Java String Manipulation

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	char
codePointAt()	Returns the Unicode of the character at the specified index	int
codePointBefore()	Returns the Unicode of the character before the specified index	int
codePointCount()	Returns the number of Unicode values found in a string.	int
compareTo()	Compares two strings lexicographically	int
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences	int
concat()	Appends a string to the end of another string	String
contains()	Checks whether a string contains a sequence of characters	boolean
contentEquals()	Checks whether a string contains the exact same sequence of characters of the specified CharSequence or StringBuffer	boolean
copyValueOf()	Returns a String that represents the characters of the character array	String
endsWith()	Checks whether a string ends with the specified character(s)	boolean
equals()	Compares two strings. Returns true if the strings are equal, and false if not	boolean
equalsIgnoreCase()	Compares two strings, ignoring case considerations	boolean
format()	Returns a formatted string using the specified locale, format string, and arguments	String
getBytes()	Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array	byte[]
getChars()	Copies characters from a string to an array of chars	void
hashCode()	Returns the hash code of a string	int

Method	Description	Return Type
indexOf()	Returns the position of the first found occurrence of specified characters in a string	int
intern()	Returns the canonical representation for the string object	String
isEmpty()	Checks whether a string is empty or not	boolean
lastIndexOf()	Returns the position of the last found occurrence of specified characters in a string	int
length()	Returns the length of a specified string	int
matches()	Searches a string for a match against a regular expression, and returns the matches	boolean
offsetByCodePoints()	Returns the index within this String that is offset from the given index by codePointOffset code points	int
regionMatches()	Tests if two string regions are equal	boolean
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String
replaceFirst()	Replaces the first occurrence of a substring that matches the given regular expression with the given replacement	String
replaceAll()	Replaces each substring of this string that matches the given regular expression with the given replacement	String
split()	Splits a string into an array of substrings	String[]
startsWith()	Checks whether a string starts with specified characters	boolean
subSequence()	Returns a new character sequence that is a subsequence of this sequence	CharSequence
substring()	Returns a new string which is the substring of a specified string	String
toCharArray()	Converts this string to a new character array	char[]
toLowerCase()	Converts a string to lower case letters	String
toString()	Returns the value of a String object	String
toUpperCase()	Converts a string to upper case letters	String

Method	Description	Return Type
trim()	Removes whitespace from both ends of a string	String
valueOf()	Returns the string representation of the specified value	String

String charAt() Method:

The charAt() method in Java is a method of the String class and is used to retrieve the character at a specified index in a string.

> 💡 **Note:StringIndexOutOfBoundsException** is given when the given specified index number is equal to this string length or the specified given index number is greater, or it is a
> negative number. The first char value is at index 0

```
public class Main {
  public static void main(String[] args) {
    String myStr = "Hello Debasish";
// returns character at index 6
    char result = myStr.charAt(6);
    System.out.println(result);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    String myStr = "Hello Debasish";
// returns character at index 15
    char result = myStr.charAt(15);
    System.out.println(result);
  }
}
```

String compareTo() Method

The main work of java string class compareTo() is to compare the string with the comment string lexicographically. It returns a positive number, negative number, or 0.

Note: The given string and the current string is compared on the basis of the Unicode value of each character present in the string.

💡 Returns
1. The outcome is a negative integer if the first string's lexical length is shorter than the second string's
2. The outcome is a Positive integer if the first string is lexicographically longer than the second string.

💡 3. The result is zero if it is lexically equal to the second string.If it is impossible to compare this item to the specified object, a **ClassCastException** is thrown.If the specified object is null, a **NullPointerException** is thrown.

```
public class Main {
  public static void main(String[] args) {
    String myStr1 = "Debasish";
    String myStr2 = "Debasish";
    System.out.println(myStr1.compareTo(myStr2));
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    String myStr1 = "PrepInsta";
    String myStr2 = "PrepInsta Prime";
    System.out.println(myStr1.compareTo(myStr2)); // Returns negative value
  }
}
```

String compareToIgnoreCase() Method:

The main work of java string class compareToIgnoreCase() is to compare the string with the current string lexicographically ignoring case considerations. A negative integer, zero, or a positive integer as the specified String is greater than, equal to, or less than this String.

- The method compareToIgnoreCase() belongs to the String class that belong to java.lang package.

Note: The given string and the current string is compared on the basis of the Unicode value of each character present in the string.

💡 **Returns**

1. The outcome is a negative integer if the first string's lexical length is shorter than the second string's

2. The outcome is a Positive integer if the first string is lexicographically longer than the second string.
3. The result is zero if it is lexically equal to the second string.

