String Methods

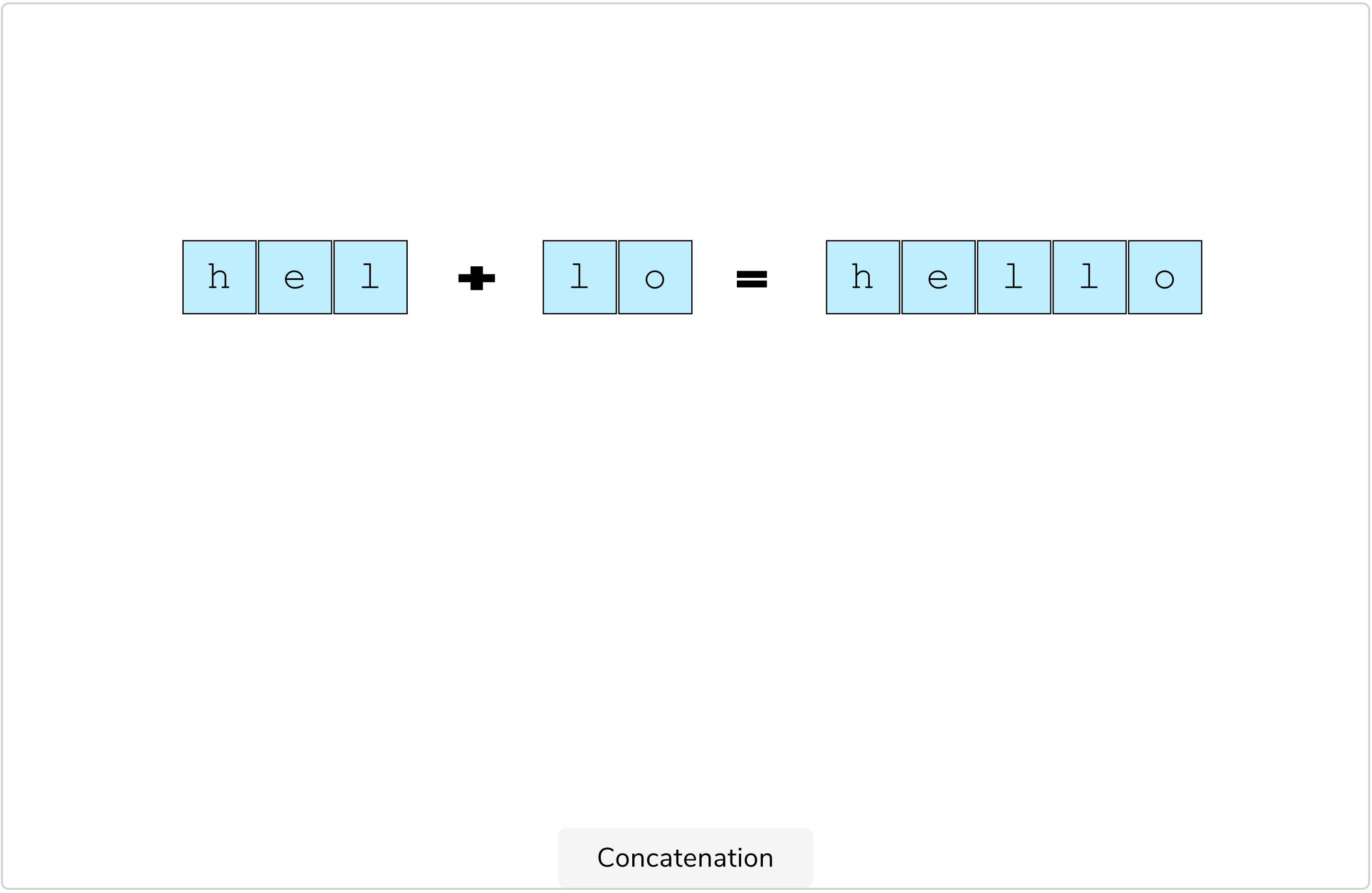
In this lesson, we will see the functionality of inbuilt methods in String Class.

