String

- Areas where an entity can be stored
    1. Stack - Primitive & Reference of Non-Primitives
    2. Heap - Non-Primitives
    3. String Pool - Strings
    4. Method Area - Methods

- The Maximum Memory Can be Wasted due to String. If we create a string using new keyword.

- String will be Stored inside the **String Pool, due to Memory Efficiency. But note that This will only be helpful when you don't want to do manipulation inside the created string. If we want to do manipulation inside the already created string later, then we should use String Builder class instead of String class.**

- We can also store String inside Heap using **new** keyword.

- Java String Class is **Immutable for Security reasons**.Because If some variables are referencing the same string pool and If we make changes into string using any one variable so that string will change for all variables.So If we create any String Variable, we can't make any changes inside it.
    - Because behind the scenes the string variable will be stored inside array of bytes and it's final. So we can't change the reference of that variable.It will copy the old value then will make changes inside it and Will return that modified string.
    - String class is final. So we can't extend it.

- **Storage**

    - String s="Rahul" ; → String Created using **String Literals**.→ This will be stored inside the **String Pool.**
    - String s= new String("Rahul") ; → String Created using **new** Keyword → This will be stored inside the **Heap Memory.**

    - **Benefits of String Pool over Heap Memory.**

        - The benefit of creating String using **String Literal** is that it will be created inside the **string pool** & if we create new string variable with the same string then the compiler will check first that is that string already created inside the string pool ? if yes then it will not create the new one. This work will be done using intern method of String class.

- But if we create a String using **new** keyword then it will create a new string instance for each and every string. Although if both strings are the same, it will still create a new String, thus it's a memory wastage.

- **Shallow Comparison & Deep Comparison**

    1. == Operator → **Shallow Comparison** → It compares only addresses which will be stored inside the reference in the stack of both operands.

    2. String**.equals()** Method → **Deep Comparison** → first it will compare the address of both operands, if address is same then will return true and if address is different then It compares the length of both strings and then compares each element of both char arrays.

- **Examples**

```
String s1="Rahul";// here intern method will be call implicitly
String s2="Rahul";// here intern method will be call implicitly

String s3=new String("Rahul");//here new reference of string will be created and
will be stored inside heap
String s4=new String("Rahul");//here new reference of string will be created and
will be stored inside heap

System.out.println(s1==s3); //false --> Shallow comparison --> Address Check
System.out.println(s3==s4); //false --> reference or address is different

System.out.println(s1.equals(s3)); //true --> Deep Comparison --> content check
System.out.println(s1==s2); //true -> Address is same


System.out.println(s3.equals(s4)); // true
```

- **Important Methods**

- **Basic String Methods**

  1.`length()` - Returns the length of the string.

  2.`charAt(int index)` - Returns the character at the specified index.

  3.`substring(int beginIndex)` - Returns a substring from the given index.

  4.`substring(int beginIndex, int endIndex)` - Returns a substring between the specified indices.

  5.`concat(String str)` - Concatenates the specified string to the end of this string.

- **Comparison Methods**

  6.`equals(String anotherString)` - Compares two strings for equality.

  7.`equalsIgnoreCase(String anotherString)` - Compares two strings, ignoring case differences.

  8.`compareTo(String anotherString)` - Compares two strings lexicographically.

  9.`compareToIgnoreCase(String anotherString)` - Compares two strings lexicographically, ignoring case.

- **Searching Methods**

  10.`indexOf(String str)` - Returns the index of the first occurrence of the specified string.

  11.`indexOf(String str, int fromIndex)` - Returns the index of the first occurrence from the given index.

  12.`lastIndexOf(String str)` - Returns the index of the last occurrence of the specified string.

  13.`contains(CharSequence s)` - Checks if the string contains the specified sequence of characters.

  14.`startsWith(String prefix)` - Checks if the string starts with the specified prefix.

  15.`endsWith(String suffix)` - Checks if the string ends with the specified suffix.

- **Modification Methods**

  16.`toLowerCase()` - Converts all characters in the string to lowercase.

  17.`toUpperCase()` - Converts all characters in the string to uppercase.

  18.`trim()` - Removes leading and trailing spaces.

  19.`replace(char oldChar, char newChar)` - Replaces all occurrences of a character.

  20.`replace(CharSequence target, CharSequence replacement)` - Replaces all occurrences of a substring.

