MODULE 4
ADVANCED FEATURES OF JAVA

CHAPTER 1
Java Library & Collections framework

Prepared By Mr. EBIN PM, AP, IESCE 1

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:

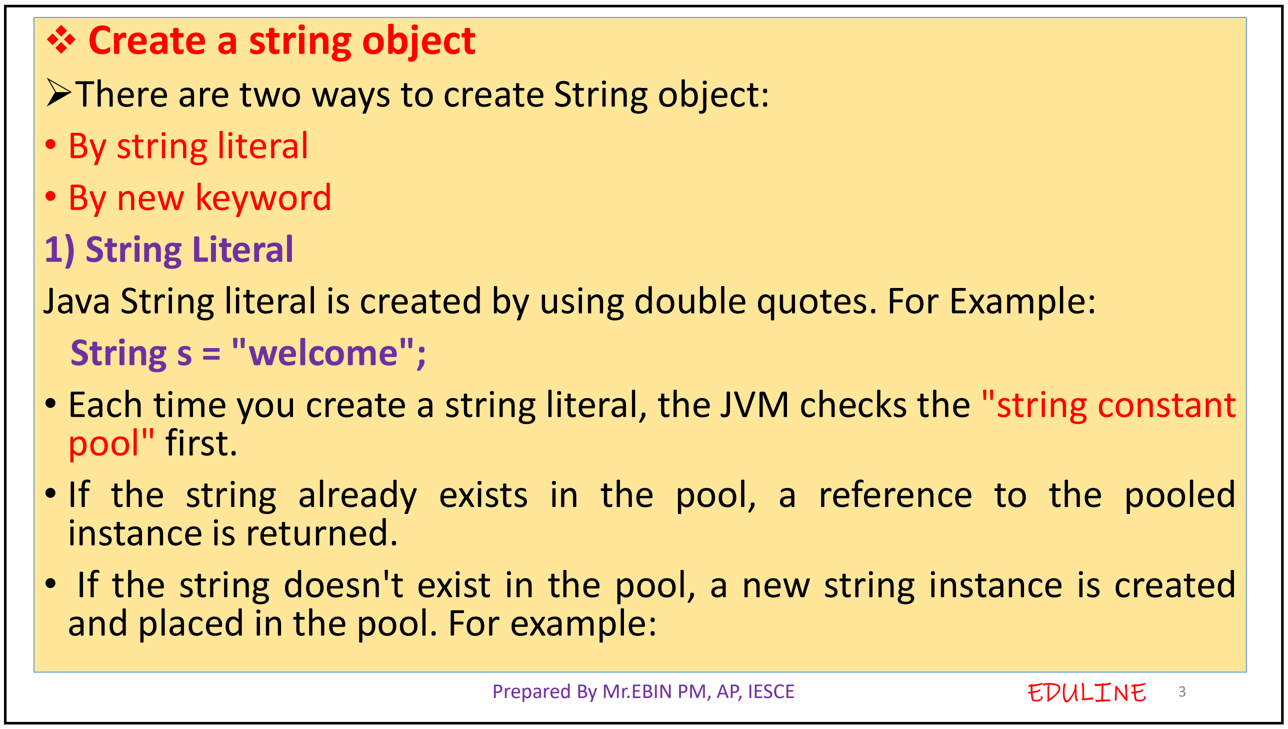
```
char[ ] ch={‘h','a',‘i',‘j',‘a',‘v',‘a'};
String s=new String(ch);
```

is same as:

```
String s = “haijava";
```

- 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.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 2



❖ Create a string object

➢There are two ways to create String object:

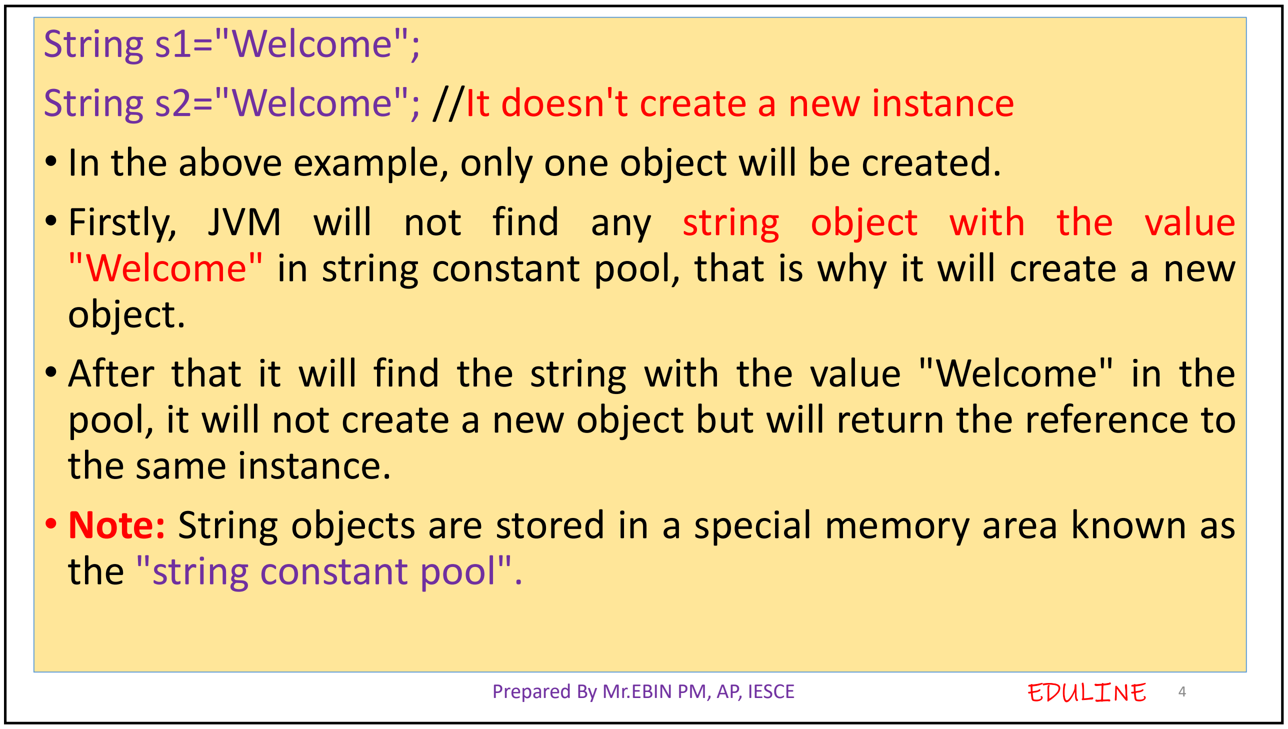
- By string literal
- By new keyword

1) String Literal

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. For example:



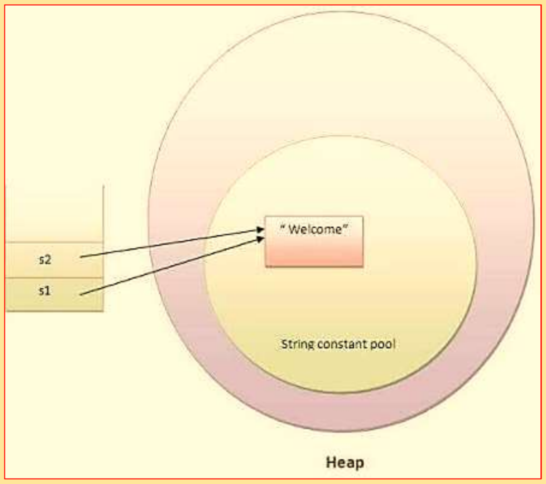
String s1="Welcome";

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

- In the above 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.
- **Note:** String objects are stored in a special memory area known as the "string constant pool".

