CS20004: Object Oriented Programming using Java

Lec- 21

In this Discussion . . .

- String class
  - String Constructor
  - String Operations
  - String Functions
  - String conversion and toString()
  - Data Conversion using valueOf()
- StringBuffer
  - StringBuffer Constructor
  - StringBuffer Operations
- References

String

- Once a String Object has been created, you cannot change the characters that comprise that string, i.e. ***String is Immutable***
- Whenever we change any string, a new instance is created.
- To solve this, Java provides a companion classes to String called StringBuffer and StringBuilder for creating mutable strings.
- StringBuffer objects can be modified after they are created

String

- In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

```
char ch[] = {'S','o','u','r','a','j','i','t'};
String s = new String(ch);
```

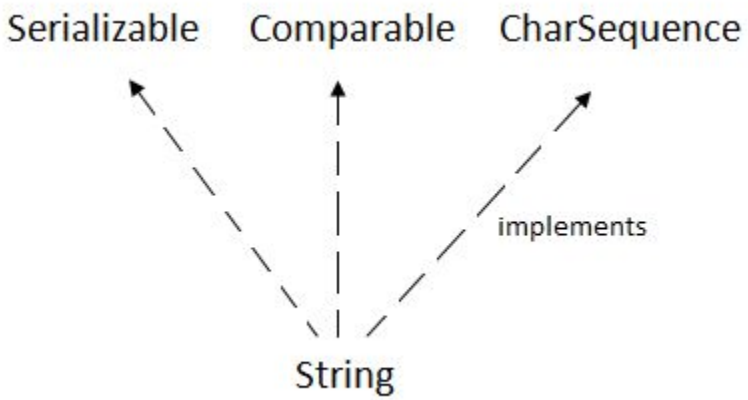
⟺

```
String s = "Sourajit";
```

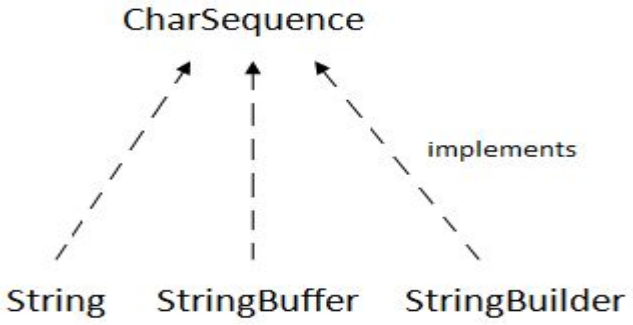
- Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

String

- The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.



- The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in Java by using these three classes.



What is a String in Java?

- Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters.
- The java.lang.String class is used to create a string object.

How to create a string object?

Ans - Two ways

By using the String Literal	By using the new keyword (invoking the String constructor)

Strings creation using String Literals

- Java String literal is created by using double quotes. For Example:

```
String s="welcome";
```

