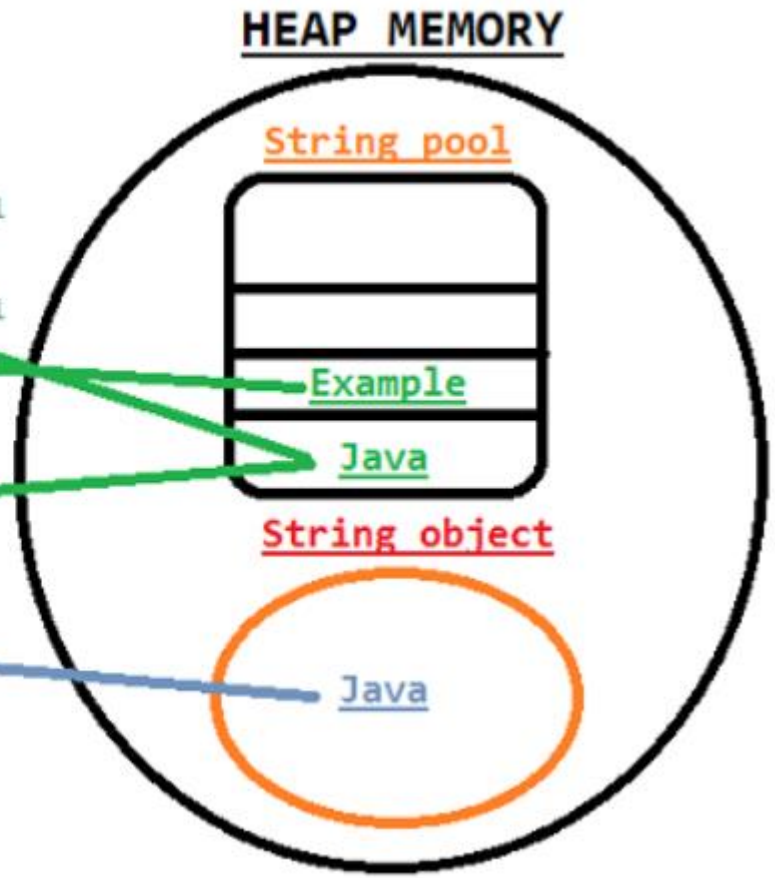
Strings in Java

String Class

Java – String Class and its Methods

- String is probably the most commonly used class in java library. String class is encapsulated under java.lang package. In java, every string that you create is actually an object of type String. One important thing to notice about string object is that string objects are immutable that means once a string object is created it cannot be altered.
- **What is an Immutable Object?**
  - An object whose state cannot be changed after it is created is known as an Immutable object. String, Integer, Byte, Short, Float, Double and all other wrapper class's objects are immutable.

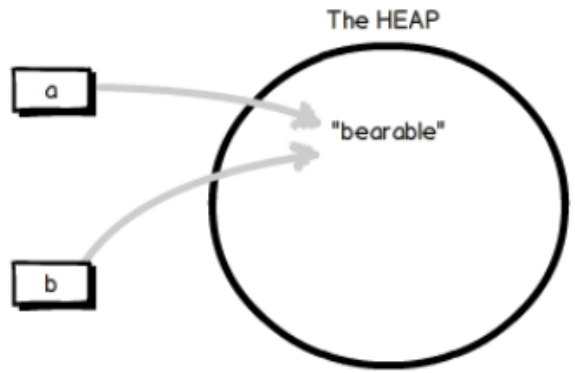
```java
public class String_Example {

  public static void main(String[] args) {

        // this line will create new String in String pool
        String stringPool1 = "Java";

        // this line will create new String in String pool
      String stringPool2 = "Example";

      // Firstly check String pool Java is already there
      // Not create new string object
      // Return refference for existing Java string object
      String stringPool3 = "Java";

      // Not check String pool
      // Create a new object in Heap memory
      String stringObject = new String("Java");

  }

}
```



ngs.

