# String Handling

String is a sequence of characters. But, unlike some other languages that implement strings as character arrays, Java implements strings as objects of type String. Implementing strings as built-in objects allows Java to provide a full complement of features that make string handling convenient. For example, Java has methods to compare two strings, search for a substring, concatenate two strings, and change the case of letters within a string. Also, String objects can be constructed a number of ways, making it easy to obtain a string when needed.

When you create a String object, you are creating a string that cannot be changed. That is, once a String object has been created, you cannot change the characters that comprise that string. You can still perform all types of string operations. The difference is that each time you need an altered version of an existing string; a new String object is created that contains the modifications. The original string is left unchanged. For those cases in which a modifiable string is desired, Java provides two options: StringBuffer and StringBuilder. Both hold strings that can be modified after they are created.

To say that the strings within objects of type String are unchangeable means that the contents of the String instance cannot be changed after it has been created. However, a variable declared as a String reference can be changed to point at some other String object at any time.

## The String Constructors:

To create an empty String, call the default constructor. For example:

```
String s = new String();
```

will create an instance of String with no characters in it.

To create a String initialized by an array of characters, use the constructor shown here:

```
String(char chars[])
```

Here is an example:

```
char chars[] = { 'a', 'b', 'c' };
String s = new String(chars);
```

This constructor initializes s with the string "abc".

You can specify a subrange of a character array as an initializer using the following constructor:

String (char chars[ ], int startIndex, int numChars)

Here, startIndex specifies the index at which the subrange begins, and numChars specifies the number of characters to use. Here is an example:

```
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
String s = new String(chars, 2, 3);
```

This initializes s with the characters cde.

```
// Construct string from subset of char array.
class SubStringCons
{
	public static void main (String args[])
	{
		byte ascii [] = {65, 66, 67, 68, 69, 70};
		String s1 = new String(ascii);
		System.out.println(s1);
		String s2 = new String(ascii, 2, 3);
		System.out.println(s2);
	}
}
```

This program generates the following output:

```
ABCDEF
CDE
```

## String Length:

The length of a string is the number of characters that it contains. To obtain this value, call the length( ) method.

```
class strlength
{
	Public static void main (String [] args)
	{
		char chars[] = { 'a', 'b', 'c' };
		String s = new String(chars);
		System.out.println(s.length());
	}
}
```

## Special String Operations:

### 1. String Literals:

```
class literals_len
{
	public static void main (String [] args)
	{
		String s1 = "abc";
		System.out.println("abc".length());
	}
}
```

## 2. String Concatenation:

This allows you to chain together a series of + operations.

```
class concat_example
{
	public static void main (String [] args)
	{
		String age = "9";
		String s = "He is " + age + " years old.";
		System.out.println(s);
	}
}
```

**String Concatenation with Other Data Types:**

```
// This will give same output as above example.
class concat_example
{
	public static void main(String [] args)
	{
		int age = 9;
		String s = "He is " + age + " years old.";
		System.out.println(s);
	}
}
```

But, What about this one:

```
String s = "four: " + 2 + 2;
System.out.println(s);
```

This fragment displays: four: 22

## 3. String Conversion and toString( ):

```
class Box
{
    double width, height, depth;
    Box(double w, double h, double d)
    {
        width = w;    height = h;    depth = d;
    }
    public String toString()
    {
        return "Dimensions are " + width + " by "+  depth + " by " + height;
    }
}
class toStringDemo
{
    public static void main (String
    args[]) {
        Box b = new Box (10, 12, 14);
        String s = "Box b: " + b;                    // concatenate Box object
        System.out.println (b);
        System.out.println(s);
    }
}
```

```
C:\Users\Nimesh\Desktop\stringhandling>java toStringDemo
Dimensions are 10.0 by 14.0 by 12.0
Box b: Dimensions are 10.0 by 14.0 by 12.0
```

## Character Extraction:

- The String class provides a number of ways in which characters can be extracted from a String object.

**1. charAt():**

- To extract a single character from a String

```
class charAtdemo
{
        public static void main (String [] args)
        {
                String s1 = "KCMIT";
                System.out.println("KCMIT".charAt(2));
        }
}
```

**2. getChars():**

- To extract more than one character at a time from a String

***void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)***

```
class getCharsDemo
{
        public static void main (String args[])
        {
                String s = "Example of getChars method";
                int start = 4;     int end = 12;
                char buf[] = new char[end - start];
                s.getChars(start, end, buf, 0);
                System.out.println(buf);
        }
}
```

**3. getBytes( ):**

- character-to-byte conversions

```
class getBytesDemo
{
   public static void main(String arg[])
   {
      String input = new String("BIM");
      try
          {
         byte[] bytes = input.getBytes();
         System.out.print("After encoding : ");
         for (byte b : bytes)
            System.out.print(b + " ");
          }
          catch (Exception e)
          {
         System.out.println("Unsupported character set");
      }
   }
}
```

```
C:\Users\Nimesh\Desktop\stringhandling>java getBytesDemo
After encoding : 66 73 77
```