- **Splitting & Joining Methods**

  21.`split(String regex)` - Splits the string around matches of the given regex.

  22.`split(String regex, int limit)` - Splits the string around matches, limiting the number of results.

`23.join(CharSequence delimiter, CharSequence... elements)` **- Joins multiple strings with the given delimiter.**

- **Conversion Methods**

  `24.toCharArray()` **- Converts the string to a character array.**
  `25.getBytes()` **- Converts the string into a byte array.**
  `26.valueOf(int/float/double etc.)` **- Converts a primitive type or object into a string.**
  `27.intern()` **- Returns a canonical representation of the string.**

- **Regular Expression Methods**

  `28.matches(String regex)` **- Checks if the string matches the given regex.**
  `29.replaceAll(String regex, String replacement)` **- Replaces all occurrences of the regex pattern.**
  `30.replaceFirst(String regex, String replacement)` **- Replaces the first occurrence of the regex pattern.**

- Basic String Methods

1. **length**
   - **Returns length of string or we can say the length of char array.**

```
       String name="Mayank";
//        System.out.println(name.length()); // 6
```

2. **charAt**
   - **Will return the character at given index**

```
//        System.out.println(name.charAt(2)); //y
//        System.out.println(name.charAt(6)); // error --> index out of bound
```

3. **subString**
   - **Can slice substring from a String.**

```
//        System.out.println(name.substring(1)); //ayank
//        System.out.println(name.substring(1,3)); //ay
```

4. **concat**
   - **Will join two strings and will return it.**

```
//         System.out.println(name.concat(" Yadav")); //Mayank Yadav
```

- Comparison Methods

5. **equals**
    - **Makes a deep comparison between two strings.**

```
//         System.out.println(name.equals("Mayank")); //true --> will make deep
comparison
//         System.out.println(name.equals("mayank")); //false --> will make deep
comparison
```

6. **equalsIgnoreCase**
    - **Will do deep comparison but will ignore upper and lower case.**

```
//         System.out.println(name.equalsIgnoreCase("mayank"));//true
```

7. **compareTo**
    - **It compares both strings lexicographically. Where each character will be compared with another one based on UNICODE values.**

```
//         System.out.println(name.compareTo("Mayank")); //0
//         System.out.println(name.compareTo("mayank")); //-32
//         System.out.println(name.compareTo("Maya")); //2
//         System.out.println("azcdefgx".compareTo("abddefz")); //24
//         System.out.println("a".compareTo("b"));//-1
//         System.out.println("Rahul".compareTo("Rohit")); //-14
//         System.out.println("Rohan".compareTo("Rohit")); //-8
```

8. **compareToIgnoreCase**
    - **It will work same as compareTo method but it will Ignore the case.**

```
//         System.out.println(name.compareToIgnoreCase("mayank"));//0
```

- **Searching Methods**

9. **indexOf**
    - **It will return the index of first occurrence in the given string.**

```
//          System.out.println(name.indexOf("M"));//0
//          System.out.println(name.indexOf("a"));//1
//          System.out.println(name.indexOf("k"));//5
//          System.out.println(name.indexOf("z"));//-1
```

10. **indexOf**
    - **It can also return the index between given range.**

```
//          System.out.println("Rahul Bhutaiya".indexOf("a",1));//1
//          System.out.println("Rahul Bhutaiya".indexOf("a",2));//10

//          System.out.println("Rahul Bhutaiya".indexOf("a",2,11));//10
```

11. **lastIndexOf**
    - **Returns the index of the last occurrence of the given string.**

```
//          System.out.println(name.lastIndexOf("a"));//3
//          System.out.println(name.lastIndexOf("a",2));//1
```

12. **contains**
    - **Checks if the given string contains the specified sequence of characters.**

```
//          System.out.println(name.contains("aya"));//true
//          System.out.println(name.contains("Maja"));//false
//          System.out.println("Add Zero Group".contains("Group"));//true
//          System.out.println("Add Zero Group".contains("group"));//false
```

13. startsWith
   - **Checks if the string starts with the specified prefix.**

```
//         System.out.println(name.startsWith("Ma"));//true
//         System.out.println(name.startsWith("ayank"));//false
//         System.out.println(name.startsWith("ma"));//false
//         System.out.println(name.startsWith("aya",1));//true
//         System.out.println(name.startsWith("Maya",1));//false
//         System.out.println("Add Zero Group".startsWith("Zero",4));//true
```