➢Why Java uses the concept of String literal

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

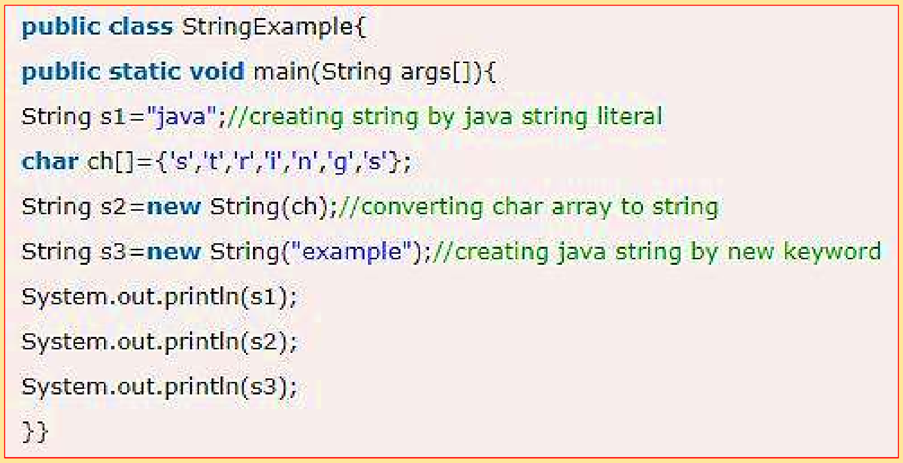


2) By new keyword

String s=new String("Welcome"); //creates two objects and one reference variable

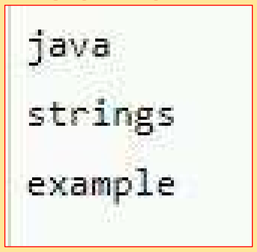
- In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool.
- The variable s will refer to the object in a heap (non-pool).

String Example



```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

OUTPUT



```
java
strings
example
```

STRING CONSTRUCTORS

- The string class supports several types of constructors in Java APIs. The most commonly used constructors of String class are as follows:

1. String() : To create an empty String, we will call a default constructor. For example:

String s = new String();

- It will create a string object in the heap area with no value

2. String(String str) : It will create a string object in the heap area and stores the given value in it. For example:

String s2 = new String("Hello Java");

Now, the object contains Hello Java.

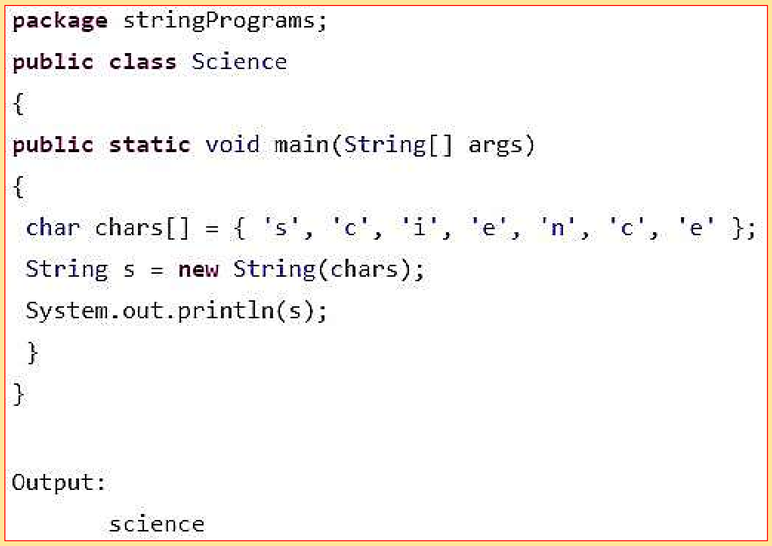
3. String(char chars[ ]) : It will create a string object and stores the array of characters in it. For example:

char chars[ ] = { 'a', 'b', 'c', 'd' };

String s3 = new String(chars);

The object reference variable s3 contains the address of the value stored in the heap area.

- Let's take an example program where we will create a string object and store an array of characters in it



```
package stringPrograms;
public class Science
{
public static void main(String[] args)
{
 char chars[] = { 's', 'c', 'i', 'e', 'n', 'c', 'e' };
 String s = new String(chars);
 System.out.println(s);
 }
}


Output:
       science
```

4. String(char chars[ ], int startIndex, int count)

- It will create and initializes a string object with a subrange of a character array.
- The argument startIndex specifies the index at which the subrange begins and count specifies the number of characters to be copied.

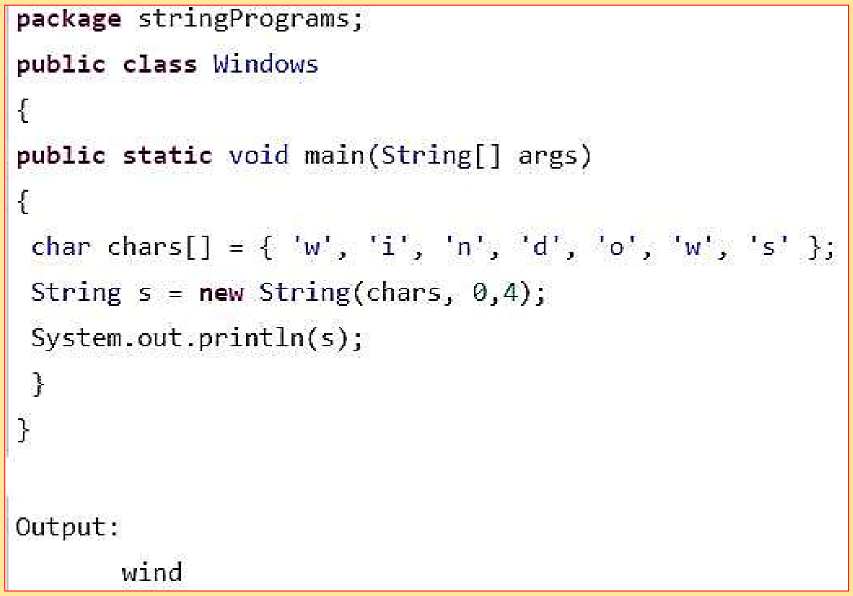
For example:

char chars[ ] = { 'w', 'i', 'n', 'd', 'o', 'w', 's' };

String str = new String(chars, 2, 3);

- The object str contains the address of the value "ndo" stored in the heap area because the starting index is 2 and the total number of characters to be copied is 3

EXAMPLE



```
package stringPrograms;
public class Windows
{
public static void main(String[] args)
{
 char chars[] = { 'w', 'i', 'n', 'd', 'o', 'w', 's' };
 String s = new String(chars, 0,4);
 System.out.println(s);
 }
}


Output:
       wind
```

- In this example program, we will construct a String object that contains the same characters sequence as another string object.

