you want the user to input his email address, and then extract the account and the host address.

Example: nagatd@vaniercollege.qc.ca

- The account is the string before the '@', while the host address is the string after the '@'.
- But the idx of '@' can be almost everywhere. We don't know how long is the account.
- So to extract the account, we need to first find the index of the '@'.
- The method "`.indexOf()`" in the String class can help us to find it.
- If there are more than one match, .indexOf() will return the index of the first match.
- If there is no match, .indexOf() will return -1.

indexOf( )Method

- For the Email example:

```
System.out.print("Please enter your email address: ");
String email = console.nextLine();
int atIdx = email.indexOf('@');
String account = email.substring(0, atIdx);
String host = email.substring(atIdx + 1);
System.out.printf("Account: %s.\n", account);
System.out.printf("Host: %s.\n", host);
```

Note that:

- Mehtod .charAt() receives an idx, and returns the character at that position.
- Method .indexOf() is the opposite, which receives a character or a String, and return its first position in the string.

lastIndexOf() Method

- "lastIndexOf()" behaves very similar to "indexOf()", the only difference is that it returns the last index of a specific character.

```
String str = "laval";

System.out.print(str.indexOf('l'));          // 0
System.out.print(str.lastIndexOf('l'));      // 4
```

contains( )Method

- "indexOf()" and "lastIndexOf()" will return an int value to indicate where is the specific character.
- But sometimes we only want to know if a string contains a specific character or not, for example, I want to check if an email address contains a '@' or not, if no, that means it is not valid. But I don't care what is the index of that '@'
- The contains method returns true if the calling string object contains a particular substring.

```
String str = "nagatd@vaniercollege.qc.ca";
      System.out.println("it has @!"+
      str.contains("@"));
```

`startsWith()` Method

- The `startsWith` method determines whether a string begins with a specified substring.

```
String str = "Four score and seven years ago";
```

- `str.startsWith("Four")` returns true because `str` does begin with “Four”.
- `startsWith` is a case sensitive comparison.

endsWith()Method

- The `endsWith` method determines whether a string ends with a specified substring.

```
String str = "Four score and seven years ago";
```

- The `endsWith` method also performs a case sensitive comparison.

Hands on:

- Ask the user to input a string whose length is odd, e.g.: "hello", "email", etc.
- Find the idx of the center letter, and use indexOf() to extract it into a variable, e.g.: 'a' for "email"
- Separate the string into two parts evenly and stored them into two variables, e.g.: "em" and "il" for "email"
- Switch the order of the two parts and create a new String: e.g.: "ilaem" and store it into a variable.
- Use printf() to print the result.

```
run:
Please enter a string even odd number of characters: email

First Part: em
Center Letter: a
Second Part: il
After Switching: ilaem
BUILD SUCCESSFUL (total time: 0 seconds)
```

toLowerCase() and toUpperCase()

- There are two very simple methods in the String class we also use very frequently:
  - toLowerCase()
  - toUpperCase()
- They will switch the entire all the letters in a string into Lower case or Upper case.
- If the String contains symbols, or numbers or anything that is not a letter, these two methods keep it as the way it is.

```
String str = "I love JAVA! Wait, do I?";
System.out.println(str.toLowerCase());  // "i love java! wait, do i?"
System.out.println(str.toUpperCase());  // "I LOVE JAVA! WAIT, DO I?"
```

Comparing `String` Objects

- In most cases, you cannot use the relational operators to compare two `String` objects.
- Reference variables contain the address of the object they represent.
- Unless the references point to the same object, the relational operators will not return true.
- When primitive-type values are compared with ==, the result is `true` if both values are identical.
- When references are compared with ==, the result is `true` if both references refer to the same object in memory.

Comparing `String` Objects

- Method `equals` tests any two objects for equality
  - The method returns `true` if the contents of the objects are equal, and `false` otherwise.
- Java treats all string literal objects with the same contents as one `String` object to which there can be many references.
- `String` method `equalsIgnoreCase` ignores whether the letters in each `String` are uppercase or lowercase when performing the comparison.

`String` Method `valueOf()`

Class `String` provides `static` valueOf methods that take an argument of any type and convert it to a `String` object.

`Strings` are Immutable Objects

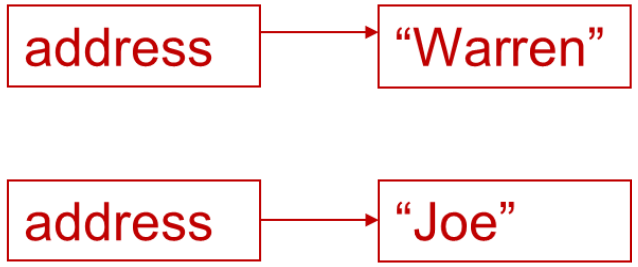
- `Strings` are immutable objects, which means that they cannot be changed. When the line

```
str = "Joe";
```

is executed, it cannot change an immutable object, so creates a new object.

