***What Is String in JAVA?***

***🡪*** *String is a sequence of characters that represents text. It is an object of the String class in the java.lang package. Strings are widely used to store and manipulate text in Java programs.*

*Here are a few key points about strings in Java:*

1. ***Immutable:*** *Strings in Java are immutable, meaning once a String object is created, its value cannot be changed. Any operation that seems to modify a string actually creates a new string.*
2. ***Immutable:****-* ***Immutable strings*** *refer to String objects that cannot be changed once they are created. This means that once a string is initialized with a certain value, it cannot be modified. If any modification is attempted (like appending, replacing characters, etc.), a new string object is created instead of changing the original one*

***There are two ways to create String Objects:****-*

1. ***Using Literal****.*

*Java String literal is created by using double quotes.*

*It directly refers to* ***“ Constant Pool Memory*** *also called the* ***string literal pool*** *“*

*For Example: 1. String s="welcome";*

1. ***Using new Keyword*** *– Dynamic memory allocation.*

***For Example:*** *String s=new String("Welcome");*

*//creates two objects and one reference variable.*

*It directly refers to “****Heap memory****”*

*When we create a String object using “ new “ keyword then Memory allocation is done in* ***“Heap memory****” and “* ***constant pool memory*** *”.*

*->But when we create String object using new it directly refers to Heap memory but if we want to forcefully want it to refer to contant pool memory then we need to add* ***intern() method .***

***For Eg:-***

*String s=new String(“Welcome”).intern();*

*intern() :- method is used to ensure that only one copy of a particular string is stored in memory. This method is part of the String class, and its purpose is to manage string memory efficiently by maintaining a* ***string pool*** *(also called the string literal pool).*

*String intern refers to string object in the string constant pool.*

***Interning****is the process of creating a string object in String Constant Pool which will be exact copy of string object in heap memory.*

***intern()****method**of java.lang.String class is used to perform interning i.e creating an exact copy of heap string object in string constant pool.*

*Initial Capacity of string is dynamic, whatever string will be passed it will be the length and capacity of that string object.*

***Note:-***

* 1. *Whenever you create a string object using string literal, JVM first checks content of an object to be created. If there exist an object in the pool with same content, then it returns reference of that object. It doesn’t create new object. If the content is different from the existing objects then only it creates new object.*

|  |  |  |
| --- | --- | --- |
| ***String Objects*** | ***StringBuffer Objects*** | ***StringBuilder Objects*** |
| *Immutable* | *Mutable* | *Mutable* |
| *Thread-safe* | *Thread-safe* | *Not thread-safe* |
| *Objects can be created either through string literal or through new operator* | *Objects can be created only through new operator* | *Objects can be created only through new operator* |
| *Objects are stored in string constant pool as well as heap memory* | *Objects are stored in heap memory only.* | *Objects are stored in heap memory only.* |
| *Slower* | *Slower* | *Faster* |

***StringBuffer:- (java.lang) package***

*A thread-safe object is one where multiple threads can safely operate on the object concurrently without causing data inconsistency or corruption. In the case of StringBuffer, thread safety is achieved by using synchronization.*

*StringBuffer:*

* *StringBuffer is thread-safe because its methods are synchronized.*
* *Synchronization ensures that only one thread can access a method of the StringBuffer at a time. This prevents the internal state of the StringBuffer from being corrupted when multiple threads try to modify it simultaneously.*
* *All string operations are performed on exiting objects only not like string as string create new object everytime , SB doesn’t creates new objects it makes modifications in existing one.*

***Pros of StringBuffer (Thread-Safe)***

* ***Thread Safety****: Safe to use in multi-threaded environments where multiple threads might be modifying the same StringBuffer object.*
* ***Synchronization****: Ensures consistency even when multiple threads are trying to modify it.*

***Cons of StringBuffer (Thread-Safe)***

* ***Slow Performance****: The synchronization mechanism adds some overhead. This can make StringBuffer slower than StringBuilder in single-threaded scenarios*