```
package stringPrograms;
public class MakeString
{
 public static void main(String[] args)
 {
   char chars[] = { 'F', 'A', 'N' };
  String s1 = new String(chars);
  String s2 = new String(s1);
 System.out.println(s1);
 System.out.println(s2);
 }
}

Output:
       FAN
       FAN
```

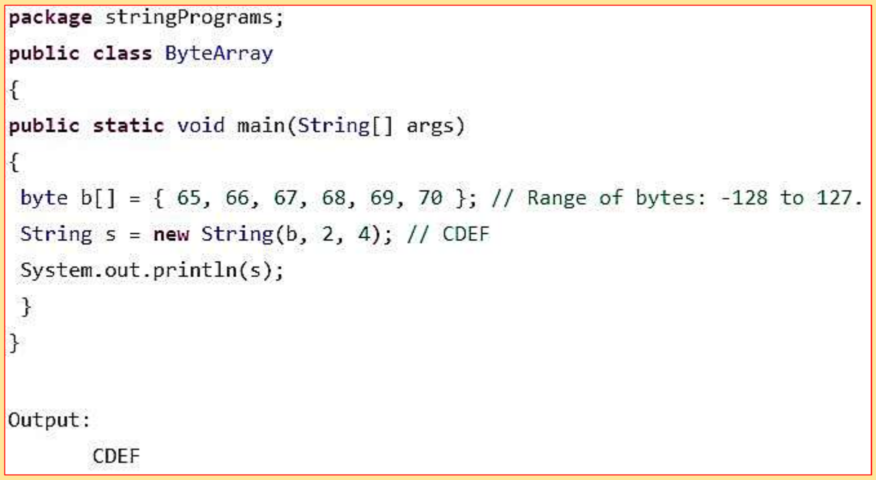
As you can see the output, s1 and s2 contain the same string. Thus, we can create one string from another string.

5. String(byte byteArr[ ]) : It constructs a new string object by decoding the given array of bytes (i.e, by decoding ASCII values into the characters) according to the system's default character set.

```
package stringPrograms;
public class ByteArray
{
public static void main(String[] args)
{
 byte b[] = { 97, 98, 99, 100 }; // Range of bytes: -128 to 127. These byte
values will be converted into corresponding characters.
 String s = new String(b);
 System.out.println(s);
 }
}

Output:
      abcd
```

6. String(byte byteArr[ ], int startIndex, int count)

This constructor also creates a new string object by decoding the ASCII values using the system's default character set.



```
package stringPrograms;
public class ByteArray
{
public static void main(String[] args)
{
 byte b[] = { 65, 66, 67, 68, 69, 70 }; // Range of bytes: -128 to 127.
 String s = new String(b, 2, 4); // CDEF
 System.out.println(s);
 }
}

Output:
       CDEF
```

STRING LENGTH

- The **java string length()** method gives length of the string. It returns count of total number of characters.
- **Internal implementation**

```
public int length() {
        return value.length;
    }
```

Signature - The signature of the string length() method is given below:

public int length()

String length() method example - 1

```
public class LengthExample{
public static void main(String args[]){
String s1="javatpoint";
String s2="python";
System.out.println("string length is: "+s1.length());//10 is the length of javatpoint string
System.out.println("string length is: "+s2.length());//6 is the length of python string
}}
```

Output

```
string length is: 10
string length is: 6
```

String length() method example - 2

```
public class LengthExample2 {
    public static void main(String[] args) {
        String str = "Javatpoint";
        if(str.length()>0) {
            System.out.println("String is not empty and length is: "+str.length());
        }
        str = "";
        if(str.length()==0) {
            System.out.println("String is empty now: "+str.length());
        }
    }
}
```

Output

```
String is not empty and length is: 10
String is empty now: 0
```

STRING COMPARISON

- We can compare string in java on the basis of content and reference
- There are three ways to compare string in java:

  By **equals() method**

  By **= = operator**

  By **compareTo() method**

❖**String compare by equals() method**

- The String equals() method compares the original content of the string.
- It compares values of string for equality. String class provides two methods

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 19

public boolean equals(Object another) compares this string to the specified object.

public boolean equalsIgnoreCase(String another) compares this String to another string, ignoring case.

```
class Teststringcomparison1{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   String s4="Saurav";
   System.out.println(s1.equals(s2));//true
   System.out.println(s1.equals(s3));//true
   System.out.println(s1.equals(s4));//false
 }
}
```

```
Output:true
       true
       false
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 20

Example 2

```
class Teststringcomparison2{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="SACHIN";

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

Output

```
false
true
```

❖String compare by == operator

- The = = operator compares references not values.

```
class Teststringcomparison3{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   System.out.println(s1==s2);//true (because both refer to same instance)
   System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)
 }
}
```

```
Output:true
       false
```

❖String compare by compareTo() method

- The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

s1 == s2	0
s1 > s2	positive value
s1 < s2	negative value

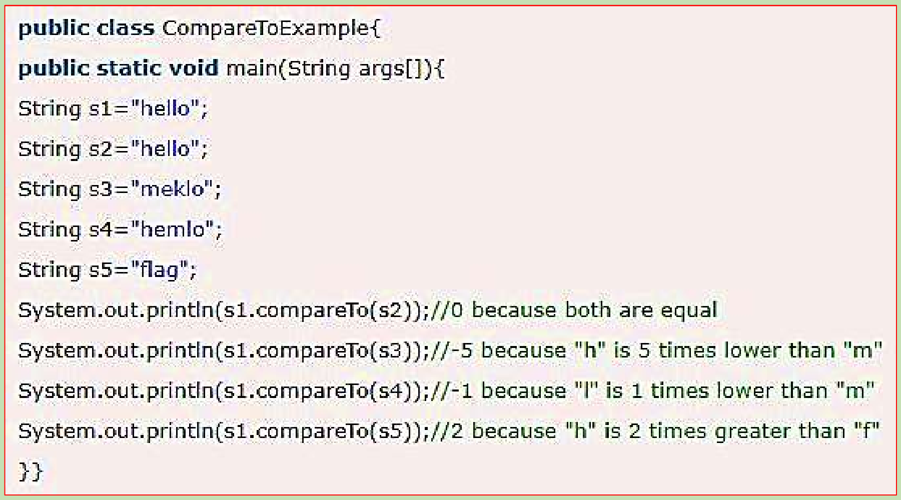
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 23

```
class Teststringcomparison4{
 public static void main(String args[]){
  String s1="Sachin";
  String s2="Sachin";
  String s3="Ratan";
  System.out.println(s1.compareTo(s2));//0
  System.out.println(s1.compareTo(s3));//1(because s1>s3)
  System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
 }
}
```

```
Output:0
        1
        -1
```

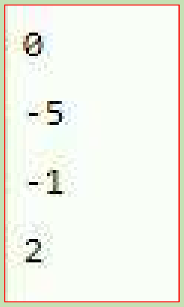
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 24

Eg:



```
public class CompareToExample{
public static void main(String args[]){
String s1="hello";
String s2="hello";
String s3="meklo";
String s4="hemlo";
String s5="flag";
System.out.println(s1.compareTo(s2));//0 because both are equal
System.out.println(s1.compareTo(s3));//-5 because "h" is 5 times lower than "m"
System.out.println(s1.compareTo(s4));//-1 because "l" is 1 times lower than "m"
System.out.println(s1.compareTo(s5));//2 because "h" is 2 times greater than "f"
}}
```

Output