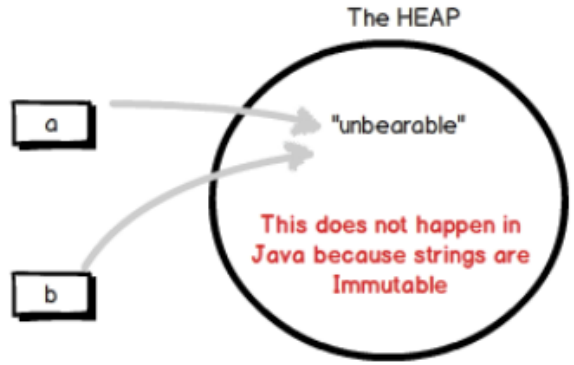
String a = "bearable";

String b = a;



hem

String b = "unbearable";

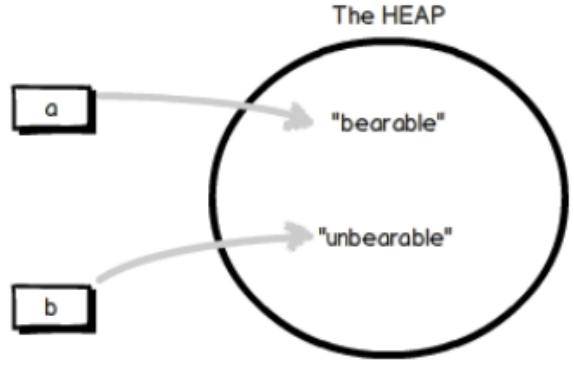


If Strings were NOT immutable, another reference to your string object could change the value of that object.

You don't want that, so "Homey don't play dat!"

real

String b = "unbearable";



This is what actually happens.

Java – String Class and its Methods

- **Creating a String**
- There are two ways to create a String in Java
  - String literal
  - Using new keyword

Java – String Class and its Methods

- **String literal**
- In java, Strings can be created like this: Assigning a String literal to a String instance:

```
String str1 = "Welcome";
String str2 = "Welcome";
```

- **The problem With this Approach**: As I stated in the beginning that String is an object in Java. However we have not created any string object using new keyword above. The compiler does that task for us it creates a string object having the string literal (that we have provided , in this case it is “Welcome”) and assigns it to the provided string instances.

Java – String Class and its Methods

- **But** if the object already exist in the memory it does not create a new Object rather it assigns the same old object to the new instance, that means even though we have two string instances above(str1 and str2) compiler only created on string object (having the value “Welcome”) and assigned the same to both the instances. For example there are 10 string instances that have same value, it means that in memory there is only one object having the value and all the 10 string instances would be pointing to the same object.
- What if we want to have two different object with the same string? For that we would need to create strings using **new keyword**.

Java – String Class and its Methods

- **Using New Keyword**
- As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this:

```
String str1 = new String("Welcome");
String str2 = new String("Welcome");
```

- In this case compiler would create two different object in memory having the same text.

Java – String Class and its Methods

- **Using + Operator (Concatenation)**

```
String str4 = str1 + str2;
or,
String str5 = "hello"+"Java";
```

- Each time you create a String literal, the JVM checks the string pool first. If the string literal already exists in the pool, a reference to the pool instance is returned. If string does not exist in the pool, a new string object is created, and is placed in the pool. String objects are stored in a special memory area known as string constant pool inside the heap memory.

Java – String Class and its Methods

- **Concatenating String**
- There are 2 Ways to concatenate two or more string.
    - Using concat() method
    - Using + operator

Java – String Class and its Methods

- **1) Using concat() method**

```
String s = "Hello";
String str = "Java";
String str2 = s.concat(str);
String str1 = "Hello".concat("Java");    //works with string literals too.
```

Java – String Class and its mMethods

- 2) Using + operator

```
String str = "Adil";
String str1 = "Aslam";
String str2 = str + str1;   // or
String str3 = "Adil"+“Aslam";
```

Java – String Class and its Methods

- **String Comparison**
- String comparison can be done in 3 ways.
  - Using equals() method
  - Using == operator
  - By CompareTo() method

Java – String Class and its Methods

- **Using equals() method**
- equals() method compares two strings for equality. Its general syntax is.

boolean equals (Object str)

- It compares the content of the strings. It will return **true** if string matches, else returns **false**.

```
String s = "Help";
String s1 = "Hello";
String s2 = "Hello";
s1.equals(s2);    //true
s.equals(s1) ;   //false
```

Java – String Class and its Methods

- **Using == operator**
- == operator compares two object references to check whether they refer to same instance. This also, will return true on successful match.

```
String s1 = "Java";
String s2 = "Java";
String s3 = new string ("Java");
test(s1 == s2)     //true
test(s1 == s3)      //false
```