***Summary:***

* ***StringBuffer****:*
  + ***Thread-safe*** *(methods are synchronized).*
  + *Slower due to the overhead of synchronization.*
  + *Suitable for multi-threaded applications where thread safety is necessary.*
* ***StringBuilder****:*
  + ***Not thread-safe*** *(methods are not synchronized).*
  + *Faster in single-threaded or non-concurrent applications.*
  + *Suitable for single-threaded scenarios or when you manage synchronization explicitly.*

***StringBuilder:-***

*A* ***non-thread-safe*** *object is one where multiple threads* ***cannot*** *safely operate on the object concurrently. In the case of StringBuilder, it is* ***not synchronized****, meaning that multiple threads can access and modify the object simultaneously, potentially leading to inconsistent results.*

***StringBuilder:***

* *StringBuilder is* ***not thread-safe*** *because its methods are* ***not synchronized****.*
* *This makes StringBuilder faster than StringBuffer in scenarios where thread safety is not a concern (i.e., in single-threaded applications or when you know only one thread will modify the object at a time).*

***Pros of StringBuilder (Non-Thread-Safe)***

* *Better Performance: In single-threaded applications or when thread safety is not a concern, StringBuilder is faster than StringBuffer because it does not have the overhead of synchronization.*
* *Simpler Design: It is easier to use in situations where synchronization is either unnecessary or handled elsewhere.*

***Cons of StringBuilder (Non-Thread-Safe)***

* *Not Safe for Concurrent Use: If multiple threads are modifying the same StringBuilder instance concurrently, it could lead to data corruption or unexpected behavior. Care must be taken to ensure* only one thread accesses the object at a time.

***When to Use Which:***

* ***Use StringBuffer*** *if your application is multi-threaded and multiple threads might access the same string data.*
* ***Use StringBuilder*** *if you're working in a single-threaded context, or you are sure that only one thread will modify the string at a time. It's preferred when performance is critical.*

***String Comparison Operators:-***

* .*equals()*
* *== Operator*
* *.equalsIgnoreCase()*

1. ***.equals() Method compares Content of 2 Objects.***
2. ***==Operator compares Reference/Memory Address.***
3. ***.equalsIgnoreCase() compares two objects by Ignoring cases of them.***

* *If we are comparing any object created by new operator using ‘ == ’ Operator then its alwaysgoing to be false because new operator always assigns new memory and == check memory address.*
* *Only 2 literals can be compared using ==*
* *If we compare one literal and one object created using new using == Operator then always it will return false.*

