07/05/23

String

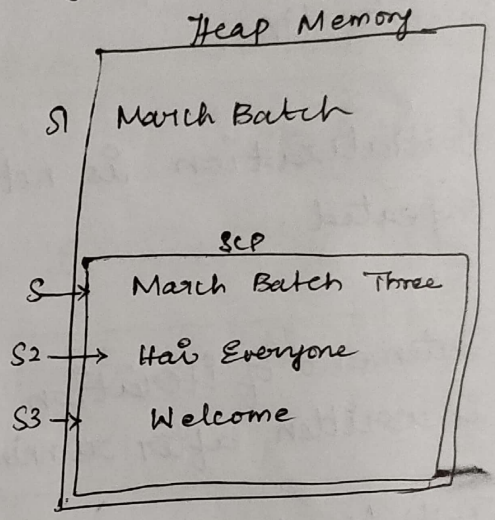
* Non primitive datatype
* Works based on index, starts from 0 to n-1
* Avail in java.lang. package.

Two ways of ~~creating~~ declaring String

* Literal ⟶ String s = "March Batch Three";
* Immutable ⟶ String s1 = new String ("March Batch");

SCP ⇒ String Constant Pool.

* Literal way of creating String is stored in SCP
* Immutable way of creating string is stored in Heap Memory.

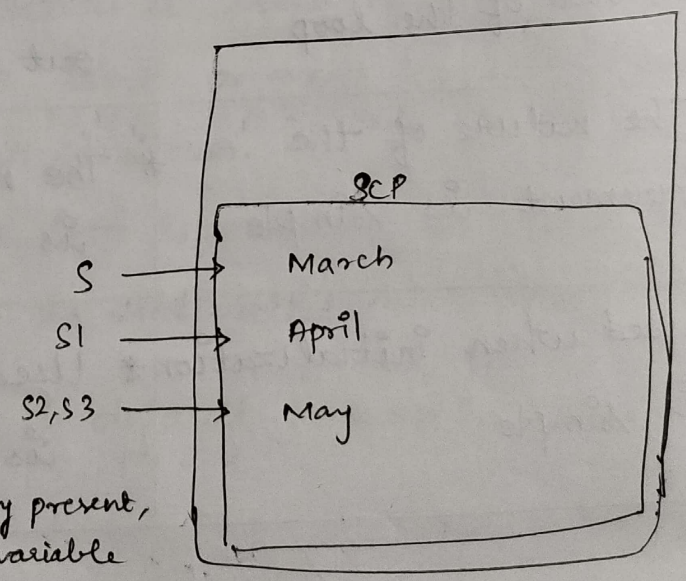


~~hash~~

hashcode() is a method of String

It returns the memory location of String where it is stored.

```
String s = "March";
String s1 = "April";
String s2 = "May";
String s3 = "May";
```



~~Hash~~

Hashcode search for the SCP, whether if the same text is already present, then it just declare the variable there, without creating new memory for same text.

String is immutable

If the user wants to make the change, then it is not replaced and it is stored in the new memory location

String can be ~~changed~~ ~~mutable~~ Mutable, by using String Buffer and Builder

Immutable → Mutable by using apperd() ⇒ used to join String

String Builder – ASynchronized,

String Buffer – Synchronized, Thread-Safe

STRING:-

* String is a non primitive datatype (act as a class and also a datatype)
* String is a sequence of Characters. But in java, string is an object that represents a sequence of characters.
* String enclosed with double quotes and works on the principal of index basis. (Starts from 0 to n-1).
* String is class and it is in java.lang package. Additionally most of predefined methods and classes in java are from java.lang hence explicit import does not happen
* The java.lang.String class is used to create a String object.

* String values are stored in String Constant pool (SCP). SCP is a type of memory which is shared by heap memory.

* String is literal and immutable.

There are two ways to create String Object :-

* String literal
* By new keyword.

STRING LITERAL :-

Java string literal is created by using double quotes

EXAMPLE :-

```
String s = "Welcome";
```

Each time you create a string literal, JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned.