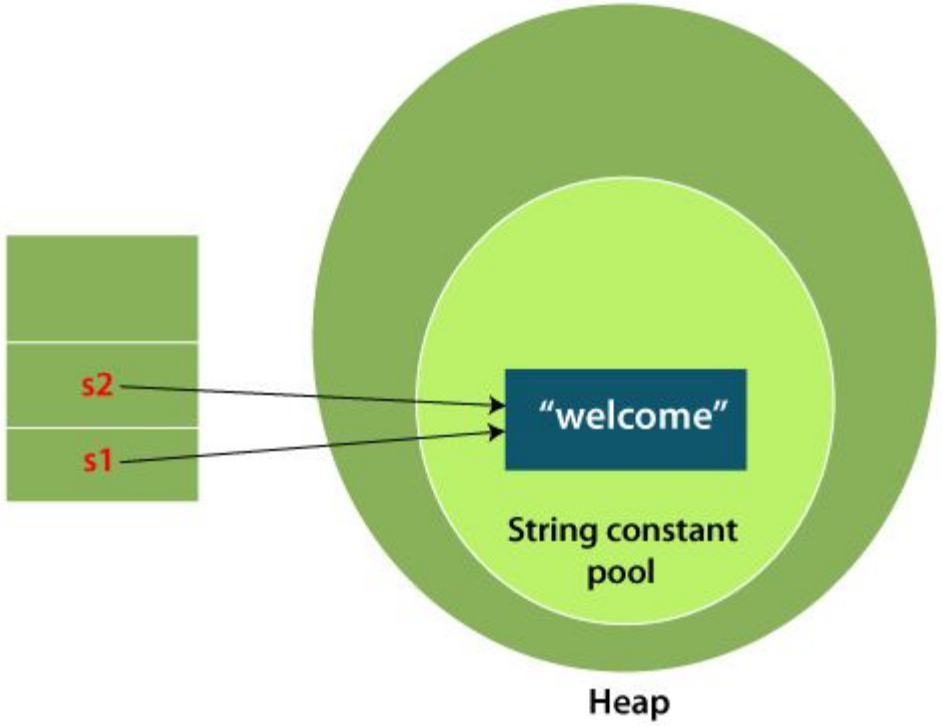
- Each time you create a string literal:
  - the JVM checks the "string constant pool" first.
  - If the string already exists in the pool, a reference to the pooled instance is returned.
  - If the string doesn't exist in the pool, a new string instance is created and placed in the pool.

Strings creation using String Literals

- For example:

```
String s1="Welcome";
String s2="Welcome";//It doesn't create a new instance
```

- In this example, only one object will be created.

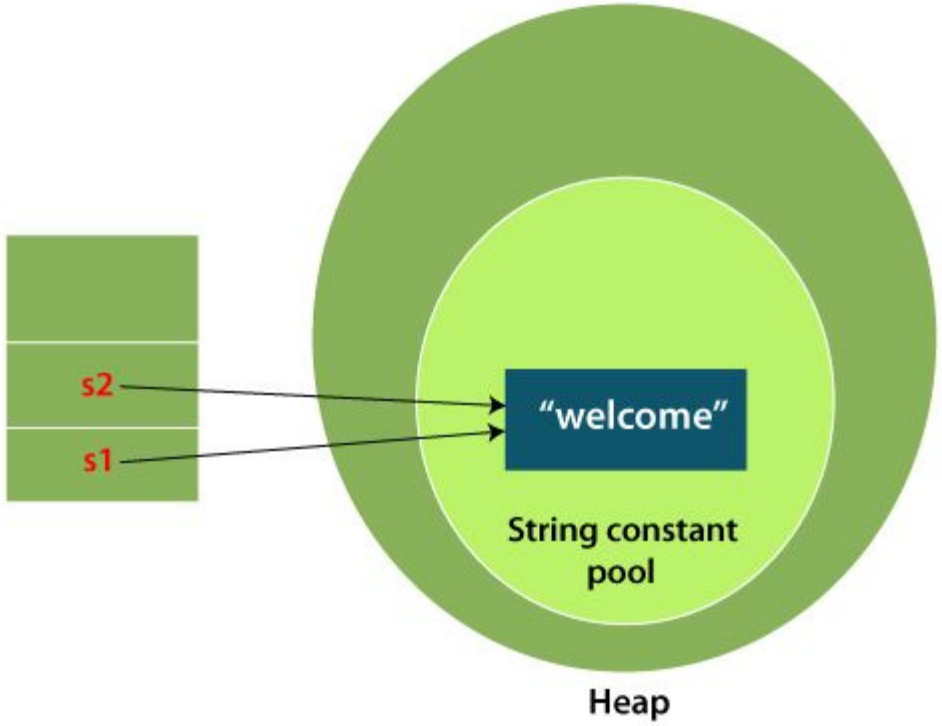


Strings creation using String Literals

- For example:

```
String s1="Welcome";
String s2="Welcome";//It doesn't create a new instance
```

- In this example, only one object will be created.
- Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object.