```
0
-5
-1
2
```

SEARCHING STRINGS

String contains()

- 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.

Internal implementation

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

Signature

- The signature of string contains() method is given below:

  public boolean contains(CharSequence sequence)

```
class ContainsExample{
public static void main(String args[]){
String name="what do you know about me";
System.out.println(name.contains("do you know"));
System.out.println(name.contains("about"));
System.out.println(name.contains("hello"));
}}
```

Output

```
true
true
false
```

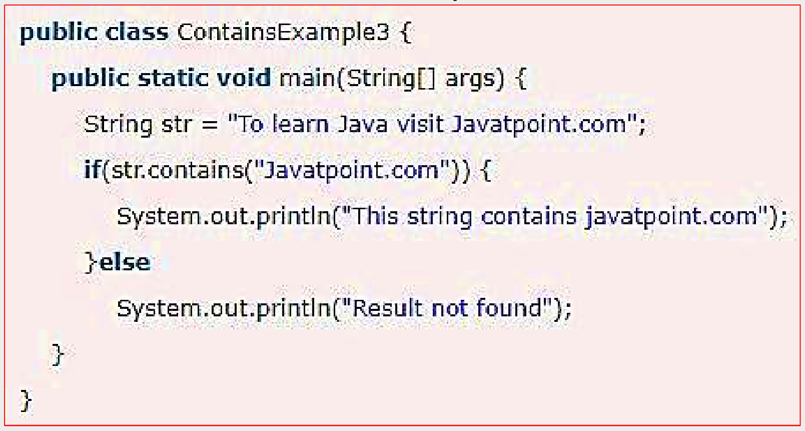
Eg 2 - The contains() method searches case sensitive char sequence. If the argument is not case sensitive, it returns false. Let's see an example below.

```
public class ContainsExample2 {
    public static void main(String[] args) {
        String str = "Hello Javatpoint readers";
        boolean isContains = str.contains("Javatpoint");
        System.out.println(isContains);
        // Case Sensitive
        System.out.println(str.contains("javatpoint")); // false
    }
}
```

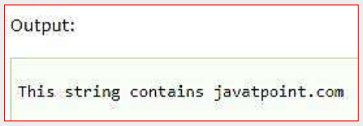
Output

```
true
false
```

Eg 3 -The contains() method is helpful to find a char-sequence in the string. We can use it in control structure to produce search based result. Let us see an example below.



```
public class ContainsExample3 {
    public static void main(String[] args) {
        String str = "To learn Java visit Javatpoint.com";
        if(str.contains("Javatpoint.com")) {
            System.out.println("This string contains javatpoint.com");
        }else
            System.out.println("Result not found");
    }
}
```



```
Output:

This string contains javatpoint.com
```

CHARACTER EXTRACTION

❖String charAt()

- The **java string charAt()** method returns a char value at the given index number.
- The index number starts from 0 and goes to n-1, where n is length of the string.
- It returns StringIndexOutOfBoundsException if given index number is greater than or equal to this string length or a negative number.
- **Signature** - The signature of string charAt() method is given below:

  public char charAt(int index)

Example:

```
public class CharAtExample{
public static void main(String args[]){
String name="javatpoint";
char ch=name.charAt(4);//returns the char value at the 4th index
System.out.println(ch);
}}
```

Output

```
t
```

❖StringIndexOutOfBoundsException with charAt()

- Let's see the example of charAt() method where we are passing greater index value.
- In such case, it throws StringIndexOutOfBoundsException at run time.

```
public class CharAtExample{
public static void main(String args[]){
String name="javatpoint";
char ch=name.charAt(10);//returns the char value at the 10th index
System.out.println(ch);
}}
```

Output:

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException:
String index out of range: 10
at java.lang.String.charAt(String.java:658)
at CharAtExample.main(CharAtExample.java:4)
```

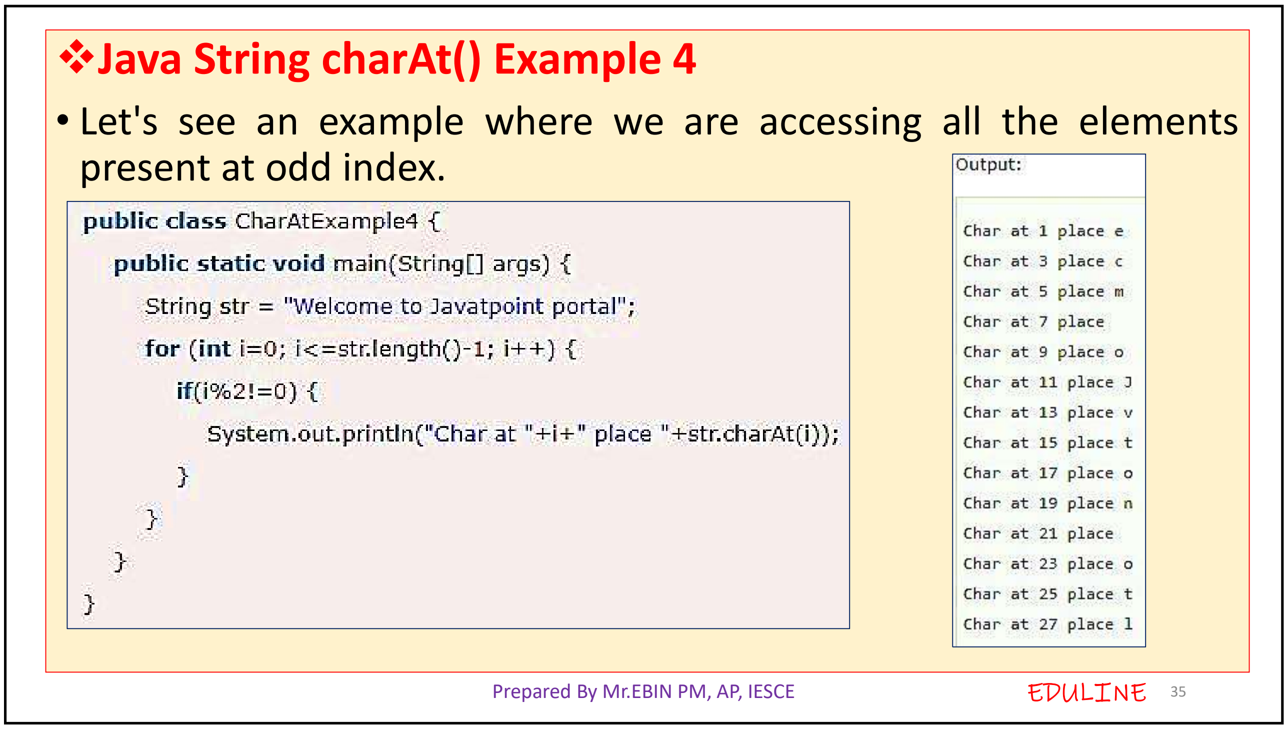
❖Java String charAt() Example 3

- Let's see a simple example where we are accessing first and last character from the provided string.

```java
public class CharAtExample3 {
    public static void main(String[] args) {
    String str = "Welcome to Javatpoint portal";
    int strLength = str.length();
    // Fetching first character
    System.out.println("Character at 0 index is: "+ str.charAt(0));
    // The last Character is present at the string length-1 index
    System.out.println("Character at last index is: "+ str.charAt(strLength-1));
    }
}
```