💡 **Exceptions**

If it is impossible to compare this item to the specified object, a ClassCastException is thrown.
If the specified object is null, a NullPointerException is thrown.

```
public class Main
{
public static void main(String[] args) {
    String myStr1 = "Dev";
    String myStr2 = "dev";
    System.out.println(myStr1.compareToIgnoreCase(myStr2));
  }
}
```

```
public class Main {
    //give some complex exaqmples of compareToIgnoreCase
    public static void main(String[] args) {
        String s1 = "Hello";
        String s2 = "hello";
        String s3 = "Hello";
        String s4 = "Hello World";
        String s5 = "Hello World";
        String s6 = "Hello World";
        String s7 = "Hello World";
        String s8 = "Hello World";
        String s9 = "Hello World";
        String s10 = "Hello World";
```

```
        System.out.println(s1.compareToIgnoreCase(s2));
        System.out.println(s1.compareToIgnoreCase(s3));
        System.out.println(s1.compareToIgnoreCase(s4));
        System.out.println(s1.compareToIgnoreCase(s5));
        System.out.println(s1.compareToIgnoreCase(s6));
        System.out.println(s1.compareToIgnoreCase(s7));
        System.out.println(s1.compareToIgnoreCase(s8));
        System.out.println(s1.compareToIgnoreCase(s9));
        System.out.println(s1.compareToIgnoreCase(s10));
}

}
```

```
public class Main
{
    public static void main(String[] args) {
        String str1 = "Debasish";
        String str2 = "debasish";

        if(str1.compareToIgnoreCase(str2)==0){
            System.out.println("str1 is equal to str2");
        }
        else{
            System.out.println("str1 is not equal to str2");
        }
    }
}
```

String concat() Method:

The Java String class concat() method combines strings and returns a combined string. Multiple strings are concatenated using the Java string concat() function. The combined string is returned after the requested string is appended to the end of the original string using this
function. To merge multiple strings together, we can utilise the
concat() technique.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings
  etc.

💡 **Returns**
A string that represents the concatenation of this object's characters followed by the string argument's characters.

💡 **Note:**
The CONCAT() function
concatenates two strings only. If you want to concatenate more than two strings, you need to apply the CONCAT() function multiple times.

```
public class Main {
  public static void main(String[] args) {
    String firstName = "Debasish";
    String lastName = "Sahoo";
    System.out.println(firstName.concat(lastName));
  }
}
```

```
public class Main {
   public static void main(String args[]) {
       //One way of doing concatenation
       String str1 = "Welcome";
       str1 = str1.concat(" to ");
       str1 = str1.concat(" ITM");
       System.out.println(str1);
   }
}
```

```
public class Main {
    public static void main(String[] args) {
        String str1 = "Hey Everyone, ";
        String str2 = "Welcome to ITM";
        String str3 = " Univers";
        String str4 = str1.concat(str2);
        System.out.println(str4);
        String str5 = str1.concat(str2).concat(str3);
        System.out.println(str5);
    }
}
```

String contains() Method:

The contains() method of the Java String class looks through the string's characters in order. If the series of char values is present in the string, it returns true; otherwise, it returns false.

- The Java String contains() method checks whether a string contains a sequence of characters and Returns true if the characters exist and false if not.
- For example, you may check to see if the substring "PrepInsta" is included in the string "PrepInsta Prime is learning Platform." Java's string contains() function comes in handy in these circumstances.
- **NullPointerException** – if the returned value is null

💡 **Returns**
true if this string contains s, false otherwise

💡 **Note:**
The Java String contains()
method checks whether a string contains a sequence of characters and Returns true if the characters exist and false if not.

```
public class Main {
  public static void main(String[] args) {
    String myStr = "Debasish";
    System.out.println(myStr.contains("sis"));
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    String myStr = "Welcome to ITM Univers";
    System.out.println(myStr.contains("come"));
    System.out.println(myStr.contains("UNI"));
    System.out.println(myStr.contains("it"));
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    String myStr = "Datastructure with Java";
    System.out.println(myStr.contains("jav"));
    System.out.println(myStr.contains("str"));
    System.out.println(myStr.contains("va"));
    System.out.println(myStr.contains("ctu"));
  }
}
```

String contentEquals() Method:

The contentEquals() method in Java is a method of the String class that compares the content of a string to the content of another object. The method returns a boolean value indicating whether the content of the two objects are equal or not.