Java – String Class and its Methods

- **By compareTo() method**
- compareTo() method compares values and returns an int which tells if the string compared is less than, equal to or greater than other string. Its general syntax is,

```
int compareTo(String str)
```

- To use this function you must implement the Comparable Interface. compareTo() is the only function in Comparable Interface.

```
String s1 = "Adil";
String s2 = "Kashif";
String s3 = "Adil";
s1.compareTo(S2);    //return -1 because s1 < s2
s1.compareTo(S3);    //return 0 because s1 == s3
s2.compareTo(s1);    //return 1 because s2 > s1
```

Java – String Class and its Methods

- **String length() Method**
- This method returns the length of the string. The length is equal to the number of 16-bit Unicode characters in the string.
- **Syntax**

```
int length()
```

- **Parameters**
  - **NA**
- **Return Value**
  - This method returns the length of the sequence of characters represented by this object.

String length() Method Example

```
public class Example {

   public static void main(String[] args) {
     String str1 = " ";
     String str2 = "Welcome";
     String str3 = "My Name is Adil Aslam";

   System.out.println("Length of First String is: "+str1.length());
   System.out.println("Length of Second String is: "+str2.length());
   System.out.println("Length of Third String is: "+str2.length());
   }
}
```

Java – String Class and its Methods

- **String charAt() Method**
- This method returns the character located at the String's specified index. The string indexes start from zero.
- **Syntax**

```
char charAt(int index)
```

- **Parameters**
  - **index** – Index of the character to be returned.
- **Return Value**
  - This method returns a char at the specified index.

String charAt() Method Example

```
public class Example {


   public static void main(String[] args) {
   String str = "Welcome to string handling Lecture";
            char ch1 = str.charAt(0);
            char ch2 = str.charAt(5);
            char ch3 = str.charAt(11);
            char ch4 = str.charAt(20);
            System.out.println("Character at 0 index is: "+ch1);
            System.out.println("Character at 5th index is: "+ch2);
            System.out.println("Character at 11th index is: "+ch3);
            System.out.println("Character at 20th index is: "+ch4);
   }
}
```

Java – String Class and its Methods

- **String toLowerCase() Method**
- toLowerCase() method returns string with all uppercase characters converted to lowercase
- **Syntax**

String toLowerCase())

- **Parameters**
  - **NA**
- **Return Value**
  - It returns the String, converted to lowercase.

String toLowerCase() Method Example

```
public class Example {


    public static void main(String[] args) {
    String str = "My Name is Adil Aslam";
    System.out.println(str.toLowerCase());


    }
}
```

String toUpperCase() Method Example

```
public class Example {


    public static void main(String[] args) {
    String str = "My Name is Adil Aslam";
    System.out.println(str.toUpperCase());


    }
}
```

String replace() Method Example

```
public class Example {


    public static void main(String[] args) {
    String str = "My Name is Adil Aslam";
    System.out.println(str.replace('A', 'a'));
    System.out.println(str.replace('N', 'I'));


    }
}
```

Java – String Class and its Methods

- **String trim() Method**
- This method returns a copy of the string, with leading and trailing whitespace omitted.
- **Syntax**

String trim()

- **Parameters**
  - **NA**
- **Return Value**
  - It returns a copy of this string with leading and trailing white space removed, or this string if it has no leading or trailing white space.

String trim() Method Example

```
public class Example {


   public static void main(String[] args) {
   String str = "  My Name is Adil Aslam";
   System.out.println("String before trim:"+str);
   System.out.println("String after trim: "+str.trim());


   }
}
```

Java – String Class and its Methods

- **String 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.

String compareTo Method Example

```
public class Example {


   public static void main(String[] args) {
     String str1 = "Strings are immutable";
     String str2 = "Strings are immutable";
     String str3 = "Integers are not immutable";
     int result = str1.compareTo( str2 );
     System.out.println(result);
     result = str2.compareTo( str3 );
     System.out.println(result);
     result = str3.compareTo( str1 );
     System.out.println(result);
   }
}
```

Java – String Class and its Methods

- **String compareToIgnoreCase() Method**
- 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.