**4. toCharArray( ):**

- To convert all the characters in a String object into a character array

```
class toCharArrayDemo
{
        public static void main(String args[])
        {
                String Str = new String ("Java Programming");
                System.out.println (Str.toCharArray() );
        }
}
```

Output: Java Programming

## String Comparison:

**1. equals () and equalsIgnoreCase ():**

- To compare two strings for equality, use equals ().
- equals() returns true if the strings contain the same characters in the same order, and false otherwise. The comparison is case-sensitive.
- To perform a comparison that ignores case differences, call equalsIgnoreCase(). When it compares two strings, it considers A-Z to be the same as a-z.

```
class equalsDemo
{
        public static void main (String args[])
        {
                String s1 = "Java",     s2 = "Java",      s3 = "WebTech",     s4 = "JAVA";
                System.out.println (s1.equals(s2));
                System.out.println (s1.equals(s3));
                System.out.println (s1.equals(s4));
                System.out.println (s1.equalsIgnoreCase(s4));
        }
}
```

2. **regionMatches():**
   - compares a specific region inside a string with another specific region in another string

   ***boolean regionMatches(int startIndex, String str2, int str2StartIndex, int numChars)***

```
class regionMatch
{
    public static void main (String args[])
    {
        String Str1 = new String("Welcome to Tutorialspoint.com");
        String Str2 = new String("Tutorials");
        String Str3 = new String("TUTORIALS");
        System.out.println(Str1.regionMatches(11, Str2, 0, 9));
        System.out.println(Str1.regionMatches(12, Str3, 1, 9));
    }
}
```

3. **startsWith( ) and endsWith( ):**
   - startsWith() method determines whether a given String begins with a specified string.
   - endsWith() determines whether the String in question ends with a specified string

   **eg:**

   - "Foobar".startsWith("Foo")
   - "Foobar".endsWith("bar")
     - Both true

"Foobar".startsWith("bar", 3)

returns true.

**4. equals() Versus ==**

- equals () method compares the characters inside a String object.
- The == operator compares two object references to see whether they refer to the same instance.

```
class EqualsNotEqualTo
{
    public static void main (String args[])
    {
        String s1 = "Hello";
        String s2 = new String(s1);
        System.out.println(s1.equals(s2));
        System.out.println(s1 == s2);
    }
}
```

**5. <u>compareTo():</u>**

```
class CompareToExample
{
    public static void main(String args[])
    {
        String s1="hello";
        String s2="hello";
        String s3="meklo";
        String s4="hemlo";
        System.out.println(s1.compareTo(s2));   //0
        System.out.println(s1.compareTo(s3));   //-5
        System.out.println(s1.compareTo(s4));   //-1
    }
}
```

## Modifying a String:

**1. substring ():**

```
class TestSubstring
{
        public static void main(String args[])
        {
                String s="Tribhuwan University";
                System.out.println(s.substring(10));
                System.out.println(s.substring(0,9));
        }
}
```

**Output:**

```
Nimeshs-MacBook-Pro:javaprog nimesh$ javac TestSubstring.java
Nimeshs-MacBook-Pro:javaprog nimesh$ java TestSubstring
 University
 Tribhuwan
```

**2. concat ():**

- You can concatenate two strings using **concat()**
- **concat()** performs the same function as **+**.
- For example:

```
String s1 = "one";
String s2 = s1.concat("two");
```

- puts the string "onetwo" into **s2**. It generates the same result as the following sequence:

```
String s1 = "one";
String s2 = s1 + "two";
```

### 3. replace ():

- The **replace ()** method has two forms:
  (a) The first replaces all occurrences of one character in the invoking string with another character. It has the following general form:

String replace (char *original*, char *replacement*)

- Here, *original* specifies the character to be replaced by the character specified by *replacement*.

For example,

String s = "Hello".replace('l', 'w');

- puts the string "Hewwo" into **s**.

(b) The second form of **replace()** replaces one character sequence with another. It has this general form:

String replace(CharSequence *original*, CharSequence *replacement*)

### 4. trim():

- The **trim( )** method returns a copy of the invoking string from which any leading and trailing whitespace has been removed.

String s = "  Hello World  ".trim();

This puts the string "Hello World" into **s**.

**/* Define String array of size 4 and take name of 4 students. Then display the name of students whose name starts with character t. */**

```
import java.util.Scanner;
class name
{
    public static void main(String[]args)
    {
            int i;
            String a[]=new String[4];
            Scanner scan=new Scanner(System.in);
            System.out.println("Enter the name of students:");
            for(i=0;i<4;i++)
                    a[i]=scan.nextLine();
            for(i=0;i<4;i++)
            {
                    if(a[i].startsWith("t"))
                            System.out.println(a[i]);
            }

    }
}
```