Strings

- String Declaration in Java

```
public static void main(String args[]) {
char arr[]={'C','o','d','i','n','g'}; //Declaration of string as a character array.
String str="Coding";    // Best way to declare the array.
String str1=" "; //To create an empty string.
```

Note: Space is count as a character.

```
System.out.println(str1.length()); //To find the length of an string.
System.out.println(str.charAt(2)); // To use the character at particular position.
```

Note: In java we don’t access element like str[i];

```
System.out.println(str.charAt(6));
}
```

Note:

- The String class length method returns an int, the maximum length that would be returned by the method would be Integer.MAX_VALUE, which is 2^31 - 1, which is equivalent to 2,14,74,83,647.
- String is a non-primitive datatype.
- The length() function returns an integer value.
- The maximum length of String in java is 2,14,74,83,647.
- Strings can store spaces as well.

- Different Functions on Strings

I. Merge two string

The one way is using normal addition

```
String str = "Sanskar";
String str1 = " Gupta";
System.out.print(str+str1);
```

The second way is using concat keyword:

```
String str2 = str.concat(str1);
System.out.println(str2);
```

The out of both the way will be Sanskar Gupta

Strings

- Difference between Concat and +

Although concat() method of Strings class and + operator are both used for the concatenation of strings, there are some differences between them which are depicted below in the tabular format as follows:

Factor 1: Number of arguments the concat() method and + operator takes

- **concat()** method takes only one argument of string and concatenates it with other string.
- **+ operator** takes any number of arguments and concatenates all the strings.

Example

- Java

```
// Java Program to Illustrate concat() method
// vs + operator in Strings

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Custom input string
        String s = "Geeks", t = "for", g = "geeks";

        // Printing combined string using + operator
        System.out.println(s + t + g);

        // Printing combined string using concat() method
        System.out.println(s.concat(t));
    }
}
```

Output

```
Geeksforgeeks
Geeksfor
```

Factor 2: Type of arguments

- string **concat()** method takes only string arguments, if there is any other type is given in arguments then it will raise an error.
- **+ operator** takes any type and converts to string type and then concatenates the strings.

Factor 3: *concat() method* raises java.lang.NullPointer Exception

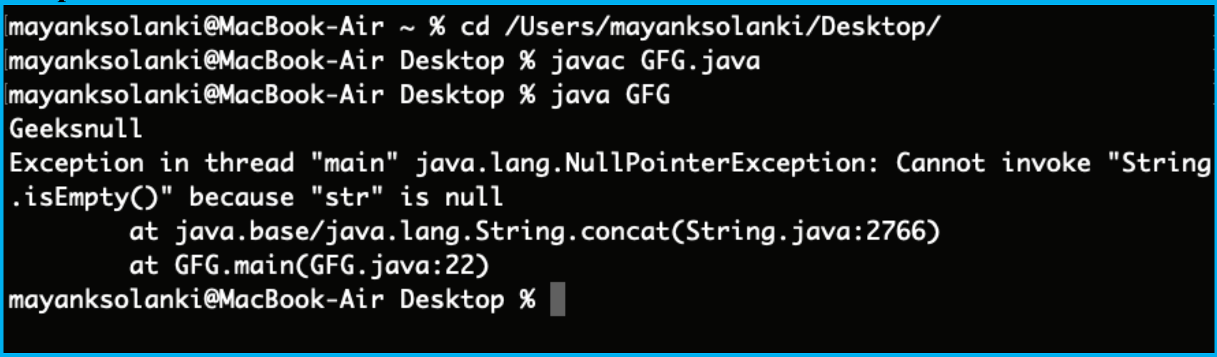
- **concat() method** throws NullPointer Exception when a string is concatenated with null
- **+ operator** did not raise any Exception when the string is concatenated with null.