The `name` variable holds the address of a `String` object

The `str` variable holds the address of a different `String` object



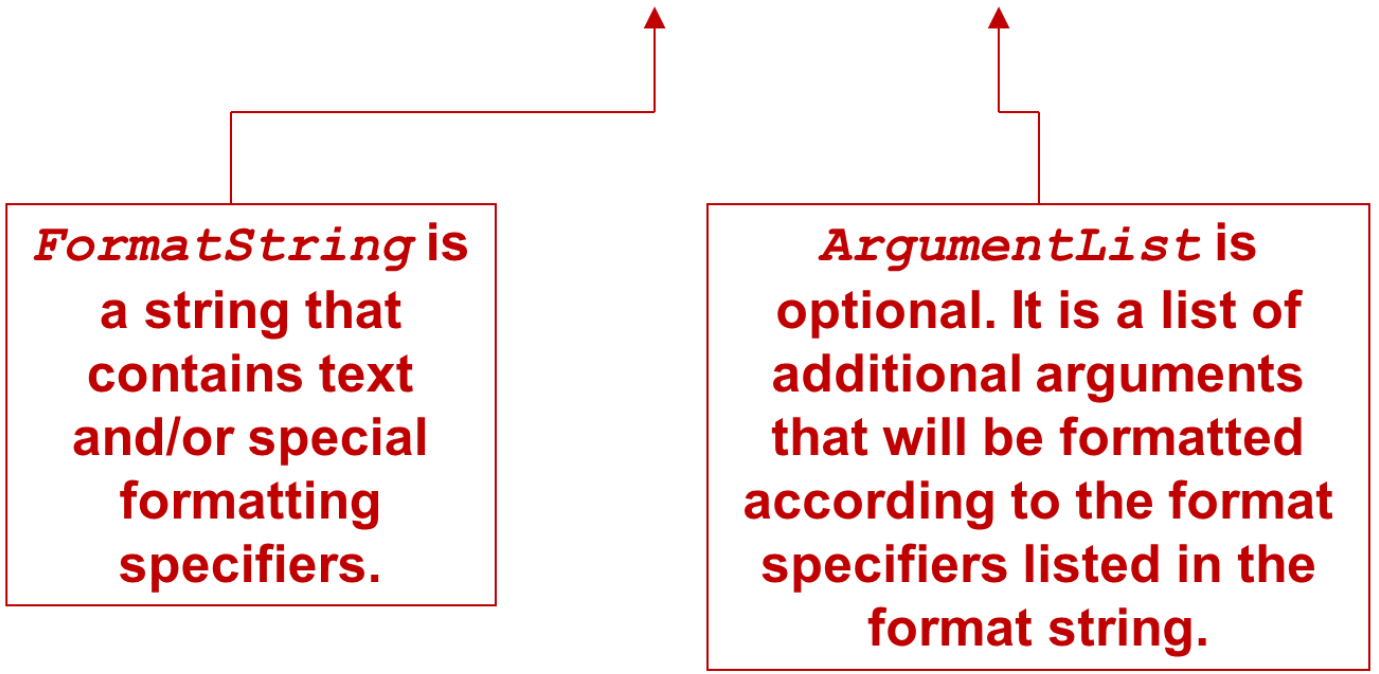
The `String.format` Method (1 of 3)

- The `String.format` method works exactly like the `System.out.printf` method, except that it does not display the formatted string on the screen.
- Instead, it returns a reference to the formatted string.
- You can assign the reference to a variable, and then use it later.
- Before we have: "hello".length(), "hello".indexOf('h'), "hello".charAt(0), where we always have a real string (object) before the "."
- But String.format() has the class "String" before the ".".
  - This is because String.format() is a static method, which will be cover in the future.

The `String.format` Method (2 of 3)

- The general format of the method is:

```
String.format(FormatString,ArgumentList);
```



The `String.format` Method (3 of 3)

- `String.format()` is widely used in OOP, we will see it soon.

```
String fname = "John";
String lname = "Smith";
int age = 34;

// String.format() will generate a string for you, without printing it.
String str = String.format("My name is %s %s, and I am %d years old",
fname, lname, age);
```

Try it yourself

Write a piece of code to:

- Cut the sentence: "How are you?" into 3 strings: str1: "How", str2: "are", str3: "you".
- Use printf() to join the three substrings and print "How are you".