We'll cover the following

Concatenation

Java provides special support for the concatenation of multiple *Strings*. **Concatenation** is referred to as the joining of two or more Strings. This is done by the use of the `+` operator. The code below shows an example of both.

> **Did you know?** The interesting thing is that the `+` operator can be used to **not only** join a String with other Strings but also join Strings with **other types** of objects.



```
class concat {
    public static void main(String[] args) {
        String one = "Hello";
        String two = " World";
        int number = 10;

        // concatenating two strings
        System.out.println(one + two);

        //concatenating a number and string
        System.out.println(one + " " + number);

        //saving concatenated string and printing
        String new_string = one + two + " " + number;
        System.out.println(new_string);
    }
}
```

> **Note:** Keep in mind that using the + operator will first convert the number or other objects to String type and then do the concatenation!

Comparing strings

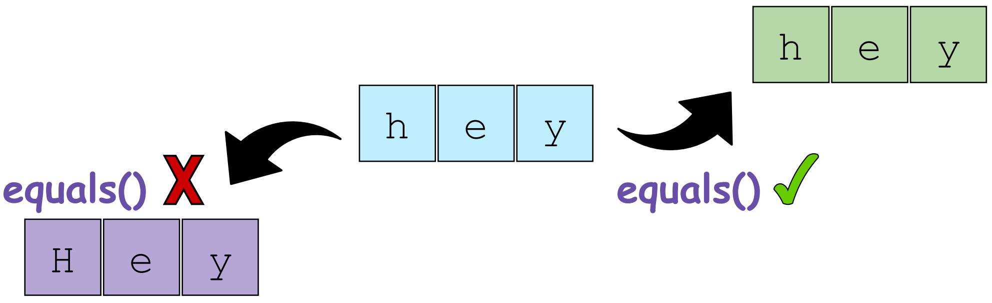
The **String** class has an in-built function called **equals()** for this operation. The method returns `true` if the two Strings are identical and `false` if they aren't. The function is case-sensitive, as can be seen in the code snippet below.

```
class concat {
    public static void main(String[] args) {
        String one = "Hello";
        String two = "World";
        String lower = "hello";
        String same = "Hello";

        System.out.println(one.equals(two));

        System.out.println(one.equals(lower));

        System.out.println(one.equals(same));
    }
}
```

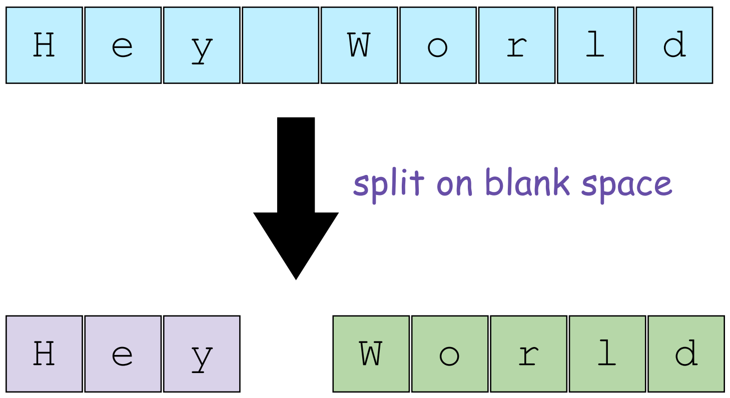


Comparing strings

Splitting a string

This allows the programmer to split the **String** on the basis of a *regular expression*.

This, in simple terms, means that the *String* will be split on a particular pattern that we can give to the **split()** function built into the String class. The function will return a **String array** with elements separated on the basis of the *expression* given. Let's see how it works in the snippet below.