***Difference between Object.equals and String.equals:-***

*Object.equals() compares two objects of Object class.*

*String.equals() compares two objects of String class.*

***For String Functions : Refer Java Project:: StringHandling in Eclipse***

***StringBuilder & StringBuffer Functions:-***

|  |  |  |
| --- | --- | --- |
| [***append(String str)***](https://www.geeksforgeeks.org/stringbuilder-append-method-in-java-with-examples/) | ***Appends the specified string to the end of the StringBuilder.*** | ***sb.append("Geeks");*** |
| ***insert(int offset, String)*** | ***Inserts the specified string at the given position in the StringBuilder.*** | ***sb.insert(5, " Geeks");*** |
| [***replace(int start, int end, String)***](https://www.geeksforgeeks.org/stringbuilder-replace-in-java-with-examples/) | ***Replaces characters in a substring with the specified string.*** | ***sb.replace(6, 11, "Geeks");*** |
| [***delete(int start, int end)***](https://www.geeksforgeeks.org/stringbuilder-delete-in-java-with-examples/) | ***Removes characters in the specified range.*** | ***sb.delete(5, 11);***  ***(begin,end-1)*** |
| [***reverse()***](https://www.geeksforgeeks.org/stringbuilder-reverse-in-java-with-examples/) | ***Reverses the sequence of characters in the StringBuilder.*** | ***sb.reverse();*** |
| [***capacity()***](https://www.geeksforgeeks.org/stringbuilder-capacity-in-java-with-examples/) | ***Returns the current capacity of the StringBuilder.*** | ***int cap = sb.capacity();*** |
| [***length()***](https://www.geeksforgeeks.org/stringbuilder-length-in-java-with-examples/) | ***Returns the number of characters in the StringBuilder.*** | ***int len = sb.length();*** |
| [***charAt(int index)***](https://www.geeksforgeeks.org/stringbuilder-charat-in-java-with-examples/) | ***Returns the character at the specified index.*** | ***char ch = sb.charAt(4);*** |
| [***setCharAt(int index, char)***](https://www.geeksforgeeks.org/stringbuilder-setcharat-in-java-with-examples/) | ***Replaces the character at the specified position with a new character.*** | ***sb.setCharAt(0, 'G');*** |
| [***substring(int start, int end)***](https://www.geeksforgeeks.org/stringbuilder-substring-method-in-java-with-examples/) | ***Returns a new String that contains characters from the specified range.*** | ***String sub = sb.substring(0, 5);*** |
| [***ensureCapacity(int minimum)***](https://www.geeksforgeeks.org/stringbuilder-ensurecapacity-in-java-with-examples/) | ***Ensures the capacity of the StringBuilder is at least equal to the specified minimum.*** | ***sb.ensureCapacity(50);*** |
| [***deleteCharAt(int index)***](https://www.geeksforgeeks.org/stringbuilder-deletecharat-in-java-with-examples/) | ***Removes the character at the specified position.*** | ***sb.deleteCharAt(3);*** |
| [***indexOf(String str)***](https://www.geeksforgeeks.org/java-string-indexof/) | ***Returns the index of the first occurrence of the specified string.*** | ***int idx = sb.indexOf("Geeks");*** |
| [***lastIndexOf(String str)***](https://www.geeksforgeeks.org/stringbuilder-lastindexof-method-in-java-with-examples/) | ***Returns the index of the last occurrence of the specified string.*** | ***int idx = sb.lastIndexOf("Geeks");*** |
| [***toString()***](https://www.geeksforgeeks.org/stringbuilder-tostring-method-in-java-with-examples/) | ***Converts the StringBuilder object to a String.*** | ***String result = sb.toString();*** |

***String buffer creates an empty constructor with an initial capacity. 🡪16 Characters***

| ***String Method*** | ***Description*** |
| --- | --- |
| ***1.***[***int length()***](https://www.geeksforgeeks.org/java-string-length-method-with-examples/) | ***Returns the number of characters in the String.*** |
| ***2.***[***Char charAt(int i)***](https://www.geeksforgeeks.org/java-string-charat-method-example/) | ***Returns the character at ith index.*** |
| ***3.***[***String substring (int i)***](https://www.geeksforgeeks.org/substring-in-java/) | ***Return the substring from the ith  index character to end.*** |
| ***4.***[***String substring (int i, int j)***](https://www.geeksforgeeks.org/substring-in-java/) | ***Returns the substring from i to j-1 index.*** |
| ***5.***[***String concat( String str)***](https://www.geeksforgeeks.org/java-string-concat-examples/) | ***Concatenates specified string to the end of this string.*** |
| ***6.***[***int indexOf (String s)***](https://www.geeksforgeeks.org/java-string-indexof/) | ***Returns the index within the string of the first occurrence of the specified string. If String s is not present in input string then -1 is returned as the default value.*** |
| ***7.***[***int indexOf (String s, int i)***](https://www.geeksforgeeks.org/java-string-indexof/) | ***Returns the index within the string of the first occurrence of the specified string, starting at the specified index.*** |
| ***8.***[***Int lastIndexOf( String s)***](https://www.geeksforgeeks.org/java-lang-string-lastindexof-method/) | ***Returns the index within the string of the last occurrence of the specified string. If String s is not present in input string then -1 is returned as the default value.*** |
| ***9. [boolean equals( Object otherObj)](https://www.geeksforgeeks.org/boolean-equals-method-in-java-with-examples/)*** | ***Compares this string to the specified object.*** |
| ***10. [boolean  equalsIgnoreCase (String anotherString)](https://www.geeksforgeeks.org/equalsignorecase-in-java/)*** | ***Compares string to another string, ignoring case considerations.*** |
| [***11. int compareTo( String anotherString)***](https://www.geeksforgeeks.org/java-lang-string-compareto/) | ***Compares two string lexicographically.*** |
| ***12.***[***int compareToIgnoreCase( String anotherString)***](https://www.geeksforgeeks.org/java-string-compareto-method-with-examples/) | ***Compares two string lexicographically, ignoring case considerations.***  ***Note: In this case, it will not consider case of a letter (it will ignore whether it is uppercase or lowercase).*** |
| ***13.***[***String toLowerCase()***](https://www.geeksforgeeks.org/java-string-tolowercase-examples/) | ***Converts all the characters in the String to lower case.*** |
| ***14.***[***String toUpperCase()***](https://www.geeksforgeeks.org/java-touppercase-examples/) | ***Converts all the characters in the String to upper case.*** |
| ***15.***[***String trim()***](https://www.geeksforgeeks.org/java-string-trim-method-example/) | ***Returns the copy of the String, by removing whitespaces at both ends. It does not affect whitespaces in the middle.*** |
| ***16.***[***String replace (char oldChar, char newChar)***](https://www.geeksforgeeks.org/java-lang-string-replace-method-java/) | ***Returns new string by replacing all occurrences of oldChar with newChar.***  ***Note: s1 is still feeksforfeeks and s2 is geeksgorgeeks*** |
| ***17. [boolean contains(CharSequence sequence)](https://www.geeksforgeeks.org/java-string-contains-method-example/" \t "_blank)*** | ***Returns true if string contains contains the given string.*** |
| ***18.***[***Char[] toCharArray():***](https://www.geeksforgeeks.org/java-string-tochararray-example/) | ***Converts this String to a new character array.*** |
| ***19.***[***boolean startsWith(String prefix)***](https://www.geeksforgeeks.org/string-startswith-method-in-java-with-examples/) | ***Return true if string starts with this prefix.*** |
| ***20. trimToSize();*** | ***Deallocates unused memory of String Buffer to the size of parameter passed.Used to free up extra allocated memory*** |