String compareToIgnoreCase() Method Example

```
public class Example {
   public static void main(String[] args) {
     String str1 = "HELLO";
     String str2 = "Hello";
     String str3 = “SeLL";

     int result = str1.compareToIgnoreCase(str2 );
     System.out.println(result);
     result = str2.compareToIgnoreCase(str3);
     System.out.println(result);
     result = str3.compareToIgnoreCase(str1 );
     System.out.println(result);
   }
}
```

String compareToIgnoreCase() Method Example

```
public class Example {
   public static void main(String[] args) {
     String str1 = "HELLO";
     String str2 = "Hello";
     String str3 = “SeLL";

     int result = str1.compareToIgnoreCase(str2 );
     System.out.println(result);
     result = str2.compareToIgnoreCase(str3);
     System.out.println(result);
     result = str3.compareToIgnoreCase(str1 );
     System.out.println(result);
   }
}
```

Output is:

0

-11

11

Java – String Class and its Methods

- **String substring() Method**
- 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**

```
String substring(int beginIndex)
```

- **Parameters**
  - **beginIndex** – the begin index, inclusive.
- **Return Value**
  - The specified substring.

String substring() Method Example

```
public class Example {


   public static void main(String[] args) {
    String str = "My Name is Adil Aslam";
          System.out.println("Substring starting from index
          3:"+str.substring(3));
          System.out.println("Substring starting from index
          11: "+str.substring(11));


   }
}
```

Java – String Class and its Methods

- **String substring(beginIndex, endIndex) Method**
- 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**

```
String substring(int beginIndex, int endIndex)
```

- **Parameters**
  - **beginIndex** – the begin index, inclusive.
  - **endIndex** – the end index, exclusive.
- **Return Value**
  - The specified substring.

String substring(beginIndex, endIndex) Method

```
public class Example {


    public static void main(String[] args) {
      String str = "My Name is Adil Aslam";
      System.out.println(str.substring(11,21));
      System.out.println(str.substring(3,15));



    }
}
```

Java – String Class and its Methods

- **String indexOf() Method**
- This method returns the index within this string of the first occurrence of the specified character or -1, if the character does not occur.
- **Syntax**

```
int indexOf(int ch )
```

- **Parameters**
  - **ch** – a character.
- **Return Value**
  - See the description.

Java – String Class and its Methods

- **String indexOf(String str) Method**
- This method returns the index within this string of the first occurrence of the specified substring. If it does not occur as a substring, -1 is returned.
- **Syntax**

```
int indexOf(String str)
```

- **Parameters**
  - **str** – a string.
- **Return Value**
  - See the description.

Java – String Class and its Methods

- **String toString() Method**
- This method returns itself a string.
- **Syntax**

```
String toString()
```

- **Parameters**
  - **NA**
- **Return Value**
  - This method returns the string itself.

String toString() Method Example

```
public class Example {


    public static void main(String[] args) {
      String str = "My Name is Adil Aslam";
      System.out.println(str.toString());


    }
}
```

Java – String Class and its Methods

• String startsWith() Method

- This method has two variants and tests if a string starts with the specified prefix beginning a specified index or by default at the beginning.

• Syntax

```
boolean startsWith(String prefix)
```

• Parameters

- **prefix** – the prefix to be matched.

• Return Value

- It returns true if the character sequence represented by the argument is a prefix of the character sequence represented by this string; false otherwise.

String startsWith() Method Example

```
public class Example {


   public static void main(String[] args) {
    String str = "My Name is Adil Aslam";
          System.out.println("Return Value is:
          "+str.startsWith("My"));
          System.out.println("Return Value is:
          "+str.startsWith("Name"));


   }
}
```

Java – String Class and its Methods

- **String endsWith(String suffix) Method**
  - The **java string endsWith()** method checks if this string ends with given suffix. It returns true if this string ends with given suffix else returns false.
- **Syntax**

```
boolean endsWith(String suffix)
```

- **Parameter**
  - suffix : Sequence of character
- **Returns**
  - true or false

String endsWith(String suffix) Method Example

```
public class Example {


   public static void main(String[] args) {
    String str="My Name is Adil Aslam";
    System.out.println(str.endsWith("m"));
    System.out.println(str.endsWith("Aslam"));
    System.out.println(str.endsWith("Adil"));


   }
}
```

Output is:

true
true
false

Java – String Class and its Methods

- **String split() Method**
- This method has two variants and splits this string around matches of the given regular expression.
- **Syntax**

```
String split(String regex)
```

- **Parameters**
  - **regex** – the delimiting regular expression.
- **Return Value**
  - It returns the array of strings computed by splitting this string around matches of the given regular expression.