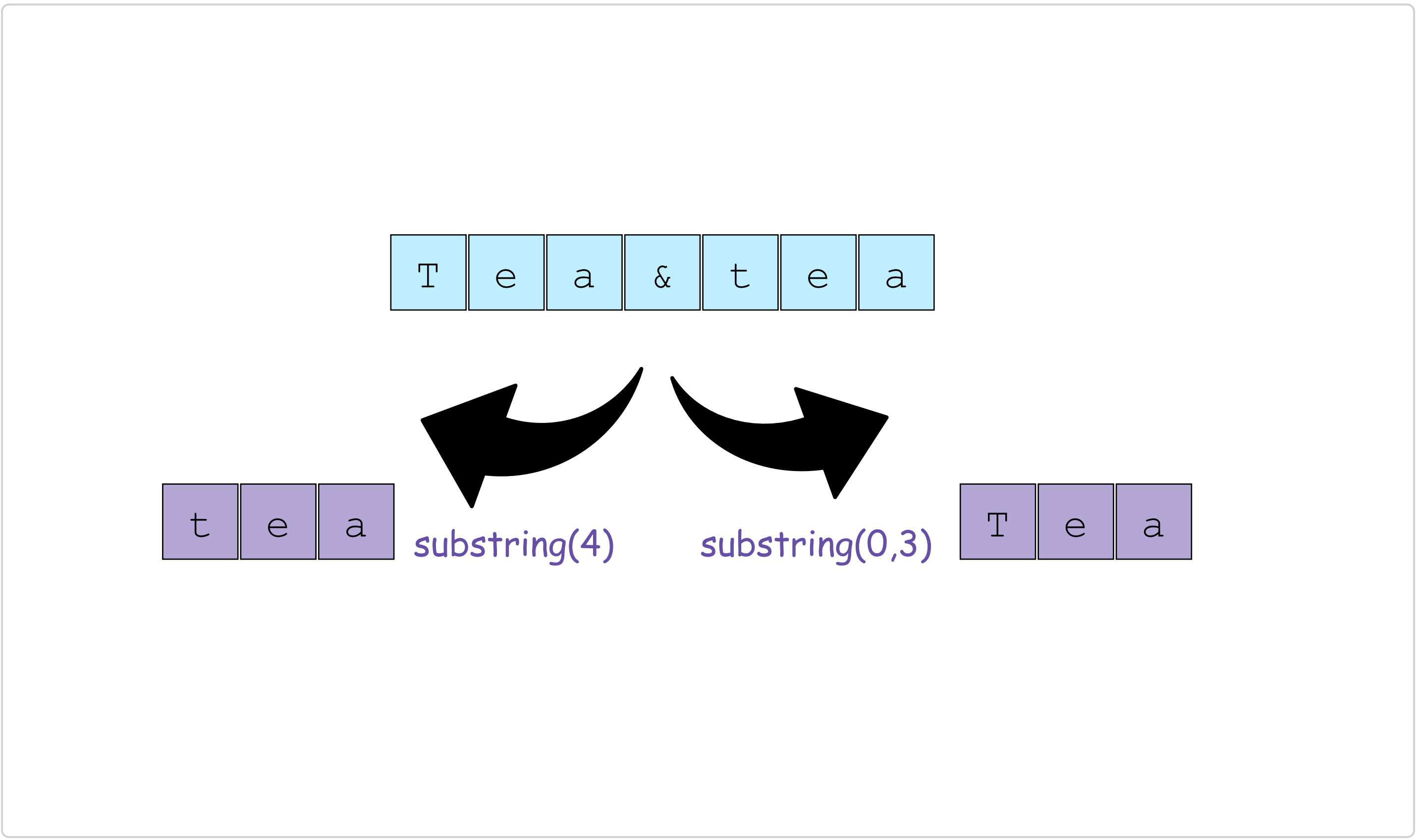


Splitting a String

```
class split_string {
    public static void main(String[] args) {
        String greet = "Hello World,My name is Waldo,How are you?";
        String[] greetings = greet.split(",");
        System.out.println(greetings[0]);
        System.out.println(greetings[1]);
        System.out.println(greetings[2]);
    }
}
```

Substrings

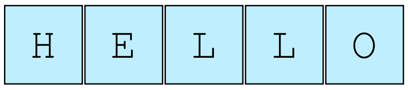
This method allows the programmer to extract **substrings** from given Strings, i.e., you can take out a part of an existing String as a new String. The method that allows this functionality is called **substring**, and it works in two ways. The code snippet below shows both ways of using this method.



```
class substring_ {
    public static void main(String[] args) {
        String choice = "CoffeeOrTea";
        //First: Only one argument
        System.out.println(choice.substring(8));

        //Second: Two arguments
        System.out.println(choice.substring(0, 6));
    }
}
```

Understanding the two implementations



The image above shows how a String is stored with each letter in a box. The counting of the boxes starts with `0`. Hence the letter `H` will be at **index 0** of the string. Now that this is understood let's understand the implementation of the two methods.