Output:

```
Character at 0 index is: W
Character at last index is: l
```



❖Java String charAt() Example 4

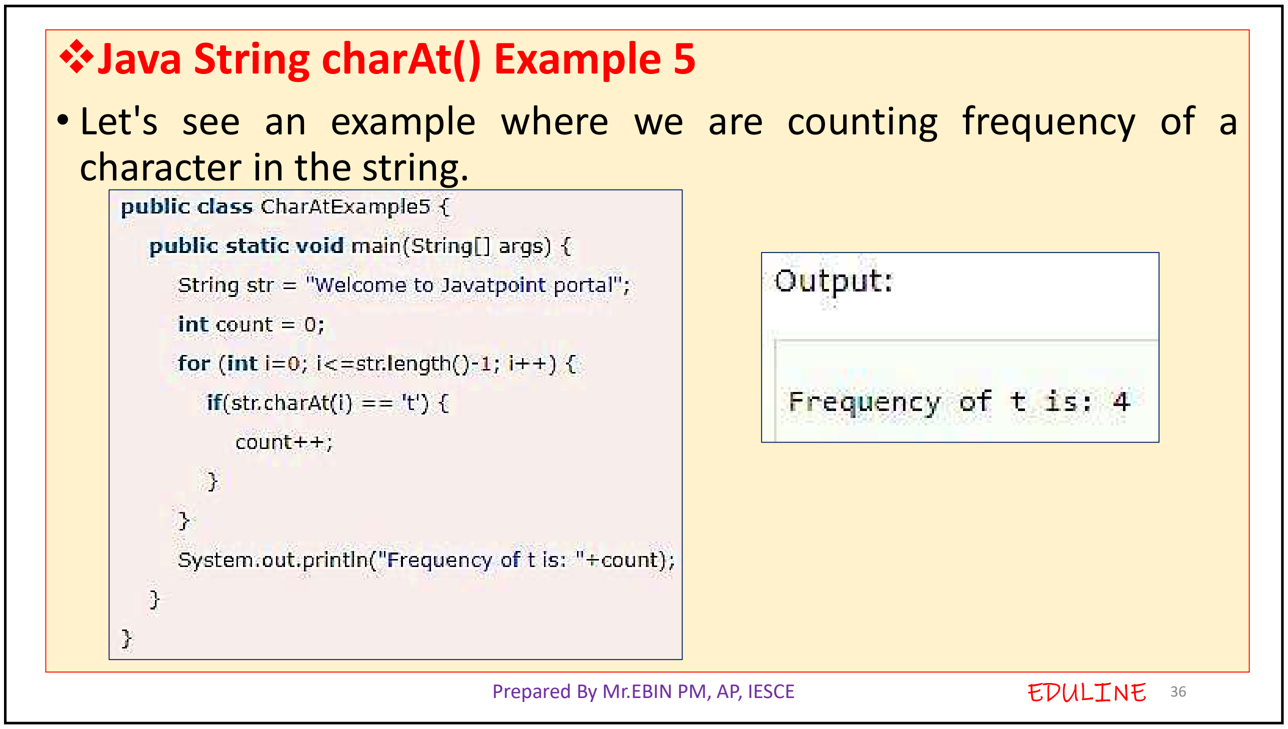
- Let's see an example where we are accessing all the elements present at odd index.

```
public class CharAtExample4 {
    public static void main(String[] args) {
        String str = "Welcome to Javatpoint portal";
        for (int i=0; i<=str.length()-1; i++) {
            if(i%2!=0) {
                System.out.println("Char at "+i+" place "+str.charAt(i));
            }
        }
    }
}
```

Output:

```
Char at 1 place e
Char at 3 place c
Char at 5 place m
Char at 7 place
Char at 9 place o
Char at 11 place J
Char at 13 place v
Char at 15 place t
Char at 17 place o
Char at 19 place n
Char at 21 place
Char at 23 place o
Char at 25 place t
Char at 27 place l
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 35



❖Java String charAt() Example 5

- Let's see an example where we are counting frequency of a character in the string.

```
public class CharAtExample5 {
    public static void main(String[] args) {
        String str = "Welcome to Javatpoint portal";
        int count = 0;
        for (int i=0; i<=str.length()-1; i++) {
            if(str.charAt(i) == 't') {
                count++;
            }
        }
        System.out.println("Frequency of t is: "+count);
    }
}
```

Output:

```
Frequency of t is: 4
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 36

MODIFY STRINGS

- The **java string replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

Signature

- There are two type of replace methods in java string.

```
public String replace(char oldChar, char newChar)
                        and
public String replace(CharSequence target, CharSequence
                                          replacement)
```

- The second replace method is added since JDK 1.5.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 37

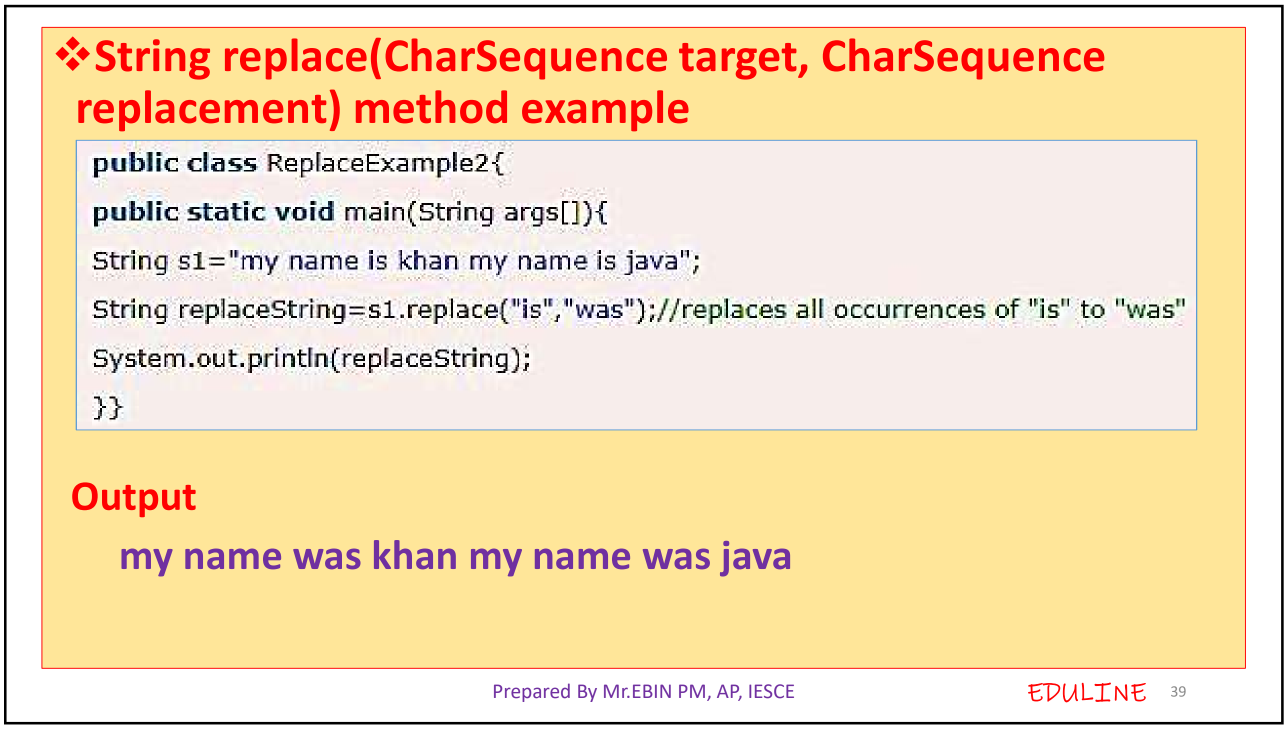
❖ String replace(char old, char new) method example

```
public class ReplaceExample1{
public static void main(String args[]){
String s1="java is a very good language";
    // replaces all occurrences  of 'a' to 'e'
String replaceString=s1.replace('a','e');
System.out.println(replaceString);
}}
```