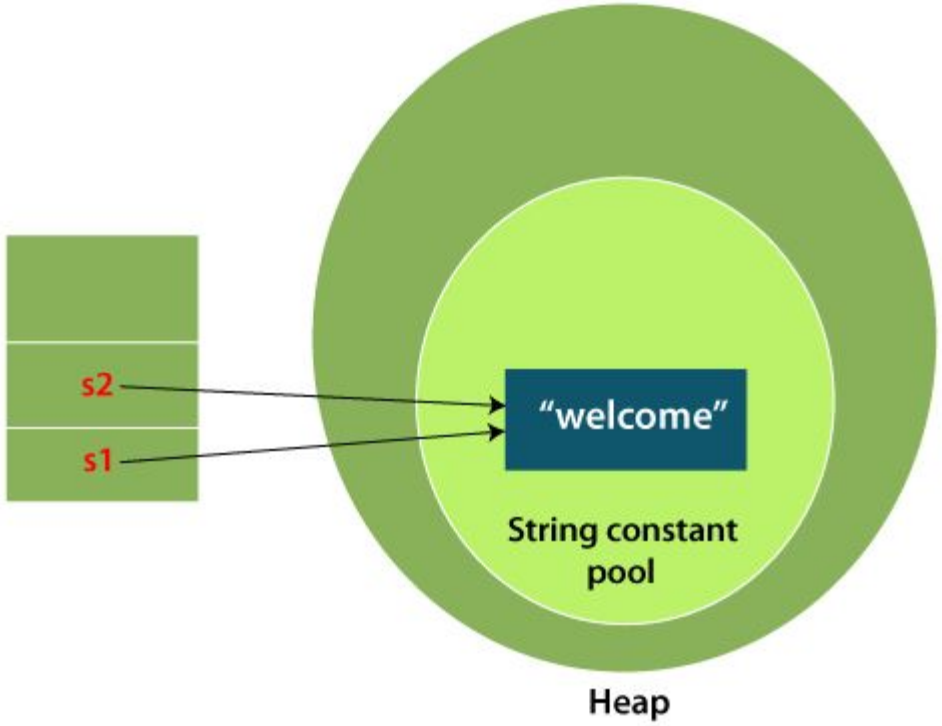


Strings creation using String Literals

- For example:

```
String s1="Welcome";
String s2="Welcome";//It doesn't create a new instance
```

- In this example, only one object will be created.
- Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object.
- After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.



Why use String Literals?

- To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).
- **Note:** String objects are stored in a special memory area known as the "string constant pool".

Strings creation using new Keyword (String Constructor)

- Creating an empty String:

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

- Creating a string that has initial values:

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

For Ex-

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

Strings creation using new Keyword (String Constructor)

- Creating an empty String:

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

In such case, JVM will create a new string object in normal (non-pool) heap memory, and any literal if passed as parameter will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

- Creating a string that has initial values:

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

For Ex-

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

Strings creation using new Keyword (String Constructor)

- Creating a string as a subrange of a character array:

```
String(char chars[ ], int startindex, int numchars)
```

For Ex-

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

- Constructing a String object that contains the same character sequence as another String object:

```
String(String obj)
```

For Ex-

```
char c[ ] = {'J', 'a', 'v', 'a'};
String s1 = new String(c);
String s2 = new String(s1);
```

Java String class Example

```
public class Stringexp
{
        public static void main(String args[])
        {
                String s1="java";
                char ch[ ]={'s','t','r','i','n','g','s'};
                String s2=new String(ch);
                String s3=new String("example");
System.out.println(s1);
                System.out.println(s2);
                System.out.println(s3);
        }
}
```