If the String doesn't exist in the pool, a new string instance is created and placed in the pool.

EXAMPLE :-

```
String s1 = "Welcome";
String s2 = "Welcome"; //It doesn't create new instance.
```

BY NEW KEYWORD :-

```
String s = new String("Welcome");
```

* The string can also be declared using new operator i.e. dynamically allocated.

* In case of String are dynamically allocated they are assigned a new memory location in heap. This String will not be added to String Constant Pool.

EXAMPLE PROGRAM FOR STRING LITERAL :-

```
package org.String.methods;
public class StringMethods {
  Public static void main (String[] args) {
      String s = "Welcome";
      String s1 = "welcome";
      String s2 = "java";
      String s3 = "java";
      System.out.println(s.hashcode());
      System.out.println(s1.hashCode());
      System.out.println(s2.hashCode());
      System.out.println(s3.hashCode());
    }
}
```

STRING METHODS :-

toUpperCase()
toLowerCase()
StartsWith()
endsWith()
indexOf()
lastIndexOf()
equals()
equalsIgnoreCase()
length()
Contains()
Concat()
charAt()
toCharArray()
replace()
isEmpty()
IsBlank()
Split()
trim()
SubString

```
package org.String;
public class StringConcepts {
    public static void main (String[] args ) {
        String s = "march Batch Three";
        String s1 = "Students";
        String s2 = "   ";
        int length = s.length();
        System.out.println(length);

        String upper = s.toUpperCase();
        System.out.println(upper);
        String lower = s1.toLowerCase();
        System.out.println( lower);
        System.out.println(s.startsWith("m"));
        System.out.println(s.endsWith("e"));
        System.out.println(s.indexOf('a'));
        System.out.println(s.lastIndexOf('a'));
        System.out.println(s.equals(s1));
        System.out.println(s.equalsIgnoreCase(s));
        System.out.println(s.contains("march");
        String concat = s.concat(s1);
        System.out.println(concat);

        char charAt = s.charAt(7);
        System.out.println(charAt);

        System.out.println(s.replace("march","may");
```

```
        System.out.println(s3.isEmpty());
        System.out.println(s3.isBlank());
        System.out.println(s);
        System.out.println(s.trim());
        System.out.println(s.substring(3));
        System.out.println(s.substring(3,10));
    }
}
```

IMMUTABLE STRING:-

* A String is an immutable object which means we cannot change them after creating the objects. Whenever we change any string, a new instance is created.

* Immutable refers to something that cannot be changed or modified. Hence, immutable objects are ones that cannot be modified after they have been created.

This is an object whose internal state does not change after it has been created completely.

STRING BUILDER AND STRING BUFFER

[IMMUTABLE TO MUTABLE STRING]

```
package org.String;
Public class MutableString {
```

```
public void ImmutableString() {
    System.out.println("Immutable String:");
    String s = "March Batch";
    String s1 = "Three";
    System.out.println(s.hashCode());
    String concat = s.concat(s1);
    System.out.println(concat.hashCode());
}
public void mutableString() {
    System.out.println("Mutable String:");
    StringBuffer sb = new StringBuffer("March Batch");
    System.out.println(sb.hashCode());
    System.out.println(sb.append("Three"));
    System.out.println(sb.hashCode());
    System.out.println(sb.deleteCharAt(2));
    System.out.println(sb.insert(2,'r'));
    System.out.println(sb.replace(0,4,"May"));
}
public static void main(String[] args) {
    MutableString ms = new MutableString();
    ms.ImmutableString();
    ms.mutableString();
}
}
```

POINTS TO REMEMBER :-

* String is immutable whereas StringBuffer and StringBuilder are mutable classes.
* StringBuffer is thread-safe and synchronized whereas StringBuilder is not. That's why StringBuilder is faster than StringBuffer.
* String concatenation operator (+) internally uses StringBuffer or StringBuilder class.
* For String manipulations in a non-multi threaded environment, we should use StringBuilder else use StringBuffer class.