Output

```
jeve is e very good lenguege
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 38



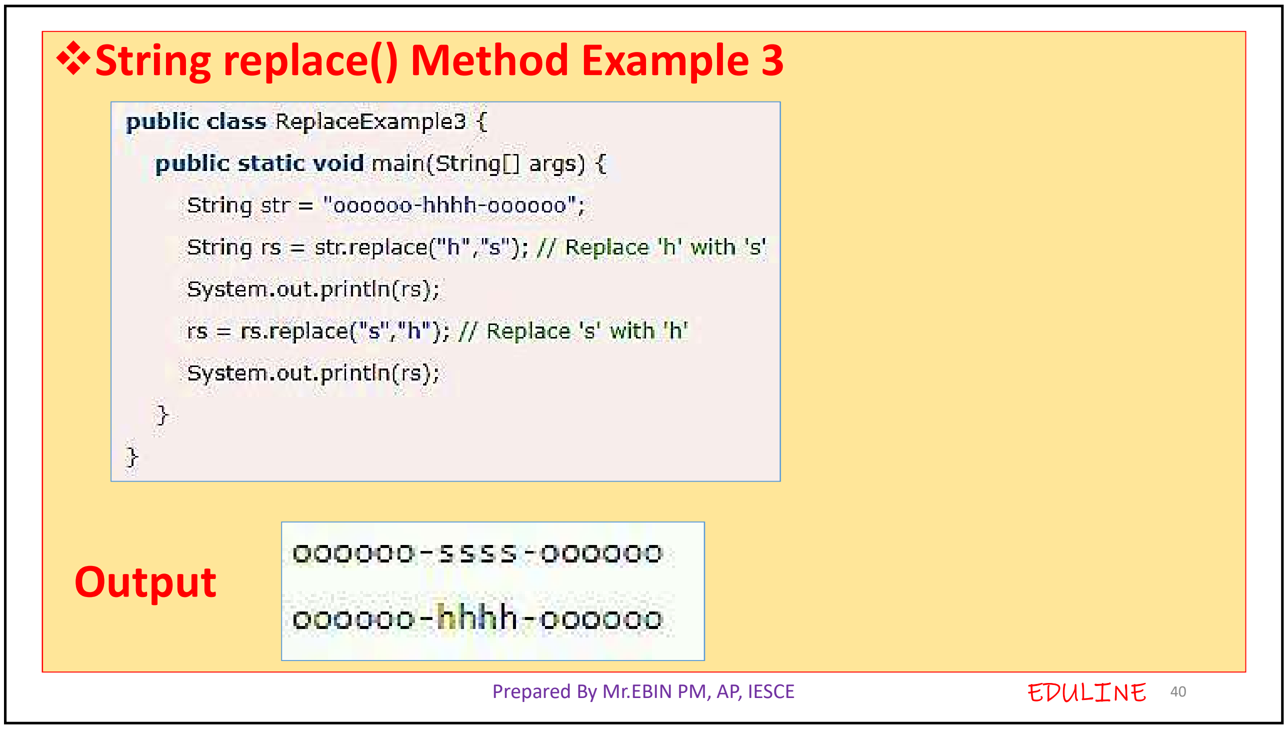
❖String replace(CharSequence target, CharSequence replacement) method example

```
public class ReplaceExample2{
public static void main(String args[]){
String s1="my name is khan my name is java";
String replaceString=s1.replace("is","was");//replaces all occurrences of "is" to "was"
System.out.println(replaceString);
}}
```

Output

my name was khan my name was java

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 39



❖String replace() Method Example 3

```
public class ReplaceExample3 {
    public static void main(String[] args) {
        String str = "oooooo-hhhh-oooooo";
        String rs = str.replace("h","s"); // Replace 'h' with 's'
        System.out.println(rs);
        rs = rs.replace("s","h"); // Replace 's' with 'h'
        System.out.println(rs);
    }
}
```

Output

```
oooooo-ssss-oooooo
oooooo-hhhh-oooooo
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 40

- The **java string replaceAll()** method returns a string replacing all the sequence of characters matching regex and replacement string.

Internal implementation

```
 public String replaceAll(String regex, String replacement) {
return Pattern.compile(regex).matcher(this).replaceAll(replacement);
 }
```

Signature

```
        public String replaceAll(String regex, String replacement)
```

❖ String replaceAll() example: replace character

- Let's see an example to replace all the occurrences of a single character.

```
public class ReplaceAllExample1{
public static void main(String args[]){
String s1="java is a very good language";
String replaceString=s1.replaceAll("a","e");//replaces all occurrences
                                                              of "a" to "e"
System.out.println(replaceString);
}}
```

Output jeve is e very good lenguege

❖ String replaceAll() example: replace word

- Let's see an example to replace all the occurrences of single word or set of words.

```
public class ReplaceAllExample2{
public static void main(String args[]){
String s1="My name is Khan. My name is Bob. My name is Sonoo.";
String replaceString=s1.replaceAll("is","was");//replaces all occurrences of "is" to "was"
System.out.println(replaceString);
}}
```

Output

```
My name was Khan. My name was Bob. My name was Sonoo.
```

❖ String replaceAll() example: remove white spaces

- Let's see an example to remove all the occurrences of white spaces.

```
public class ReplaceAllExample3{
public static void main(String args[]){
String s1="My name is Khan. My name is Bob. My name is Sonoo.";
String replaceString=s1.replaceAll("\\s","");
System.out.println(replaceString);
}}
```

Output

```
MynameisKhan.MynameisBob.MynameisSonoo.
```

STRING VALUE OF ( )

- The java string **valueOf()** method converts different types of values into string.
- By the help of string valueOf() method, we 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.