*🡪 Whenever we create an object using “ new” keyword, object will be created in heap memory area as well as string constant pool area for reusability.*

*Object created In SCP implicit reference variable will be created automatically by JVM so it wont be eligible for Garbage collection. SCP area is present in method area/ permgen till JAVA 1.6. But after 1.6 it gets stored in 1.7 for better memory utilization.*

*Whenever string object/string variable is created at runtime then it gets stroed in heap memory. But Whenever final string variable is created it is considered as constant and gets stored in SCP.*

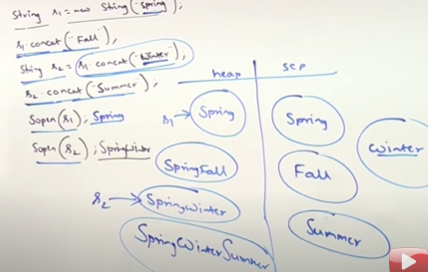
*🡪 When we create object using string literal then only one object created in SCP area…*

*In SCP area, with the same content there cannot be two/more different objects all the objects having same content created using literals will have single reference only the reference name will be different but they will point to same /common object in SCP.*

[*https://www.youtube.com/watch?v=mCM8OeM2CAc&list=PLd3UqWTnYXOlrdBwt2DTnF3u3W-1Kozk0&index=6*](https://www.youtube.com/watch?v=mCM8OeM2CAc&list=PLd3UqWTnYXOlrdBwt2DTnF3u3W-1Kozk0&index=6) *.Usage of SCP will contribute in reusability and maintain higher performance ,and lesser memory usage.*

*🡪Because of any runtime operation if thers is any change in object content then new object will be created in Heap area with provided reference name, but if the content is same and not changed then the new refernce name will point towards existing object only.*

***This rule is same and follows for both Heap area and SCP area.***



*Eg:-* ***VoterRegistrationForm****🡪same address can be used by n number of people having same address, so address will be stored in SCP and all references will point to that object, but if someone comes with different address than others then new object will be created in SCP with new reference so the previous address wont be changed due to immutable behavior of string objects.. This was the main motive of making string immutable*