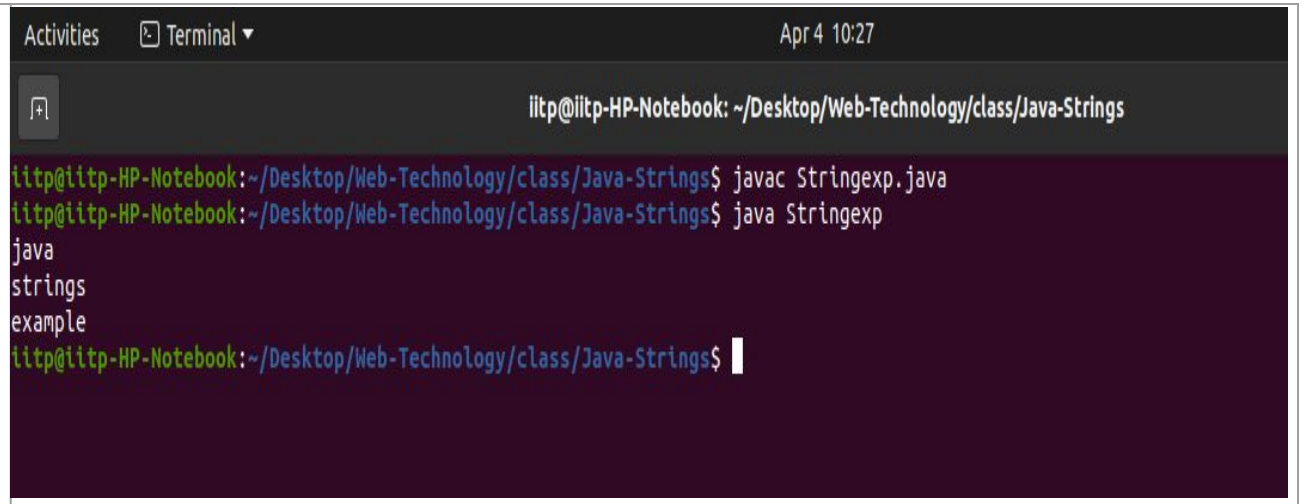
creating string by Java string literal

converting char array to string

creating Java string by new keyword

Java String class Example

```
public class Stringexp
{
        public static void main(String args[])
        {
                String s1="java";
                char ch[ ]={'s','t','r','i','n','g','s'};
                String s2=new String(ch);
                String s3=new String("example");
                System.out.println(s1);
                System.out.println(s2);
                System.out.println(s3);
        }
}
```



The above code, converts a **char** array into a **String** object. And displays the String objects **s1, s2,** and **s3** on console using println() method.

String operations

- String Literals:
  - For each String literal, Java automatically constructs a String object

```
char chars[ ] = {’a’,’b’,’c’};
String s1 = new String(chars);
```

  - Using String literals

```
String s2 = äbc";
```

  - String Concatenation:

```
String age = "19";
String s = "He is "+ age + "years old.";
```

String Operations Example

```
class Concatstringexp
{
	public static void main(String args[])
	{
		String s="Sachin"+" Tendulkar";
		System.out.println(s);
		String s=50+30+"John"+40+40;
		System.out.println(s);
		String s1="John ";
		String s2="Abraham";
		String s3=s1.concat(s2);
		System.out.println(s3);
	}
}
```

String Operations Example

```
class Concatstringexp
{
	public static void main(String args[])
	{
		String s="Sachin"+" Tendulkar";
		System.out.println(s);
		String s=50+30+"John"+40+40;
		System.out.println(s);
		String s1="John ";
		String s2="Abraham";
		String s3=s1.concat(s2);
		System.out.println(s3);
	}
}
```