String split() Method Example

```
public class Example {


   public static void main(String[] args) {
    String str = "My _Name is _Adil Aslam";
    //splits the string based on "_"
    String[] words=str.split("_");
    //using java foreach loop to print elements of string array
    for(String w:words){
    System.out.println(w);
    }
   }
}
```

String split() Method Example(2)

```
public class Example {


   public static void main(String[] args) {
    String str = "My Name is Adil Aslam";
    //splits the string based on "_"
    String[] words=str.split(" ");
    //using java foreach loop to print elements of string array
    for(String w:words){
    System.out.println(w);
    }
   }
}
```

Java – String Class and its Methods

• String split(String regex, int limit) Method

- This method has two variants and splits this string around matches of the given regular expression.

• Syntax

```
String split(String regex, int limit)
```

• Parameters

- **regex** – the delimiting regular expression.
- **limit** – the result threshold, which means how many strings to be returned.

• Return Value

- It returns the array of strings computed by splitting this string around matches of the given regular expression.

String split(String regex, int limit) Method Example

```
public class Example {
   public static void main(String[] args) {
   String s1="welcome to split world";
   System.out.println("Returning Words:");
      for(String w:s1.split(" ",0)) {
   System.out.println(w);
      }
   System.out.println("Returning Words:");
      for(String w:s1.split(" ",1)) {
   System.out.println(w);
      }
   System.out.println("Returning Words:");
      for(String w:s1.split(" ",2)) {
   System.out.println(w);
      }
   }
}
```

Java – String Class and its Methods

- **String matches() Method**
  - This method tells whether or not this string matches the given regular expression. An invocation of this method of the form str.matches(regex) yields exactly the same result as the expression Pattern.matches(regex, str).
- **Syntax**

```
boolean matches(String regex)
```

- **Parameters**
  - **regex** – the regular expression to which this string is to be matched.
- **Return Value**
  - This method returns true if, and only if, this string matches the given regular expression.

String matches() Method Example

```
public class Example {
   public static void main(String[] args) {
   String Str = "My Name is Adil Aslam";

     System.out.print("Return Value :" );
     System.out.println(Str.matches("(.*)Adil Aslam(.*)"));

     System.out.print("Return Value :" );
     System.out.println(Str.matches("Adil Aslam"));

     System.out.print("Return Value :" );
     System.out.println(Str.matches("My(.*)"));

   }
}
```

Output is:

```
Return Value :true
Return Value :false
Return Value :true
```

Java – String Class and its Methods

- **String valueOf() Method**
  - The **java string valueOf()** method converts different types of values into string. By the help of string valueOf() method, you can convert int to string, long to string, boolean to string, character to string, float to string, double to string, object to string and char array to string.
- **Syntax**

```
public static String valueOf(boolean b)
public static String valueOf(char c)
public static String valueOf(char[] c)
public static String valueOf(int i)
public static String valueOf(long l)
public static String valueOf(float f)
public static String valueOf(double d)
public static String valueOf(Object o)
```

- **Returns**
- string representation of given value

String valueOf() Method Example

```
public class Example {
   public static void main(String[] args) {
     double d = 9029399.939;
     boolean b = true;
     long l = 1234567;
     char[] arr = {'a', 'b', 'c', 'd', 'e', 'f','g' };

     System.out.println("Return Value : " + String.valueOf(d) );
     System.out.println("Return Value : " + String.valueOf(b) );
     System.out.println("Return Value : " + String.valueOf(l) );
     System.out.println("Return Value : " + String.valueOf(arr) );

   }
}
```

Java – String Class and its Methods

- **String isEmpty() Method**
  - The **java string isEmpty()** method checks if this string is empty. It returns *true*, if length of string is 0 otherwise *false*.
  - The isEmpty() method of String class is included in java string since JDK 1.6.
- **Syntax**

```
boolean isEmpty()
```

- **Parameters**
  - **NA**
- **Return Value**
  - This method returns true if length() is 0, else false.

Java – String Class and its Methods

- **String contains() Method**
  - The **java string contains()** method searches the sequence of characters in this string. It returns *true* if sequence of char values are found in this string otherwise returns *false*.
- **Syntax**

```
boolean contains(CharSequence sequence)
```

- **Parameters**
  - **sequence** : specifies the sequence of characters to be searched.
- **Return Value**
  - This method returns true if this string contains s, else false.

