String (part - 2 - Mutable String)

Index

Mutable String

These are the strings which can be changed after their declaration.

Syntax to declare the **mutable string** are

For creating **mutable String** we have two classes **StringBuffer** and **StringBuilder**.

1. **StringBuffer** - released in java version - 1.0

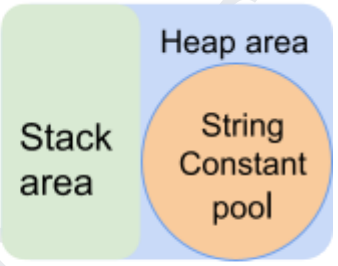
```
StringBuffer sbuff = new StringBuffer ("Java");
```

2 object will be created 1 in string pool and other in heap

2. **StringBuilder** - released in java version - 1.5

```
StringBuilder sbuild = new StringBuilder ("Program");
```

2 object will be created 1 in string pool and other in heap



Don't get confused, both are the same with little difference between them, that we will see further.

For now, we will be using **StringBuilder**.

We know **heap area**, **string constant pool** and **stack area**

Using `StringBuilder`

In **StringBuilder** and **StringBuffer** we don't have a `.concat()` method instead we have an `.append()` method to join two strings.

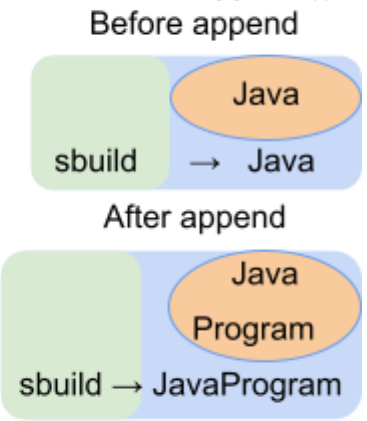
```
StringBuilder sbuild = new StringBuilder ("Java");

sbuild.append ("Program");

System.out.println (sbuild);
```

Output : JavaProgram

Here `sbuild` will get updated and **no new object** will be created.



If we try to change the String object the **change will happen inside the same memory**.

Comparison in `StringBuilder`

```
StringBuilder    sbuild1    =    new StringBuilder ("Java");

StringBuilder    sbuild2    =    new StringBuilder ("Java");

System.out.println (sbuild1  == sbuild2);          Output :     false
```

We know the == operator compares the reference of the object so the above statement's result will be false as we know it is stored in the heap area.

```
System.out.println (sbuild1.equals(sbuild2));     Output :     false
```

How is the answer false?

We have learnt that in **immutable string**, `equals()` method compares the values of the object.

In **StringBuffer** and **StringBuilder** `equals()` method is **Overridden**, so it also compares the reference value of the object.

`==` operator as well as `equals()`method both are comparing the reference value then how to compare the actual values of the **StringBuilder** object.

We can check character by character of both the values. By using a `for` loop.

But we will be using a better approach. We have already seen the `.compareTo()` function.

```
System.out.println (sbuild1.compareTo(sbuild2));       Output :   0
```

Output is 0 that means none of the characters inside the two stringBuilders are different. compareTo() method will stop when it finds the first non matching character inside the object by comparing ASCII values.

We can also check the value of StringBuilder objects in other ways by converting the StringBuilder object to string and then applying equals() method to it.

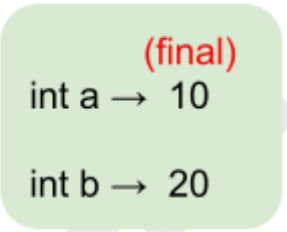
```
System.out.println (sbuild1.toString().equals(sbuild2.toString());
```

Output : **true**

`final` access keyword at variable level

If the variable is made as final then the value for those variables can't be changed,if we try to change it would result in *"CompileTimeError"*. final variables would be resolved at the compile time only by the compiler.

```
final int a = 10;
int b = 20;
b++;
a++;  //a=a+1 change for the variable: CompileTimeError
System.out.println (a);
System.out.println (b);
```

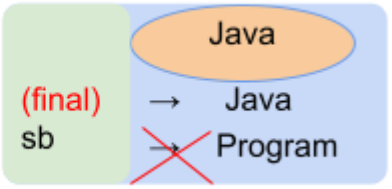


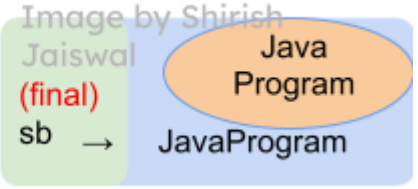
`final` variables value cannot be changed even if we try to change, it will throw a compile time error.