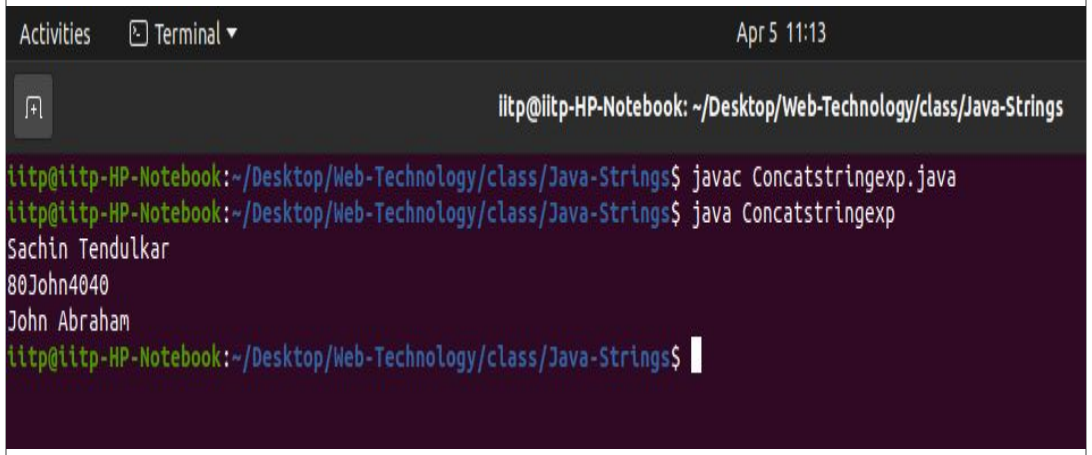
```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Strings$ javac Concatstringexp.java
Concatstringexp.java:7: error: variable s is already defined in method main(String[])
                String s=50+30+"John"+40+40;
                       ^
1 error
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Strings$
```

String Operations Example

```
class Concatstringexp
{
	public static void main(String args[])
	{
		String s="Sachin"+" Tendulkar";
		System.out.println(s);
		String s1=50+30+"John"+40+40;
		System.out.println(s1);
		String s2="John ";
		String s3="Abraham";
		String s4=s2.concat(s3);
		System.out.println(s4);
	}
}
```

String Operations Example

```
class Concatstringexp
{
	public static void main(String args[])
	{
		String s="Sachin"+" Tendulkar";
		System.out.println(s);
		String s1=50+30+"John"+40+40;
		System.out.println(s1);
		String s2="John ";
		String s3="Abraham";
		String s4=s2.concat(s3);
		System.out.println(s4);
	}
}
```



After a string literal, all the + will be treated as string concatenation operator.

String Functions

- String s = "Welcome to demo program";
  - **length():**

```
int len = s.length();
```

  - **charAt(n):**

```
String fruit = "banana";
char ch = fruit.charAt(1);
System.out.println(ch);
```

  - **getChars(n1, n2, s, n3):**

```
int start = 4, end = 8; destoffset;
char buf[] = new char[end - start];
s.getChars(start, end, buf, destoffset);
```

String Functions

- **equals(s) :** Returns true if the strings contain the same characters; otherwise false

```
String name1 = "GOOD";
String name2 = "GoOD";
if(name1.equals(name2))
{
        System.out.println("The names are the same");
}
```

- **equalsIgnoreCase(s):** Similar to equals() by ignoring the cases

```
if(name1.equalsIgnoreCase(name2))
{
        System.out.println("The names are same");
}
```

String Functions

- **compareTo() :** Returns the difference between the first characters in the strings that differ

Note:- It throws the following two exceptions:

ClassCastException: If this object cannot get compared with the specified object.

NullPointerException: If the specified object is null.

```
int flag = name1.compareTo(name2);
if(flag == 0)
{
        System.out.println("The names are same");
}
```

- **indexOf(c):** searches the first occurrence of a character

```
String fruit="banana";
int index = fruit.indexOf('a');
```

```
public class Equfuncexp
{
	public static void main(String args[])
	{
		String s1="sourajit";
		String s2="sourajit";
		String s3="SOURajit";
		String s4="behera";
		String s5="behera";
		System.out.println(s1.equals(s2));//true because content and case is same
		System.out.println(s1.equals(s3));//false because case is not same
		System.out.println(s1.equalsIgnoreCase(s4));//false because content is not same
		System.out.println(s2.equalsIgnoreCase(s3));//true because contents are same
		System.out.println(s2.compareTo(s5));//true because contents are same
	}
}
```

```
public class Equfuncexp
{
        public static void main(String args[])
        {
                String s1="sourajit";
                String s2="sourajit";
                String s3="SOURajit";
                String s4="behera";
                String s5="behera";
                System.out.println(s1.equals(s2));
                System.out.println(s1.equals(s3));
                System.out.println(s1.equalsIgnoreCase(s4));
System.out.println(s2.equalsIgnoreCase(s3));
System.out.println(s2.compareTo(s5));    }
}
```

true because content and case is same
false because case is not same
false because content is not same
true because contents are same

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.

compareTo() method compares the given string with the current string lexicographically. It returns a positive number, negative number, or 0.