Java StringBuffer class

StringBuffer class

- Java StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.
- Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.
- String is a peer class of String. While String create strings of fixed length, StringBuffer create Strings of flexible length that can be modified in terms of both length and content. We can insert characters and substrings in the middle of strings, or appended another string to the end, etc.
- **What is Mutable String**
  - A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

StringBuffer class

- Important Constructors of StringBuffer class:

- **StringBuffer()**
  - creates an empty string buffer with the initial capacity of 16.
- **StringBuffer(String str)**
  - creates a string buffer with the specified string.
- **StringBuffer(int capacity)**
  - creates an empty string buffer with the specified capacity as length.
- **StringBuffer(CharSequence seq)**
  - This constructs a string buffer that contains the same characters as the specified CharSequence.

Difference Between String and StringBuffer

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

StringBuffer class-Methods

- **StringBuffer append() Method**
  - The **java.lang.StringBuffer.append(String str)** method appends the specified string to this character sequence.The characters of the String argument are appended, in order, increasing the length of this sequence by the length of the argument. If **str** is null, then the four characters "null" are appended.
- **Declaration**

StringBuffer append(String str)

- **Parameters**
  - **str** -- This is the value of a string.
- **Return Value**
  - This method returns a reference to this object.

StringBuffer append(String str) Method Example

```
public class Example {
    public static void main(String[] args) {
     StringBuffer sb=new StringBuffer("Hello ");
     //now original string is changed
     sb.append("Java ");
     System.out.println(sb);
     //now original string is changed
     sb.append("World");
     System.out.println(sb);
    }
}
```

StringBuffer class-Methods

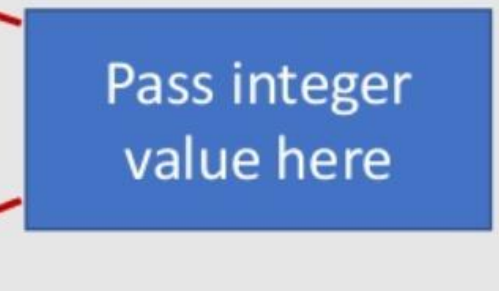
- **StringBuffer append() Method**
  - The **java.lang.StringBuffer.append(int i)** method appends the string representation of the **int** argument to this sequence.
- **Declaration**

```
StringBuffer append(int i)
```

- **Parameters**
  - i -- This is the value of an int.
- **Return Value**
  - This method returns a reference to this object.

StringBuffer append(int i) Method Example

```
public class Example {
    public static void main(String[] args) {
     StringBuffer sb=new StringBuffer("Value is ");
     //now original string is changed
     sb.append(10);
     System.out.println(sb);
     //now original string is changed
     sb.append(10);
     System.out.println(sb);
    }
}
```



StringBuffer class-Methods

• StringBuffer capacity() Method

- The **java.lang.StringBuffer.capacity()** method returns the current capacity.
- The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

• Declaration

```
int capacity()
```

• Parameters

- NA

• Return Value

- This method returns the current capacity.

StringBuffer capacity() Method Example

```
public class Example {


   public static void main(String[] args) {
    StringBuffer sb=new StringBuffer();
    System.out.println(sb.capacity());  //default 16
    sb.append("Hello");
    System.out.println(sb.capacity());   //now 16
    sb.append("java is my favourite language");
    //now (16*2)+2=34 i.e (oldcapacity*2)+2
    System.out.println(sb.capacity());
   }
}
```

StringBuffer class-Methods

- **StringBuffer ensureCapacity() Method**
  - The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.
- **Declaration**

```
void ensureCapacity(int minimumCapacity)
```

- **Parameters**
  - minimumCapacity -- This is the minimum desired capacity.
- **Return Value**
  - This method does not return any value.