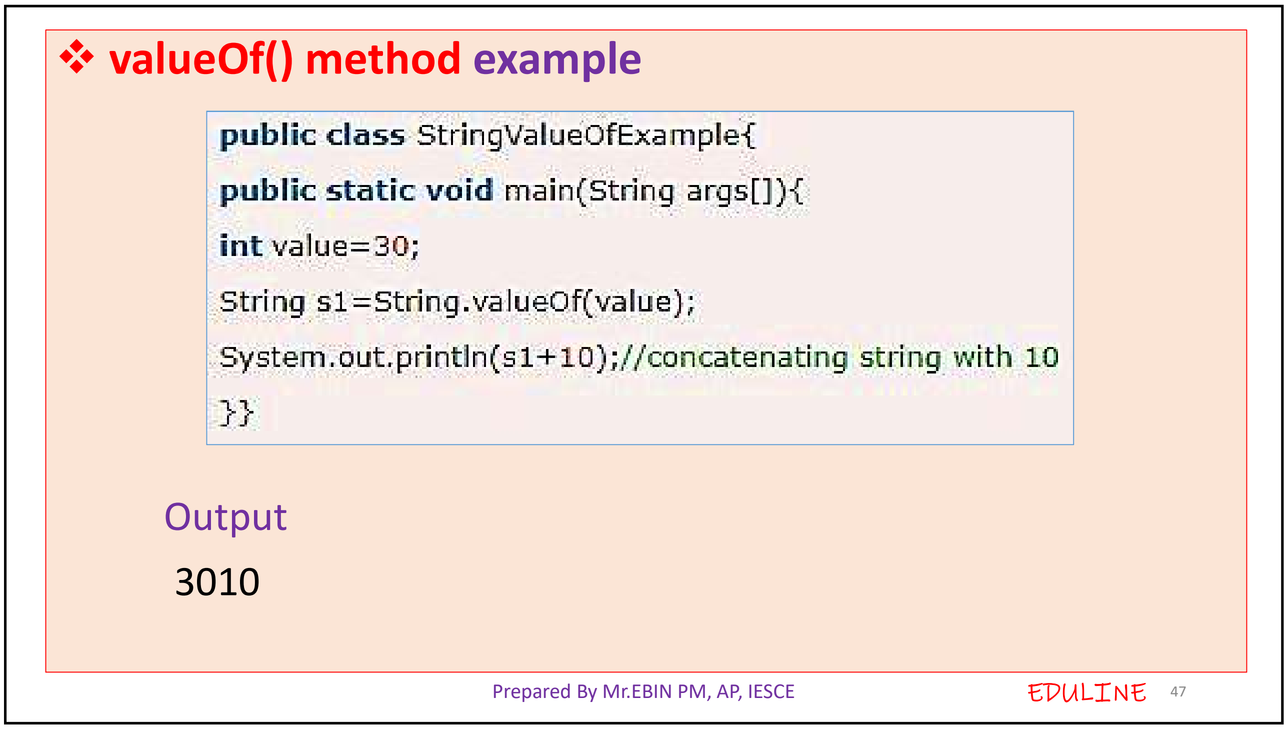
Internal implementation

```
public static String valueOf(Object obj) {
    return (obj == null) ? "null" : obj.toString();
  }
```

❖ Signature

- The signature or syntax of string valueOf() method is given below:

```
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)
```



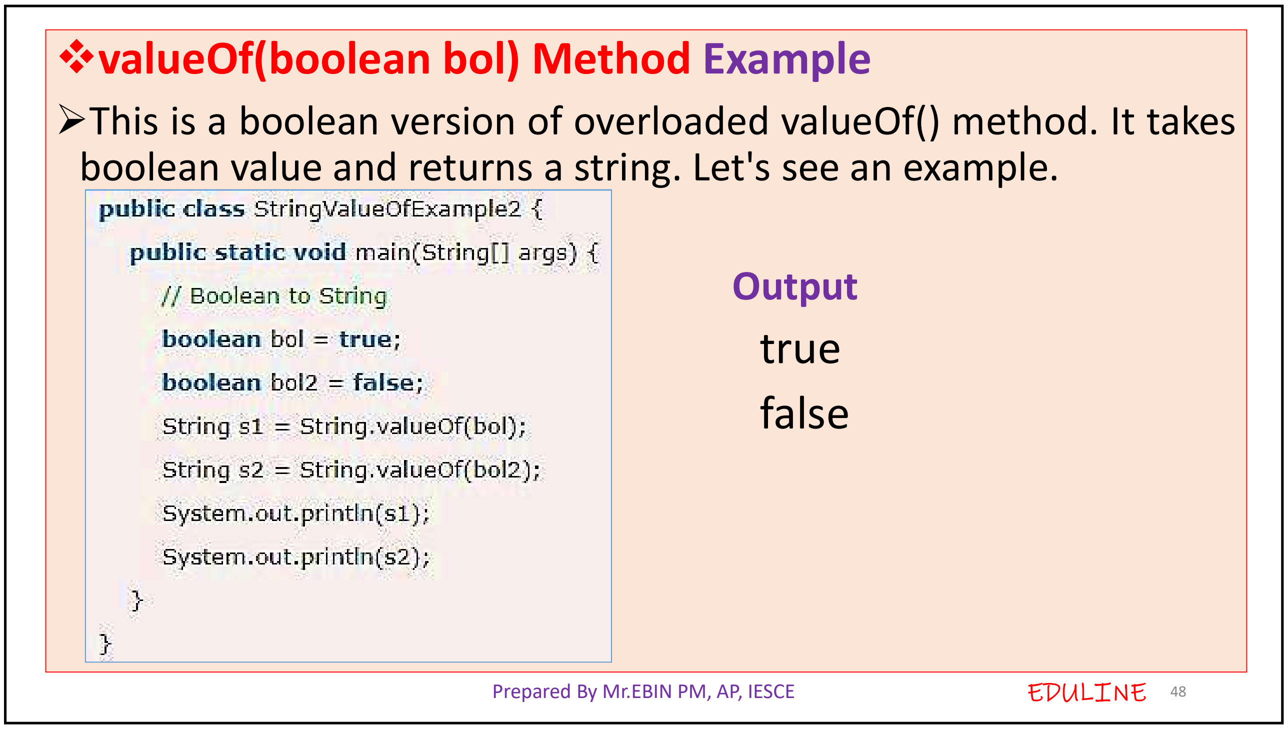
❖ valueOf() method example

```
public class StringValueOfExample{
public static void main(String args[]){
int value=30;
String s1=String.valueOf(value);
System.out.println(s1+10);//concatenating string with 10
}}
```