*Q1🡪Except String are there any other objects immutable In java?*

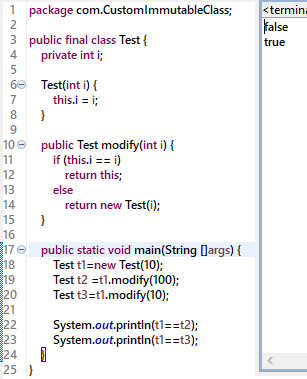
*Yes, all Wrapper classes are immutable in java.*

***Important Constructors of String Class:-***

* 1. *String s =new String() ; 🡪Creates an empty String*
  2. *String s= new String(String literal); 🡪 Creates an object in Heap and SCP*
  3. *String s = new Strin(StringBuffer sb);*
  4. *String s= new String(StringBuilder sb);*
  5. *String s = new String(char[] ch);🡪 returns a string when char array is passed*
  6. *String s = new String(byte[] by);🡪returns ascii code values for byte array numbers passed.*

***Q) Can we create custom/user defined immutable class?***

***Yes we can do it .***



***Important Constructors of StringBuffer:-***

* 1. *StringBuffer sb=new StringBuffer(); 🡪Default capacity 16 characters.*
     + *If initial Capacity is filled fully then new object will be created and existing reference variable name will be given to new object.*

*New Capacity=(Current capacity+1)\*2*

*(16+1)\*2=34*

*(34+1)\*2=70*

* 1. *StringBuffer sb=new StringBuffer(int initialCapacity);*

🡪*Can create StringBuffer object with specified capacity*

* 1. *StringBuffer sb=new StringBuffer(String s);*

*For the passed string an equivalent StringBuffer will be created.*

*This constructor will have capacity as:-*

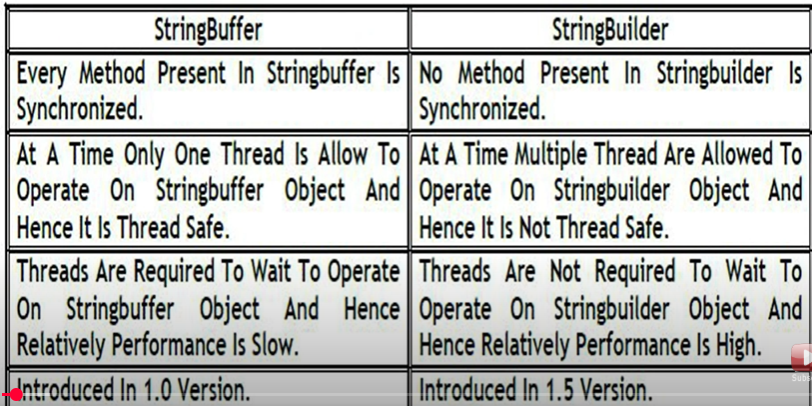
*Capacity=s.length()+16;*

*StringBuffer sb=new StringBuffer(“Yash”);*

*Capacity=4+16🡪20*

*Need of* ***StringBuilder*** *in java****:-***

* 1. *In* ***StringBuffer each and every method is Syncrhonised*** *which means at a time only one thread is allowed to operate the StringBuffer object/data. Which increases waiting time of thread and performance goes down.*
  2. *To overcome this problem in* ***Java 1.5 version*** *StringBuilder concept was introduced.*
  3. *Therefore in Multithreaded environment StringBuilder cannot be used.*



*When to go for* ***String****:-*

*1)When the content is going to be fixed and not going to change frequently , then we should use String objects.*

*2)Strings are always thread safe, because when we make any change to String objects then new String object gets created. therefore operations are applicable and running for single thread only.*

*When to go for* ***StringBuffer****:-*

*1)If the content is not constant / fixed and is going to be changed frequently as well as thread safety is also important then we should use StringBuffer.*

*When to go for* ***StringBuilder****:-*

*1)If the content is not fixed/constant and is going to be changed frequently and thread safety is not important and performance is prioritized then we should go for StringBuilder.*

***(All wrapper classes and Immutable classes/objects are thread safe in java)***