StringBuffer ensureCapacity() Method Examp

```
public class Example {
   public static void main(String[] args) {
    StringBuffer sb=new StringBuffer();
    System.out.println(sb.capacity()); //default 16
    sb.append("Hello");
    System.out.println(sb.capacity()); //now 16
    sb.append("java is my favourite language");
    //now (16*2)+2=34 i.e (oldcapacity*2)+2
    System.out.println(sb.capacity());
    sb.ensureCapacity(10); //now no change
    System.out.println(sb.capacity()); //now 34
    sb.ensureCapacity(50);//now (34*2)+2
    System.out.println(sb.capacity()); //now 70
   }
}
```

Output is:

```
16
16
34
34
70
```

StringBuffer class-Methods

- **StringBuffer insert() Method**
- The **java.lang.StringBuffer.insert(int offset, String s)** method inserts the string into this character sequence.
- The characters of the String argument are inserted, in order, into this sequence at the indicated **offset**, moving up any characters originally above that position and increasing the length of this sequence by the length of the argument.
- **Declaration**

```
StringBuffer insert(int offset, String s)
```

- **Parameters**
  - **offset** -- This is the offset.
  - **s** -- This is the value of string.
- **Return Value**
  - This method returns a reference to this object.

StringBuffer insert(int offset String s) Method

```
public class Example {

   public static void main(String[] args) {
    StringBuffer sb=new StringBuffer("Hello ");
    //now original string is changed
    sb.insert(6,"Java");
    System.out.println(sb);
    sb.insert(1,"Mix");
    System.out.println(sb);
   }
}
```

StringBuffer class-Methods

- **StringBuffer insert(int offset, char[] str) Method**
  - The **java.lang.StringBuffer.insert(int offset, char[] str)** method inserts the string representation of the char array argument into this sequence.
  - The characters of the array argument are inserted into the contents of this sequence at the position indicated by **offset**. The length of this sequence increases by the length of the argument.
- **Declaration**

```
StringBuffer insert(int offset, char[] str)
```

- **Parameters**
  - **offset** -- This is the offset.
  - **str** -- This is the character array.
- **Return Value**
  - This method returns a reference to this object.

StringBuffer insert(int offset, char[] str) Method

```
public class Example {
   public static void main(String[] args) {
    StringBuffer sb=new StringBuffer("Hello ");
    char chr1[]= {'J','a','v','a'};
    char chr2[]= {'W','o','r','l','d'};
    sb.insert(6,chr1);
    System.out.println(sb);
    sb.insert(10,chr2);
    System.out.println(sb);
   }
}
```

StringBuffer insert(int offset, char[] str) Method

```
public class Example {
   public static void main(String[] args) {
    StringBuffer sb=new StringBuffer("Hello ");
    char chr1[]= {'J','a','v','a'};
    char chr2[]= {'W','o','r','l','d'};
    sb.insert(6,chr1);
    System.out.println(sb);
    sb.insert(10,chr2);
    System.out.println(sb);
   }
}
```

StringBuffer class-Methods

- **StringBuffer delete() Method**
  - The **java.lang.StringBuffer.delete()** method removes the characters in a substring of this sequence. The substring begins at the specified **start** and extends to the character at index **end - 1** or to the end of the sequence if no such character exists. If **start** is equal to **end**, no changes are made.
- **Declaration**

```
StringBuffer delete(int start, int end)
```

- **Parameters**
  - **start** -- This is the beginning index, inclusive.
  - **end** -- This is the ending index, exclusive.
- **Return Value**
  - This method returns this object.

StringBuffer delete() Method Example

```
public class Example {


   public static void main(String[] args) {
    StringBuffer sb1=new StringBuffer("Hello");
    StringBuffer sb2=new StringBuffer("Java Language");
    sb1.delete(1,3);
    System.out.println(sb1);
    sb2.delete(1,7);
    System.out.println(sb2);
   }
}
```

StringBuffer deleteCharAt() Method Example

```
public class Example {


   public static void main(String[] args) {
    StringBuffer sb1=new StringBuffer("Hello");
    StringBuffer sb2=new StringBuffer("Java Language");
    sb1.deleteCharAt(3);
    System.out.println(sb1);
    sb2.deleteCharAt(4);
    System.out.println(sb2);
   }
}
```

StringBuffer class-Methods

- **StringBuffer reverse() Method**
  - The **java.lang.StringBuffer.reverse()** method causes this character sequence to be replaced by the reverse of the sequence.
- **Declaration**

StringBuffer reverse()

- **Parameters**
  - **NA**
- **Return Value**
  - This method returns a reference to this object.

StringBuffer reverse() Method Example

```
public class Example {

   public static void main(String[] args) {
    StringBuffer sb1=new StringBuffer("Hello");
    StringBuffer sb2=new StringBuffer("12345");
    sb1.reverse();
    System.out.println(sb1);
    sb2.reverse();
    System.out.println(sb2);

   }
}
```