It compares strings on the basis of the Unicode value of each character in the strings.

String Functions

- **lastIndexOf(c):** searches the last occurrence of a character

```
String fruit="banana";
int index = fruit.lastIndexOf('a');
```

- **indexOf(c, n):** used to specify a starting point for the search

```
String fruit = "banana";
int index = fruit.indexOf(’a’, 2);
```

String Functions

- **lastIndexOf(c,n):** returns last index position for the given char value and from index

```
String fruit="banana";
int index = fruit.lastIndexOf('a',2);
```

- **substring():** Extracts a substring

```
String org = "Welcome to Java";
String result = null;
result = org.substring(2, 6);
```

String Functions

- **concat():** combines a specified string at the end of the first string. It returns a combined string. It is like appending another string.

```
String s1 =”Ram";
String s2 = s1.concat("Hari");
```

or

```
String s2 = s1 + "Hari";
```

- **replace():** Replaces all occurrences of one character in the invoking string with another character

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

String Functions

- **trim():** Returns a copy of the involving string from which any leading and trailing whitespace has been removed

```
String s = "Hello world ".trim();
```

- **toUpperCase():** returns the string in uppercase letter, i.e., it converts all characters of the string into upper case letter.

```
String s = "Welcome to test.";
String upper = s.toUpperCase();
```

String Functions

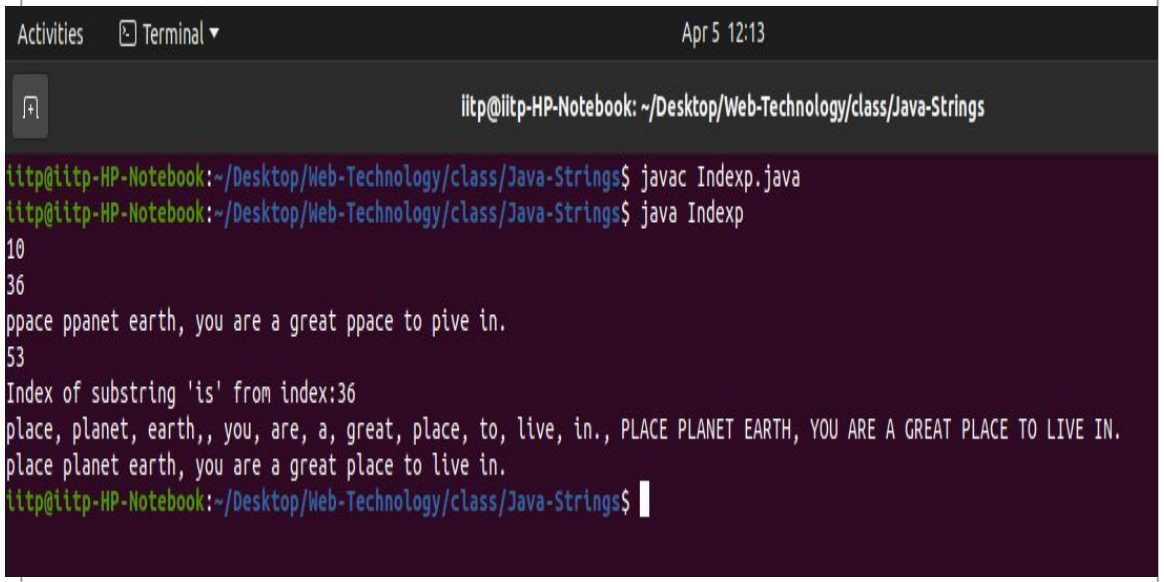
- **toLowerCase():** converts a string to lowercase letters.

```
String lower = s.toLowerCase();
```