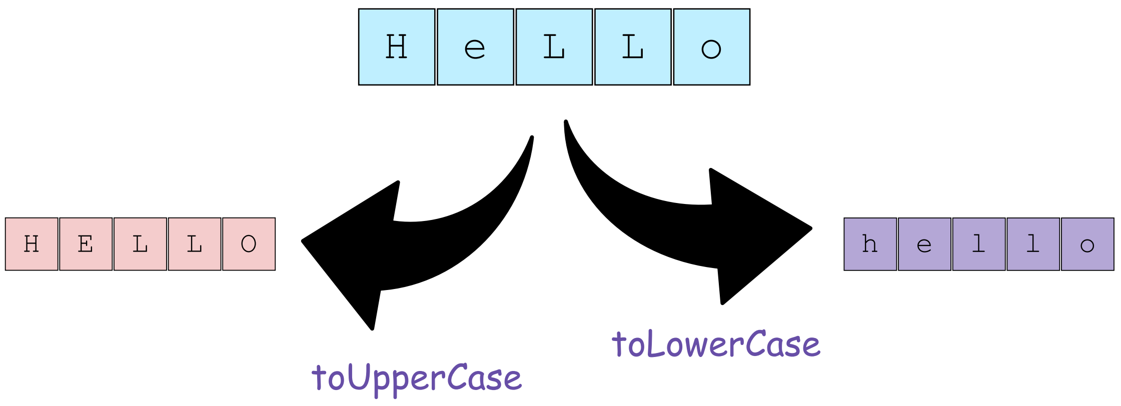
One Argument: This method takes only one input for the *substring()* function. This argument given signifies the index at which the *extraction of substring* should begin. Hence, in the snippet above, the index given is `8`, and so it starts extraction from the `index 8` till the **end** of the String.

Two Arguments: This method takes in two input arguments, one for the index at which the *substring extraction* would begin and the second for where the substring should *end*.

> **Note:** The end index is not *inclusive*, and hence the last character in the substring will be from the *end index given -1* of the original String.

String cases

There are two in-built functions that take a String in its input and return the new String that contains all the Upper or Lower case characters. The code snippet below shows how to use the methods, **toUpperCase()** and **toLowerCase()**, respectively.



String Cases

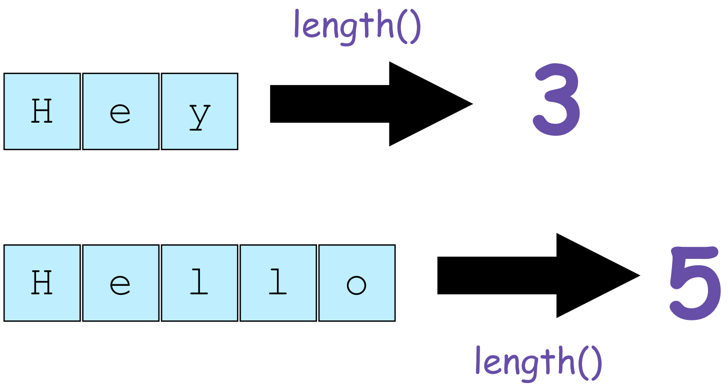
```
class split_string {
    public static void main(String[] args) {
        String greet = "HeLlo WoRld";

        //Returns new string in which all characters are converted to upper case
        System.out.println(greet.toUpperCase());
        //Returns new string in which all characters are converted to lower case
        System.out.println(greet.toLowerCase());


    }
}
```

Length of a string

There is an in-built method in Java that returns the total length of a String. This will include any *white spaces* within the String as well. The method is called **length()**. The code snippet below shows how to use this method.



Length of a String

```
class length_of_String {
    public static void main(String[] args) {
        String greeting = "Hello World";
        System.out.println("The length of greeting is: " + greeting.length());
    }
}
```

In the next lesson, we will solve a challenge related to strings.