The String's content can also be compared using the contentEquals() method. Any implementation of the CharSequence interface can be passed as an input to ContentEquals(). Therefore, it is possible to compare a String, StringBuffer, StringBuilder, CharBuffer, or Segment.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings etc.

💡 **Returns**
true if this String represents the same sequence of char values as the specified sequence, false otherwise.

💡 **Note**:
String comparison in Java is done using the equals() and contentEquals() methods of the String class.
.

```
public class Main {
  public static void main(String[] args) {
    String myStr = "ITM University";
    System.out.println(myStr.contentEquals("ITM University"));
```

```
    System.out.println(myStr.contentEquals("ITM "));
    System.out.println(myStr.contentEquals("University"));
  }
}
```

String endsWith() Method:

The Java String endsWith() method is used to know if the string ends with the user-specified substring or not.If the string ends with the specified suffix, it returns true; otherwise, it returns false.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings etc.

💡 **Returns**
true if the character sequence
represented by the argument is a suffix of the character sequence
represented by this object; false otherwise.

💡 **Note:**
That the result will be true
if the argument is the empty string or is equal to this String object as
determined by the equals(Object) method.

```
public class Main {
  public static void main(String[] args) {
    String myStr = "Hello jAVA";
      System.out.println(myStr.endsWith("VA"));
    System.out.println(myStr.endsWith("llo"));
    System.out.println(myStr.endsWith("He"));
  }
}
```

```
public class Main {
    public static void main(String[] args) {
        String message = "Happy New Year 2023";
        System.out.println(message.endsWith("2023"));
        System.out.println(message.endsWith("2022"));
```

```
    }
}
```

String Equals() Method:

The Java String equals() method of the String class compares the contents of the two strings. It returns false if any character is not found to match. It returns true if all characters match.

This string is compared to the given object. The argument must not be null and must be a String object that represents the same set of characters as this object in order for the result to be true.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings etc.

Returns
true if the given object represents a String equivalent to this string, false otherwise

Note:
To check the equality of string contents, We use the equals() function.

```
public class Main {
  public static void main(String[] args) {
    String myStr1 = "Hello";
    String myStr2 = "Hello";
    String myStr3 = "Hello ITM";
    System.out.println(myStr1.equals(myStr2));
    System.out.println(myStr1.equals(myStr3));
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    String myStr1 = "java";
    String myStr2 = "C++";
    String myStr3 = "JAVA";
    System.out.println(myStr1.equals(myStr2));
```

```
    System.out.println(myStr1.equals(myStr3));
  }
}
```

String equalsIgnorecase() Method:

The format() method of a java string returns a formatted string with the specified locale, format, and parameters.

The formatting string creates a String by formatting the supplied inputs.
An argument index, flags, width definition, precision modifier, and conversion characters are all included in a formatting string.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings etc.

Format(), an equivalent class method for the String class, returns a String object as opposed to a PrintStream object.

Returns
A formatted string using the specified format string and arguments.

Note:
If there are more arguments
than format specifiers, the extra arguments are ignored. The number of arguments is variable and may be zero.

```
public class Main{
   public static void main(String args[]){
  String str = "Hello, Notion User";
  String formattedString = String.format("My String is %s", str);
  String formattedString2 = String.format("My String is %.6f",14.140);
  System.out.println(formattedString);
  System.out.println(formattedString2);
   }
}
```

```
public class Main{
   public static void main(String args[]){
  String str1 = "Oracle";
  String str2 = "Java";
  String fstr = String.format("My String is: %1$s, %1$s and %2$s", str1, str2);
  System.out.println(fstr);
   }
}
```

String Format() Method:

The format() method of a java string returns a formatted string with the specified locale, format, and parameters.

The formatting string creates a String by formatting the supplied inputs.
An argument index, flags, width definition, precision modifier, and conversion characters are all included in a formatting string.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings etc.

Format(), an equivalent class method for the String class, returns a String object as opposed to a PrintStream object.

Returns A formatted string using the specified format string and arguments.

Note: If there are more arguments than format specifiers, the extra arguments are ignored. The number of arguments is variable and may be zero.

```
public class Main{
   public static void main(String args[]){
  String str = "Hello, ITM Students";
  String formattedString = String.format("My String is %s", str);
  String formattedString2 = String.format("My String is %.6f",14.140);
  System.out.println(formattedString);
  System.out.println(formattedString2);
```

```
    }
}
```

String Getbytes() Method:

The getBytes method of the Java programming language converts a string into a series of bytes and returns a byte array.

When this string cannot be encoded in the supplied charset, this method's behaviour is undefined. When further control over the encoding process is necessary, the CharsetEncoder class should be utilised.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings etc.

