**String class and its methods in Java**

java.lang.String is the most used class in Java. It’s important to understand and able to manipulate strings fluently.

* String Class
* String Methods
* Comparing Strings

**String Class**

The String is a built-in class in Java to store a sequence of characters between double-quotes. It’s under java.lang package so we don’t have to import, it will be imported automatically for every class.

public class Main {  
 public static void main(String[] args) {  
 String str = "apple";  
 System.*out*.println(str); // apple  
   
 str = "banana";  
 System.*out*.println(str); // banana  
  
 String name = new String("John");  
 System.*out*.println(name); // John  
 }  
}

* We can initialize string directly using = operator or we can create using new keyword. The first version will use StringPool memory and the second option will create an object in the heap memory.

public class Main {  
 public static void main(String[] args) {  
 String str = "apple";  
 System.*out*.println(str); // apple  
  
 str.toUpperCase(); // original value will not get changed  
 System.*out*.println(str); // apple  
   
 str = str.toUpperCase(); // reassign new value  
 System.*out*.println(str); // APPLE  
 }  
}

* The string is an immutable class. This is the most important definition to remember. Immutable means that once created with some value we cannot change it. Just to be clear we can reassign the value of reference (as shown in the above example with str) but the original object that was created in the beginning will never get changed.
* The string is thread-safe. All immutable classes are thread-safe because they are not modifiable.
* If you are reassigning your string reference a great number of times in some infinite loop. Consider using StringBuilder instead and convert to String after the loop. Because every time you reassign your string you are creating a new object in the memory.
* The string is the final class and we cannot extend the final classes.

**String Methods**

The methods provided by the String class can be used to manipulate its value. Most of the String methods return a new value and never change the actual value of the string.

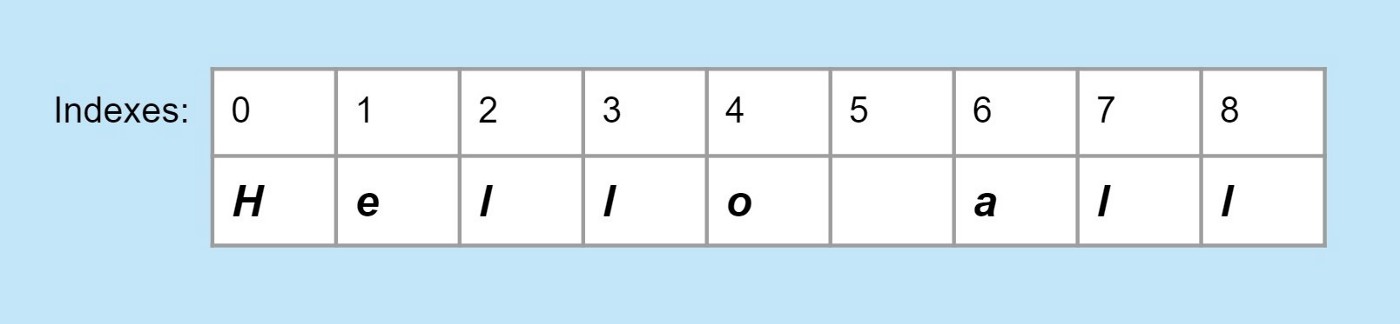
***toUpperCase()*** and ***toLowerCase()***

String str = "apple";  
str = str.toUpperCase();   
// it will return new string with all upper cases  
System.out.println(str); // APPLEString name = "John Doe";  
name = name.toLowerCase();  
// it will return new string with all lower cases  
System.out.println(name); // john doe

***length()*** method returns the number of characters in the string

String str = "apple";  
   
// .length() method returns number of characters in the string int strLength = str.length();   
System.out.println(strLength); // 5  
   
str = "car "; // space is also the character System.out.println(str.length()); // 4

String indexes



each char in the string has a position — index

String internally based on an array of char. Each character including empty space has a special position — index. The index begins with 0. Some of the methods use indexes of the string.