Example

- Java

```
// Java Program to Illustrate Raise of NullPointer Exception
// in case of concat() Method
// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Input string 1
        String s = "Geeks";
        // Input string 2
        String r = null;

        // Combining above strings using + operator and
        //  printing resultant string
        System.out.println(s + r);

        // Combining above strings using concat() method and
        // printing resultant string
        // It raises an NullPointer Exception
        System.out.println(s.concat(r));
    }
}
```

Output:



```
mayanksolanki@MacBook-Air ~ % cd /Users/mayanksolanki/Desktop/
mayanksolanki@MacBook-Air Desktop % javac GFG.java
mayanksolanki@MacBook-Air Desktop % java GFG
Geeksnull
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String
.isEmpty()" because "str" is null
        at java.base/java.lang.String.concat(String.java:2766)
        at GFG.main(GFG.java:22)
mayanksolanki@MacBook-Air Desktop %
```

Factor 4: Creates a new String object

- **concat() method** takes concatenates two strings and returns a new string object only string length is greater than 0, otherwise, it returns the same object.
- **+ operator** creates a new String object every time irrespective of the length of the string.

Example

- Java

```
// Java Program to Illustrate Creation of New String object
// in concat() method and + Operator

// Main class
public class GFG {
    // Main driver method
    public static void main(String[] args)
    {

        // Input strings
        String s = "Geeks", g = "";

        // Using concat() method over strings
        String f = s.concat(g);

        // Checking if both strings are equal

        // Case 1
        if (f == s)

            // Identical strings
            System.out.println("Both are same");

        else

            // Non-identical strings
            System.out.println("not same");

        // Case 2
        String e = s + g;

        // Again checking if both strings are equal
        if (e == s)

            // Identical strings
            System.out.println("Both are same");
        else

            // Non-identical strings
            System.out.println("not same");
    }
}
```

Output

```
Both are same
not same
```

Factor 5: Performance

concat() method is better than the **+ operator** because it creates a new object only when the string length is greater than zero(0) but the + operator always creates a new string irrespective of the length of the string.

Conclusion: *From above two programs we draw see that both are somehow adding up two strings directly or indirectly. In concat method we are bound to while adding strings as we have to pass the string in the parameter while in other case of '+' operator, it is simply adding up strings likely mathematics, so there is no bound we can add either side.*

II. Check weather two string are equal or not

The first way is using ==

System.out.println(str == str1);

The second way is using equal method

System.out.println(str.equals(str1));

- Difference between comparing String using == and .equals() method in Java
  Both equals() method and the == operator are used to compare two objects in Java. == is an operator and equals() is method. But == operator compares reference or memory location of objects in a heap, whether they point to the same location or not.
  Whenever we create an object using the operator new, it will create a new memory location for that object. So we use the == operator to check memory location or address of two objects are the same or not.
  In general, both equals() and "==" operators in Java are used to compare objects to check equality, but here are some of the differences between the two:

1. The main difference between the .equals() method and == operator is that one is a method, and the other is the operator.
2. We can use == operators for reference comparison (**address comparison**) and .equals() method for **content comparison**. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.
3. If a class does not override the equals method, then by default, it uses the equals(Object o) method of the closest parent class that has overridden this method. See Why to Override equals(Object) and hashCode() method? in detail.

- Java

```
// Java program to understand
// the concept of == operator

public class Test {
    public static void main(String[] args)
    {
        String s1 = "HELLO";
        String s2 = "HELLO";
        String s3 =  new String("HELLO");

        System.out.println(s1 == s2); // true
        System.out.println(s1 == s3); // false
        System.out.println(s1.equals(s2)); // true
        System.out.println(s1.equals(s3)); // true
    }
}
```

Output

```
true
false
true
true
```

III. Java String compareTo() Method with Examples

Strings in Java are objects that are supported internally by a char array. Since arrays are immutable, and strings are also a type of exceptional array that holds characters, therefore, strings are immutable as well.

The String class of Java comprises a lot of methods to execute various operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), substring(), etc. Out of these methods, we will be focusing on the Java String **compareTo()** method in this article.