The behavior
of this method when this string cannot be encoded in the default charset is unspecified. The Charset Encoder class should be used when more control over the encoding process is required.

💡 **Returns**
The resultant byte array

💡 **Note:**
The encoding of the string into a series of bytes is performed by the Java String class's
getBytes() function, which stores the result in an array of bytes.

```
public class Main {
    public static void main(String[] args) {
        String str = "Hello World!";
        byte[] bytes = str.getBytes();
        for (byte b : bytes) {
            System.out.print(b + " ");
        }
    }
}
```

String IndexOf() Method:

Java String hashCode() method yields the string's integer hash code value. This function of the Object class is overridden by the String class. A hash code for this Function is returned by the hashCode() method of the Method class. The hash code is calculated as the exclusive-or of the hashcodes for the method name and the class name declared in the underlying method.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in
  Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings etc.

If equals() is true for two strings, their hashCode()
will be the same and If two strings hashCode() is equal, it doesn't mean they are equal.

Returns
A hash code value for this object. The hash code for a String object is computed as s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]

Note:
using int arithmetic, where
s[i] is the ith character of the string, n is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

```
public class Main {
  public static void main(String[] args) {
    String myStr = "Hello Java";
    System.out.println(myStr.hashCode());
  }
}
```

String IndexOf() Method:

The Java String indexOf() method is used to get the index of the first instance of a condition that is stated in the method's parameters. The position of the first occurrence of the provided character(s) in a string is returned by the indexOf() function.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings etc.

The index of the first occurrence of the provided substring is returned by the indexOf() function of the Java StringBuffer class.

Returns
The index of the first occurrence of the character in the character sequence represented by this object, or -1 if the character does not occur.

Note:
If a character with value ch occurs in the character sequence represented by this String object, then the index of the first such occurrence is returned.

```
public class Main {
  public static void main(String[] args) {
    String myStr = "Hello Java.";
    System.out.println(myStr.indexOf("a", 10));
  }
}
```

String Intern() Method:

The Java String class intern() method returns the interned string. It gives back the string's canonical representation.
If the string was produced with a new keyword, it can be used to return

it from memory. In the String Constant Pool, it duplicates the heap string object exactly.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings etc.

Every distinct String value is only stored once in Java's string pool. Reusing String objects enables memory savings while running programmes. It follows that for any two strings s and t, s.intern() == t.intern() is true if and only if s.equals(t) is true.

💡 Returns
A string that has the same contents as this string, but is guaranteed to be from a pool of unique strings.

💡 Note:
It follows that for any two strings s and t, s.intern() == t.intern() is true if and only if s.equals(t) is true.

```
public class Main {
   public static void main(String args[]) {
      String str1 = new String("Hello Java");
      str1 = str1.intern();
      String str2 = "Hello";
      str2 = str2.intern();
      System.out.println(str1.equals(str2));
   }
}
```

String isEmpty() Method:

The java string isEmpty() method of the String class will tell you whether the String is empty or not.The isEmpty() method of the Java String class determines if the supplied string is empty or not. Keep in mind that empty here denotes a string that has exactly zero characters.

- The java.lang.String class provides a lot of built-in methods that are used to manipulate string in Java.These methods help us to perform operations on String objects such as trimming, concatenating, converting, comparing, replacing strings etc.

The isEmpty() method will throw a NullPointerException if it is used on a null String.

Returns
true if length() is 0, otherwise false

Note:
Since JDK 1.6, the Java String class's isEmpty () method has been a part of Java strings

```
public class Main {
  public static void main(String[] args) {
    String myStr1 = "Hello java";
    String myStr2 = "";
    System.out.println(myStr1.isEmpty());
    System.out.println(myStr2.isEmpty());
  }
}
```

String Join() method.

The String.join() method is a static method that can be used to concatenate a list of strings into a single string, using a specified delimiter.

- **The String.join() method** was introduced in Java 8 and is a convenient way to concatenate strings without having to use a loop or manually append the strings together.
  It can be especially useful when working with large lists of strings.
- **The String.join() method** is a static method in the java.lang.String class that can be used to
  join elements of an array or any Iterable object into a single string,
  using a specified delimiter or separator.

💡 Returns
a new String that is composed of the elements separated by the delimiter

💡 Throws:
NullPointerException - If delimiter or elements is null

```
public class StringJoinExample {
   public static void main(String[] args) {
      String[] words = {"Hello", "World", "Welcome", "to", "Java"};
      String sentence = String.join(" ", words);
      System.out.println(sentence);
   }
}
```