14. endsWith
   - **Checks if the string ends with the specified suffix.**

```
//         System.out.println(name.endsWith("k"));//true
//         System.out.println(name.endsWith("nkl"));//false
```

- Modification Methods

15. toLowerCase

```
//          System.out.println(name.toLowerCase());//mayank
```

16. toUpperCase

```
//          System.out.println(name.toUpperCase());//MAYANK
```

17. trim

- **Removes leading and trailing spaces**

```
//          System.out.println("    Rahul".trim());
//          System.out.println("    Rahul     ".trim());
```

18. replace

- **Replaces all occurrences of character of substring.**

```
//          System.out.println(name.replace("a","o"));//Moyonk
//          System.out.println("Add Zero Group".replace(" ","&"));//Add&Zero&Group
//          System.out.println("Add Zero Group".replace("rahul","&"));//Add Zero
Group
```

- Splitting & Joining Methods

19. split

- **It splits the string around matches of the given regex, and can also limit the number of results.**

```
//          System.out.println(Arrays.toString("Add Zero Group".split(" ")));//[Add, Zero, Group]
//          System.out.println(Arrays.toString("Add Zero Group".split("d")));//[A, , Zero Group]
//          System.out.println(Arrays.toString("Add Zero Group".split(" ",2)));//[Add, Zero Group]
```

20. join

- **Joins multiple strings with the given separator.**

```
//          String[] nameArr={"Add","Zero","Group"};
//          System.out.println(String.join(" ",nameArr));//Add Zero Group
//          System.out.println(String.join(" ","Rahul","Bhutaiya"));//Rahul Bhutaiya
```

- Conversion Methods

21. **toCharArray**
    - **Converts the string to a character array.**

```
//         char[] charArr;
//         charArr=name.toCharArray();
//         System.out.println(Arrays.toString(charArr));//[M, a, y, a, n, k]
```

22. **getBytes**
    - **Converts the string into a byte array.**

```
//         System.out.println(Arrays.toString(name.getBytes()));//[77, 97, 121,
97, 110, 107]
```

23. **valueOf**
    - **Converts a primitive type or object into a string.**

```
//         System.out.println(String.valueOf(10));//10
```

24. **intern**

```
//         String s1 = "d".intern();
```

- **Here, intern method will check that, is this string already exists in string constant pool ? if yes then the address of that string will be assigned to s1 else intern method will create a new string object inside scp.**

```
//         Following are some important points to remember regarding the intern()
method:
```

```
//        1) A string literal always invokes the intern() method, whether one
mention the intern() method along with the string literal or not. For example,
//        String s1 = "d".intern();
//        String s2 = "d"; // compiler treats it as String p = "d".intern();
//        System.out.println(s1 == s2); // prints true

//        2) Whenever we create a String object using the new keyword, two
objects are created. For example,
//        String str = new ("Hello World");
//        Here, one object is created in the heap memory outside of the SCP
because of the usage of the new keyword. As we have got the string literal too
("Hello World"); therefore, one object is created inside the SCP, provided the
literal "Hello World" is already not present in the SCP.
```

- Regular Expression Methods

25. **matches**
    - **Check if the string matches the given regex.**

```
//        System.out.println("hello".equals(".*e.*")); // false
//        System.out.println("hello".matches(".*e.*")); // true
//        The key difference is that matches matches a regular expressions
whereas equals can only match a specific String.
```

26. **replaceAll**
    - **Replaces all occurrences of the regex pattern.**

```
//        System.out.println(name.replaceAll("a","o"));//Moyonk
```

27. **replaceFirst**
    - **Replaces the first occurrence of the regex pattern.**

```
//        System.out.println(name.replaceFirst("a","o"));//Moyank
```

28. **isEmpty**
    - **Check that the given string is empty or not.**

```
//        System.out.println(name.isEmpty());//false
//        System.out.println("".isEmpty());//true
//        System.out.println(" ".isEmpty());//false
```

29. **isBlank**

- **Check that the given string has any character? If it contains any character, then will return false and if String is empty or string contains any white-space then will return true.**

```
//         System.out.println("".isBlank());//true
//         System.out.println(" ".isBlank());//true
```