`final` in Immutable

What if we make StringBuilder object as final lets see what happens:

```
final StringBuilder sbuild1 = new StringBuilder ("Java");
//trying to change final variable
sbuild1.append("Program");
System.out.println (sbuild1);
Output :   JavaProgram
```





We have made `sbuild1` as `final` but the value of StringBuilder `sb` has been changed. **How?**

Because when we use the `final` keyword on StringBuilder it means the final property will not be applied on the value of stringbuilder but on the reference. So **changing value is allowed** but if we try to change the reference it will throw an error.

While we try to change the reference and name that reference to point to a different object it will give a compile time error.

```
sbuild1 = new StringBuilder("Program");
```

1. For a **primitive** data type variable if it is final then the **value of the variable cant be changed**, if we try to change it would result in "`CompileTimeError''`.
2. For a **reference** data type variable **if mutable in nature** then as per its mutable nature the object data can be changed, it won't result in any error, but if we try to reassign reference variable with a new object address then it would result in "`CompileTimeError`".

Capacity of the StringBuilder or StringBuffer

It basically denotes the amount of space available to store new characters.

```
StringBuilder    sbuild1    =    new StringBuilder ();

System.out.println (sbuild1.capacity());                    Output :   16
```

`.capacity()` → the default capacity of StringBuffer is 16

If the capacity is filled internally JVM will increase the size using following formulae

New Capacity = (current capacity + 1) * 2

```
StringBuilder    sbuild1    =    new StringBuilder (19);

//here the integer 19 specifies the capacity of StringBuffer

System.out.println (sbuild1.capacity());                    Output :   19
```

Difference StringBuilder and StringBuffer

StringBuffer → java version 1.0 V

- All methods present are synchronised
- At one time StringBuffer object can operate only one thread
- Since only one thread operates, it is "ThreadSafety".
- Performance is low.

StringBuilder → java version 1.5

- All the methods present are not synchronised
- At a time on a StringBuilder object can operate many threads
- Since many threads operate it's not "ThreadSafety".
- Performance is high.

Some methods in StringBuffer and StringBuilder

I have used `StringBuilder` but these methods work the same on `StringBuffer`.

StringBuilder Method	Output
`StringBuilder append(X x):` This method appends the string representation of the X type argument to the sequence	

`StringBuilder sb = new StringBuilder();`
`sb.append("JavA");`
`System.out.println (sb);`
	JavA

`int capacity():` This method returns the current capacity.	
`StringBuilder sb = new StringBuilder();` `System.out.println(sb.capacity());`	16
`char charAt(int index):` returns the char value in this sequence at the specified index.	
`StringBuilder sb = new StringBuilder("JavA");` `System.out.println(sb.charAt(2));`	v
`StringBuilder delete(int start, int end):` To delete the specific range of characters inside String.	
`StringBuilder sb = new StringBuilder("JavA");` `System.out.println(sb.delete(1, 3));`	JA
`int length():` To find the length of the string	
`StringBuilder sb = new StringBuilder("JavA");` `System.out.println(sb.length());`	4
`StringBuilder reverse():` To reverse the string	
`StringBuilder sb = new StringBuilder("JavA");` `System.out.println(sb.reverse());`	AvaJ
`StringBuilder replace(int start, int end, String str):` To replace the specific range of character with another string.	
`StringBuilder sb = new StringBuilder("JavA");` `System.out.println(sb.replace(1, 3, "C++"));`	JC++A
`int compareTo(obj):` To compare the actual value between two objects and return the ASCII difference between them	

`StringBuilder sb = new StringBuilder("JavA");`
`StringBuilder sb1 = new StringBuilder("JavA");`
`System.out.println(sb.compareTo(sb1));`
	0

void setLenght(int x): To change the length of the string object	

`StringBuilder sb = new StringBuilder("JavA");`
`sb.setLength(3);`
`System.out.println(sb);`
	Jav

StringBuilder insert(int x, X x): To insert the string inside the string object from specific index	
`StringBuilder sb = new StringBuilder("JavA");` `System.out.println(sb.insert(2, "+++"));`	Ja+++vA
boolean isEmpty(): To check whether the string is empty or not	
`StringBuilder sb = new StringBuilder("JavA");` `System.out.println(sb.isEmpty());`	false
int indexOf(String str): To find out the index of the string character inside the string object	
`StringBuilder sb = new StringBuilder("JavA");` `System.out.println(sb.indexOf("A"));`	3
void SetCharAt(int index, char ch): To replace the specific index char with new one	

`StringBuilder sb = new StringBuilder("JavA");`
`sb.setCharAt(1, 'A');`
`System.out.println(sb);`
	JAvA

int deleteCharAt(int index): To delete the specific index value inside the string	
`StringBuilder sb = new StringBuilder("JavA");` `System.out.println(sb.deleteCharAt(0));`	J