Java String.compareTo() Method

The Java String class compareTo() method compares the given string with the current string lexicographically. It returns a positive number, a negative number, or 0. It compares strings on the basis of the Unicode value of each character in the strings.

The compareTo function returns the difference of length of strings when they are not equal.In case if the lengths are equal, then it returns the non-zero difference in ASCII values starting from 0th index to (n-1)st index and if all the indices of string are same, it returns 0.

- If the first string is lexicographically greater than the second string, it returns a positive number (difference of character value).
- If the first string is less than the second string lexicographically, it returns a negative number, and,
- If the first string is lexicographically equal to the second string, it returns 0.

Note:

- *if **string1 > string2**, it returns **positive** number*
- *if **string1 < string2**, it returns **negative** number*
- *if **string1 == string2**, it returns **0***

There are **three** variants of the **compareTo()** method which are as follows:

1. using int compareTo(Object obj)
2. using int compareTo(String anotherString)

3. using int compareToIgnoreCase(String str)

1. int compareTo(Object obj)

This method compares this String to another Object.

Syntax:

int compareTo(Object obj)

Parameters:

obj: the Object to be compared.

Return Value: The value 0 if the argument is a string lexicographically equal to this string; a value less than 0 if the argument is a string lexicographically greater than this string; and a value greater than 0 if the argument is a string lexicographically less than this string.

Below is the implementation of the above method:

- Java

```
// Java code to demonstrate the
// working of compareTo()
public class Cmp1 {
    public static void main(String args[])
    {

        // Initializing Strings
        String str1 = "geeksforgeeks";
        String str2 = new String("geeksforgeeks");
        String str3 = new String("astha");

        // Checking if geeksforgeeks string
        // equates to geeksforgeeks object
        System.out.print(
            "Difference of geeksforgeeks(obj) and geeksforgeeks(str) : ");
        System.out.println(str1.compareTo(str2));

        // Checking if geeksforgeeks string
        // equates to astha object
        System.out.print(
            "Difference of astha(obj) and geeksforgeeks(str) : ");
        System.out.println(str1.compareTo(str3));
    }
}
```

Output

```
Difference of geeksforgeeks(obj) and geeksforgeeks(str) : 0
Difference of astha(obj) and geeksforgeeks(str) : 6
```

2. int compareTo(String anotherString)

This method compares two strings lexicographically.

Syntax:

int compareTo(String anotherString)

Parameters:

anotherString: the String to be compared.

Return Value: The value 0 if the argument is a string lexicographically equal to this string; a value less than 0 if the argument is a string lexicographically greater than this string; and a value greater than 0 if the argument is a string lexicographically less than this string.

Below is the implementation of the above method:

- Java

```
// Java code to demonstrate the
// working of compareTo()

public class Cmp2 {
    public static void main(String args[])
    {

        // Initializing Strings
        String str1 = "geeksforgeeks";
        String str2 = "geeksforgeeks";
        String str3 = "astha";

        // Checking if geeksforgeeks string
        // equates to geeksforgeeks string
        System.out.print(
            "Difference of geeksforgeeks(str) and geeksforgeeks(str) : ");
        System.out.println(str1.compareTo(str2));

        // Checking if geeksforgeeks string
        // equates to astha string
        System.out.print(
            "Difference of astha(str) and geeksforgeeks(str) : ");
        System.out.println(str1.compareTo(str3));
    }
}
```

Output

```
Difference of geeksforgeeks(str) and geeksforgeeks(str) : 0
Difference of astha(str) and geeksforgeeks(str) : 6
```

3. int compareToIgnoreCase(String str)

This method compares two strings lexicographically, ignoring case differences.

Syntax:

int compareToIgnoreCase(String str)

Parameters:

str: the String to be compared.

Return Value: This method returns a negative integer, zero, or a positive integer as the specified String is greater than, equal to, or less than this String, ignoring case considerations.