- ***Non-alphabetical characters, such as digits are unaffected***

```
public class Indexp
{
	public static void main(String[] args)
	{
		String myStr = "place planet earth, you are a great place to live in.";
		System.out.println(myStr.indexOf("e", 5));
		System.out.println(myStr.lastIndexOf("place"));
		System.out.println(myStr.replace('l', 'p'));
		System.out.println(myStr.length());
		System.out.println("Index of substring 'is' from index:" + myStr.indexOf("place", 5));
		String[] result = myStr.split(" ");
		for (String str : result)
		{
			System.out.print(str + ", ");
		}
		System.out.println(myStr.toUpperCase());
		System.out.println(myStr.toLowerCase());
  }
}
```

```java
public class Indexp
{
	public static void main(String[] args)
	{
		String myStr = "place planet earth, you are a great place to live in.";
		System.out.println(myStr.indexOf("e", 5));
		System.out.println(myStr.lastIndexOf("place"));
		System.out.println(myStr.replace('l', 'p'));
		System.out.println(myStr.length());
		System.out.println("Index of substring 'is' from index:" + myStr.indexOf("place", 5));
		String[] result = myStr.split(" ");
		for (String str : result)
		{
			System.out.print(str + ", ");
		}
		System.out.println(myStr.toUpperCase());
		System.out.println(myStr.toLowerCase());
  }
}
```



String conversion and toString()

- If we want to represent any object as a string, toString() is used. This method returns the string representation of the object.

- If we print any object, java compiler internally invokes toString() method on the object. Overriding toString() method returns the desired output.

- By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code

String conversion and toString()

- By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code

```
public String toString()
{
        return "Dimensions: "+ height + ", "+ width;
}
```

Data Conversion using valueOf()

- It converts different types of values into string. It is a static method that is overloaded within String for all built-in types

Example Syntaxes-	Example usage-
public static String valueOf(boolean b) public static String valueOf(char c) public static String valueOf(int i) public static String valueOf(double d)	int data=30; String str=String.valueOf(data); System.out.println(str+40);

```
class Tostrvalof
{
    public static void main(String args[])
    {
        String str = null;
        Integer x = 45;
        System.out.println(str+String.valueOf(x));
        System.out.println(x.toString());
        System.out.println(Integer.toString(12));
    }
}
```

```
class Tostrvalof
{
	public static void main(String args[])
	{
		String str = null;
		Integer x = 45;
		System.out.println(str+String.valueOf(x));
		System.out.println(x.toString());
		System.out.println(Integer.toString(12));
	}
}
```

StringBuffer

- StringBuffer is a peer class of String that provides much of the functionality of Strings
- String is immutable. StringBuffer is mutable. It represents growable and writable character sequence
- StringBuffer may have characters and substring inserted in the middle or appended to the end
- It will automatically grow to make room for such additions

StringBuffer Constructor

- Creating an empty StringBuffer

```
StringBuffer sb = new StringBuffer();
```

- Creating size-defined StringBuffer

```
StringBuffer sb = new StringBuffer(int size);

Ex- StringBuffer sb = new StringBuffer(50);
```

- Creating String object - based StringBuffer

```
StringBuffer sb = new StringBuffer(String str);
Ex- StringBuffer sb = new StringBuffer("Hello");
```

StringBuffer Functions

- **length():** finds the current length

```
StringBuffer sb=new StringBuffer("Hello");
int len=sb.length();
```

- **capacity():** finds the total allocated capacity

```
StringBuffer sb=new StringBuffer("Hello");
int cap=sb.capacity();
```

- **ensureCapacity():** sets the size of the buffer after a StringBuffer has been constructed

```
void ensureCapacity(int capacity)
Where, capacity specifies the minimum size of the buffer
```