***substring(start)*** and ***substring(start, end)***

// 012345678910   
String java = "Java is fun";   
// Using substring we can get part of the string based on indexesString javaFun = java.substring(8);   
System.out.println(javaFun); // fun  
System.out.println(java); // Java is fun// 0123456789   
String str = "SoftSkills";   
str = str.substring(4);   
System.out.println(str); // Skills  
   
str = "my workspace";   
str = str.substring(7);   
System.out.println(str); // space  
   
// 0123456789   
String str1 = "SoftSkills";   
str1 = str1.substring(0, 4);   
System.out.println(str1); // Soft  
   
String str2 = "Java is Cool!";   
str2 = str2.substring(5, 7);   
System.out.println(str2); // is  
   
// 012345678   
String str3 = "workspace";   
str3 = str3.substring(0, 4);   
System.out.println(str3); // work

***indexOf*** is used to get an index of a certain string or char

// 01234567   
String str = "applpepp";   
int index = str.indexOf("e");   
System.out.println(index); // 4

System.out.println(str.indexOf("l")); // 3 System.out.println(str.indexOf("p")); // 1 System.out.println(str.lastIndexOf("p")); // 7 looking from back

***charAt(index)*** is used to get specific char by index

// 0123   
String str = "home";   
char ch = str.charAt(1);   
System.out.println(ch); // o   
   
// 012345   
str = "Public";   
char ch2 = str.charAt(5);   
System.out.println(ch2); // c  
   
// 0123456   
str = "Pricing";   
System.out.println(str.charAt(6)); // g

***replace(oldChar, newChar)***is used to replace old char with new char in the string and it will return modified string.

String city = "Alma-Ata";   
System.out.println(city); // Alma-Ata   
city = city.replace("-", " ");   
System.out.println(city); // Alma Ata

String word = "Hello, World!";   
word = word.replace("Hello", "Aloha");   
System.out.println(word); // Aloha, World!

***split(str)***we can split our string and get string array as parts.

String str = "My name is John Doe";  
String[] parts = str.split(" ");  
for (String part : parts) {  
 System.out.println(part);  
}

Output:

My  
name  
is  
John  
Doe

***contains(str)***it checks if one string contains another one. The return type is boolean.

String topic = "Variables, assignment operator, arithmetic operator, input from terminal";  
   
boolean isThere = topic.contains("input");   
System.out.println(isThere); // trueSystem.out.println(topic.contains("java")); // false System.out.println(topic.contains("terminal")); // true