Below is the implementation of the above method:

- Java

```
// Java code to demonstrate the
// working of compareToIgnoreCase()

public class Cmp3 {
    public static void main(String args[])
    {

        // Initializing Strings
        String str1 = "geeks";
        String str2 = "gEEkS";

        // Checking if geeksforgeeks string
        // equates to geeksforgeeks string
        // case sensitive
        System.out.print(
            "Difference of geeks and gEEkS (case sensitive) : ");
        System.out.println(str1.compareTo(str2));

        // Checking if geeksforgeeks string
        // equates to astha string
        // case insensitive
        // using compareToIgnoreCase()
        System.out.print(
            "Difference of geeks and gEEkS (case insensitive) : ");
        System.out.println(str1.compareToIgnoreCase(str2));
    }
}
```

Output

```
Difference of geeks and gEEkS (case sensitive) : 32
Difference of geeks and gEEkS (case insensitive) : 0
```

IV. Java String contains() method with example

The java.lang.String.contains() method searches the sequence of characters in the given string. It returns true if the sequence of char values is found in this string otherwise returns false.

Implementation of contains() method

```
public boolean contains(CharSequence sequence)
{
   return indexOf(sequence.toString()) > -1;
}
```

Here conversion of CharSequence to a String takes place and then the indexOf method is called. The method **indexOf** returns **O** or a **higher number** if it finds the String, otherwise **-1** is returned. So, after execution, contains() method returns **true** if the sequence of char values exists, otherwise **false**.

Syntax of contains() method

public boolean **contains**(CharSequence *sequence*);

Parameter

- **sequence:** This is the sequence of characters to be searched.

Exception

- **NullPointerException:** If seq is null

Examples of the java.string.contains() method

Example 1: To check whether the charSequence is present or not.

- Java

```
// Java program to demonstrate working
// contains() method
class Gfg {

    // Driver code
    public static void main(String args[])
    {
        String s1 = "My name is GFG";

        // prints true
        System.out.println(s1.contains("GFG"));

        // prints false
        System.out.println(s1.contains("geeks"));
    }
}
```

Output

```
true
false
```

Example 2: Case-sensitive method to check whether given CharSequence is present or not.

- Java

```
// Java code to demonstrate case
// sensitivity of contains() method
class Gfg1 {

    // Driver code
    public static void main(String args[])
    {
        String s1 = "Welcome! to GFG";

        // prints false
        System.out.println(s1.contains("Gfg"));

        // prints true
        System.out.println(s1.contains("GFG"));
    }
}
```

Output

```
false
true
```

Points to remember with Java string contains() method

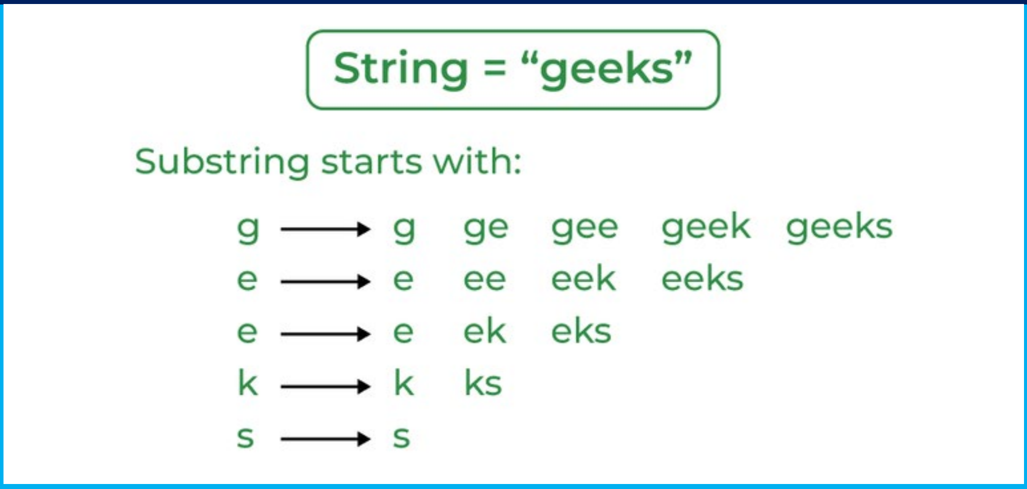
- This method does not work to search for a character.
- This method does not find an index of string if it is not present.