StringBuffer Functions

- **setLength():** sets the length of the buffer within a StringBuffer object

```
void setLength(int len)
```

len specifies the length of the buffer

```
Ex- sb.setLength(4);
```

- **charAt():** obtains the value of a single character

```
char charAt(int pos)
```

pos specifies the index of the character being obtained

```
Ex- StringBuffer sb=new StringBuffer("Hello");
System.out.println(Character: "+sb.charAt(2));
```

- **setCharAt():** sets the value of a character

```
void setCharAt(int pos, char ch)
```

pos specifies the index of the character being obtained

```
Ex- StringBuffer sb=new StringBuffer("Hello");
sb.setCharAt(2,'i');
```

StringBuffer Functions

- **getChars():** copies a substring of a StringBuffer into an array

```
void getChars (int begin, int end, char[] target, int targetbeg)
```

- **append():** concatenates the string representation of any type of data to the end of the invoking StringBuffer object

```
StringBuffer append(Object ob)
StringBuffer append(String str)
StringBuffer append(int num)
```

String.valueOf() is called for each parameter to obtain its string representation

```
String s=null;
int a=100;
StringBuffer sb=new StringBuffer (40);
s=sb.append(ä=").append(a).append("!").
toString();
```

StringBuffer Functions

- **insert():** inserts one string into another

StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
StringBuffer insert(int index, Object ob)

index specifies the index at which the string will be inserted

Ex-
```
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(7,"ok");
```

- **reverse():** reverses the characters in a StringBuffer object

StringBuffer reverse()
Ex-
```
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
```

StringBuffer Functions

- **replace():** replaces one set of characters with another set

```
StringBuffer replace(int start, int end, String str)
Ex-
StringBuffer sb=new StringBuffer("Bhubaneswar");
sb.replace(3,4,"va");
```

- **substring():** returns a portion of a StringBuffer

```
String substring(int start)
String substring(int start, int end)
```

- **delete():** deletes a sequence of characters

```
StringBuffer delete(int start, int end)
Ex- sb.delete(2,4);
```

StringBuffer Functions

- **deleteCharAt():** deletes the character at the pos index

```
StringBuffer deleteCharAt(int pos)
Ex- sb.deleteCharAt(0);
```

- **indexOf(String str):**

```
int i = sb.indexOf(two");
```

- **indexOf(String str, int start):**

```
int i = sb.indexOf(two", 4);
```

- **lastIndexOf(String str):**

```
int i = sb.lastIndexOf(two");
```

StringBuffer Functions

- **lastIndexOf(String str, int start):**

```
int i = sb.lastIndexOf(two",4);
```

- **trimToSize(int size):**

```
sb.trimToSize(10);
```

```
class Stringbufexp
{
	public static void main(String args[])
	{
		StringBuffer sb=new StringBuffer("Hello ");
		sb.append("Java");
		System.out.println(sb);
		sb.insert(1,"Java");
		System.out.println(sb);
		sb.replace(1,3,"Java");
		System.out.println(sb);
		sb.delete(1,3);
		System.out.println(sb);
		sb.reverse();
		System.out.println(sb);//prints olleH
	}
}
```

```
class Stringbufexp
{
	public static void main(String args[])
	{
		StringBuffer sb=new StringBuffer("Hello ");
		sb.append("Java");
		System.out.println(sb);
		sb.insert(1,"Java");
		System.out.println(sb);
		sb.replace(1,3,"Java");
		System.out.println(sb);
		sb.delete(1,3);
		System.out.println(sb);
		sb.reverse();
		System.out.println(sb);//prints olleH
	}
}
```

References

1. https://www.w3schools.com/java/java_ref_string.asp
2. https://www.javatpoint.com/java-string
3. https://www.baeldung.com/java-tostring-valueof
4. https://www.scaler.com/topics/java/stringbuffer-in-java/
5. https://coderanch.com/t/508160/java/Difference-valueOf-Method-toString-Method
6. https://www.javatpoint.com/StringBuffer-class
7.