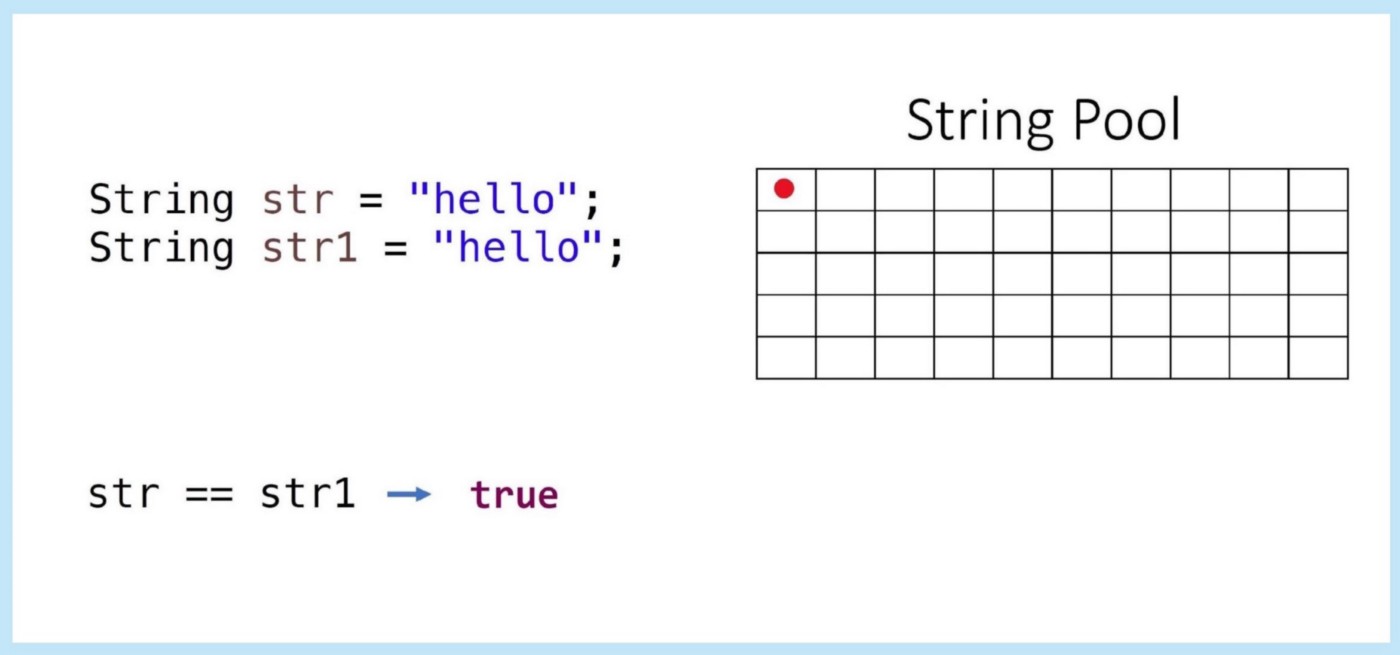
**Comparing Strings**

To compare two strings on equality we need to use equals method. Never use == operator to compare strings. It looks like it == operator comparing actual content because of the string pool, but it does check if two references are pointing to the object or not.

The string pool is a memory that String uses to reuse the same data. When we create a String as String name = "Alex"; it will get stored in the string pool. There is a mechanism that checks before creating a new string if the sting pool already has the same value and if yes it will reuse it. It’s safe to do so because String is immutable.

String str = "hello";  
String str1 = "hello";

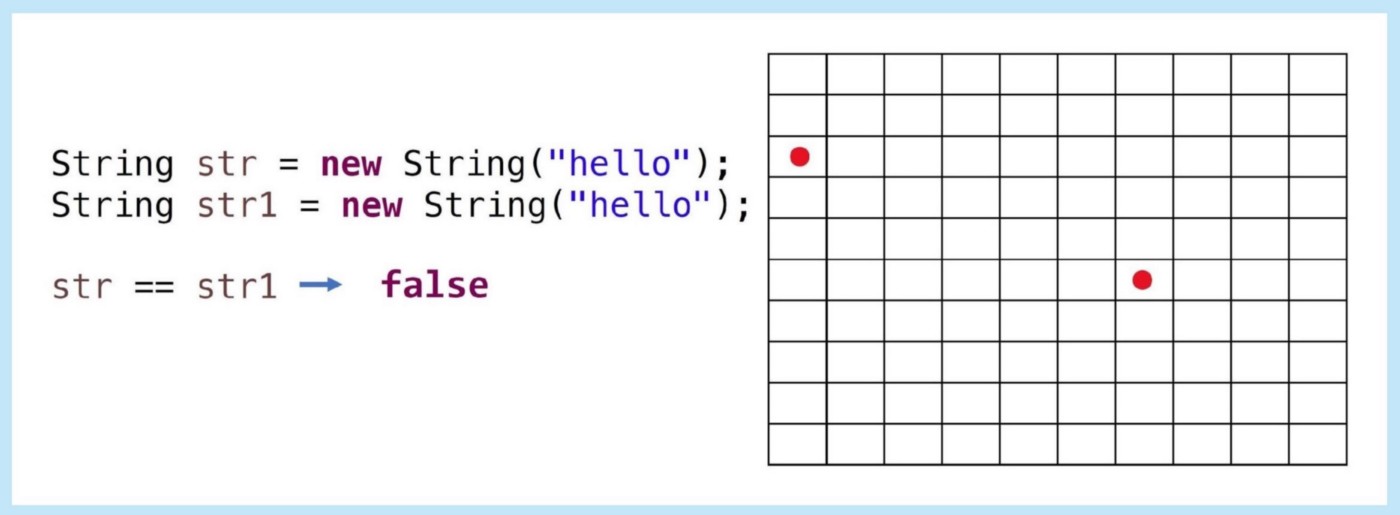
System.out.println(str == str1); // true  
System.out.println(str.equals(str1)); // true



* When we created str there are no string objects at all in the string pool so new string object with "hello" the value will get created.
* When we created str1 there is one string object with the same value in the string pool already. It will just reuse the same object and made str1 point to it. That’s a reason == returns true — both references are pointing to the same object.
* equals() compare the actual value of the string.

String str = new String("hello");  
String str1 = new String("hello");

System.out.println(str == str1); // false  
System.out.println(str.equals(str1)); // true



* if we create a string with new keyword, it will not use a string pool and two separate objects got created. That’s why we have false when using == operator.
* equals() compare the actual value of the string.