V. Substring in Java

In Java, Substring is a part of a String or can be said subset of the String. There are **two** variants of the substring() method. This article depicts all of them, as follows :

- **public String substring(int startIndex)**
- **public String substring(int startIndex, int endIndex)**

Strings



Java Substring

1. String substring()

The substring() method has two variants and **returns a new string** that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string. Endindex of the substring starts from 1 and not from 0.

Syntax

public String **substring**(int *begIndex*);

Parameters

- **begIndex:** the begin index, inclusive.

Return Value

- The specified substring.

Example of String substring() Method

- Java

```
// Java code to demonstrate the
// working of substring(int begIndex)
public class Substr1 {
    public static void main(String args[])
    {

        // Initializing String
        String Str = new String("Welcome to geeksforgeeks");

        // using substring() to extract substring
        // returns (whiteSpace)geeksforgeeks

        System.out.print("The extracted substring is : ");
        System.out.println(Str.substring(10));
    }
}
```

Output

```
The extracted substring is :  geeksforgeeks
```

2. String substring(begIndex, endIndex)

This method has two variants and **returns** a **new string** that is a substring of this string. The substring begins with the character at the specified index and **extends** to the end of this string or up **to endIndex – 1** if the second argument is given.

Syntax

public String **substring**(int *begIndex*, int *endIndex*);

Parameters

- **beginIndex :** the begin index, inclusive.
- **endIndex :** the end index, exclusive.

Return Value

- The specified substring.

Example

- Java

```
// Java code to demonstrate the
// working of substring(int begIndex, int endIndex)

// Driver Class
public class Substr2 {
    // main function
    public static void main(String args[])
    {
        // Initializing String
        String Str = new String("Welcome to geeksforgeeks");

        // using substring() to extract substring
        // returns geeks
        System.out.print("The extracted substring  is : ");
        System.out.println(Str.substring(10, 16));
    }
}
```

Output

```
The extracted substring  is :  geeks
```

The complexity of the above method

Time Complexity: *O(n), where n is the length of the original string. The substring() method takes constant time O(1) to return the substring.*

Space Complexity: *O(1), as no extra space is required to perform the substring operation.*

Possible Application

The substring extraction finds its use in many applications including prefix and suffix extraction. For example to **extract a Lastname from** the **name** or **extract only the denomination** from a string containing both the amount and currency symbol. The latter one is explained below.

Below is the implementation of the above application

- Java

```
// Java code to demonstrate the
// application of substring()
```

Strings

```
// Driver Class
public class Appli {
    // main function
    public static void main(String args[])
    {
        // Initializing String
        String Str = new String("Rs 1000");

        // Printing original string
        System.out.print("The original string  is : ");
        System.out.println(Str);

        // using substring() to extract substring
        // returns 1000
        System.out.print("The extracted substring  is : ");
        System.out.println(Str.substring(3));
    }
}
```

Output

```
The original string  is : Rs 1000
The extracted substring  is : 1000
```

- Ways to take string input.

1. Using .next()

   It is used to take the input of next token available.

   ```
   String str = s.next();
   System.out.print(str+" "+str.length());
   ```

   Output :

   ```
   Sanskar gupta
   Sanskar 7
   ```

   Note : If we take input again then Gupta will be stored in it. It consider delimiter to take input(space, tab or enter)

2. Using .nextLine()

   It is used to take input of all value of the string in the line until it find delimiter which is enter.

```
String str = s.next();

System.out.println(str+" "+str.length());

String str1 = s.nextLine();

System.out.print(str1+" "+str1.length());
```

Output:

```
I am bad boy
I 1
 am bad boy 11
```

Note: If we want to read from a file then we can pass filename instead of System.in