Output

```
3010
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 47



❖valueOf(boolean bol) Method Example

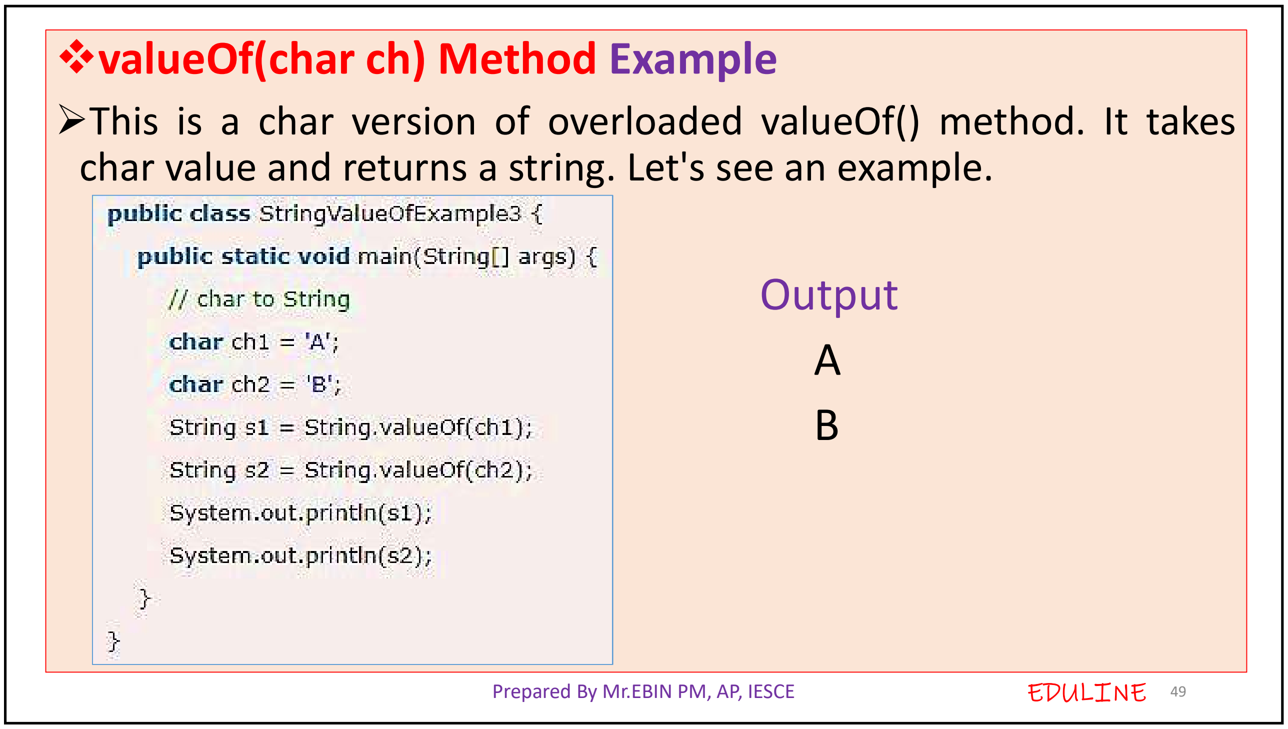
➢This is a boolean version of overloaded valueOf() method. It takes boolean value and returns a string. Let's see an example.

```
public class StringValueOfExample2 {
    public static void main(String[] args) {
        // Boolean to String
        boolean bol = true;
        boolean bol2 = false;
        String s1 = String.valueOf(bol);
        String s2 = String.valueOf(bol2);
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

Output

```
true
false
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 48



❖valueOf(char ch) Method Example

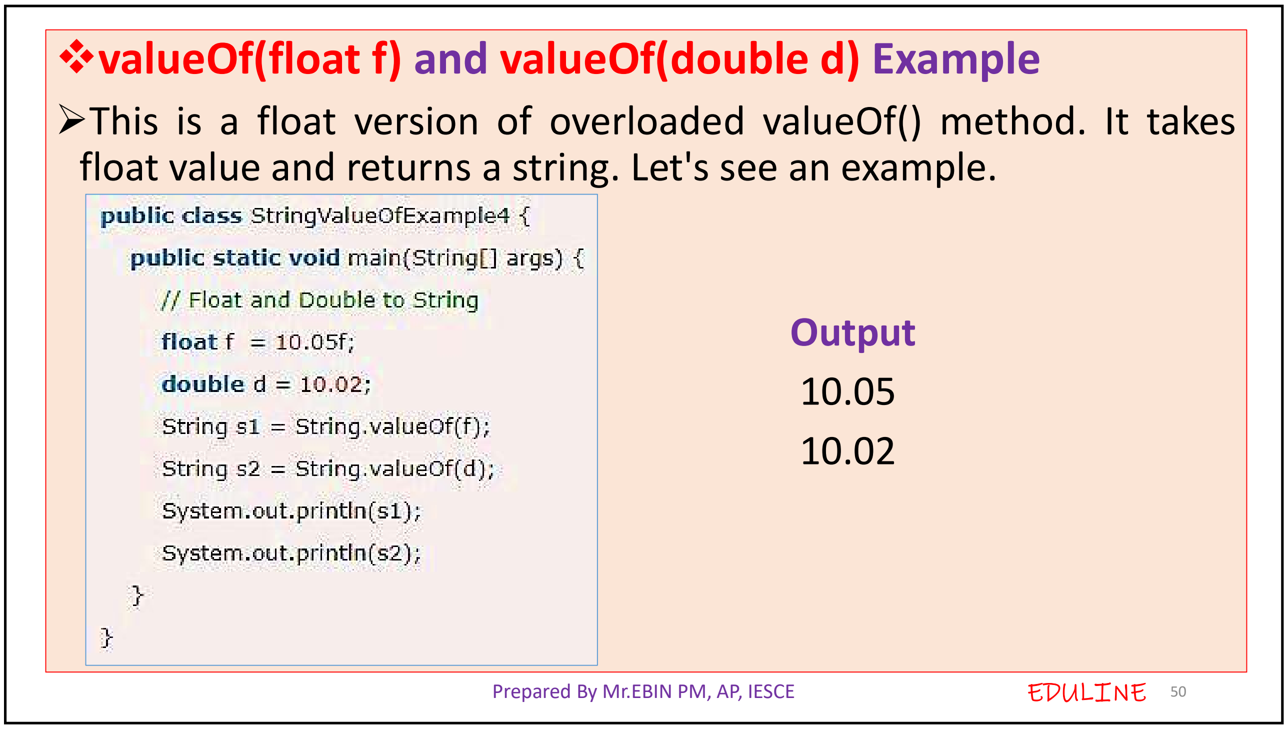
➢This is a char version of overloaded valueOf() method. It takes char value and returns a string. Let's see an example.

```
public class StringValueOfExample3 {
    public static void main(String[] args) {
        // char to String
        char ch1 = 'A';
        char ch2 = 'B';
        String s1 = String.valueOf(ch1);
        String s2 = String.valueOf(ch2);
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

Output

```
A
B
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 49



❖valueOf(float f) and valueOf(double d) Example

➢This is a float version of overloaded valueOf() method. It takes float value and returns a string. Let's see an example.

```
public class StringValueOfExample4 {
    public static void main(String[] args) {
        // Float and Double to String
        float f  = 10.05f;
        double d = 10.02;
        String s1 = String.valueOf(f);
        String s2 = String.valueOf(d);
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

Output

```
10.05
10.02
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 50

❖String valueOf() Complete Examples

```
public class StringValueOfExample5 {
   public static void main(String[] args) {
      boolean b1=true;
      byte b2=11;
      short sh = 12;
      int i = 13;
      long l = 14L;
      float f = 15.5f;
      double d = 16.5d;
      char chr[]={'j','a','v','a'};
      StringValueOfExample5 obj=new StringValueOfExample5();
      String s1 = String.valueOf(b1);
      String s2 = String.valueOf(b2);
      String s3 = String.valueOf(sh);
      String s4 = String.valueOf(i);
      String s5 = String.valueOf(l);
      String s6 = String.valueOf(f);
      String s7 = String.valueOf(d);
      String s8 = String.valueOf(chr);
      String s9 = String.valueOf(obj);
      System.out.println(s1);
      System.out.println(s2);
      System.out.println(s3);
      System.out.println(s4);
      System.out.println(s5);
      System.out.println(s6);
      System.out.println(s7);
      System.out.println(s8);
      System.out.println(s9);
   }
}
```

Output

```
true
11
12
13
14
15.5
16.5
java
StringValueOfExample5@2a139a55
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 51

❖Immutable String in Java

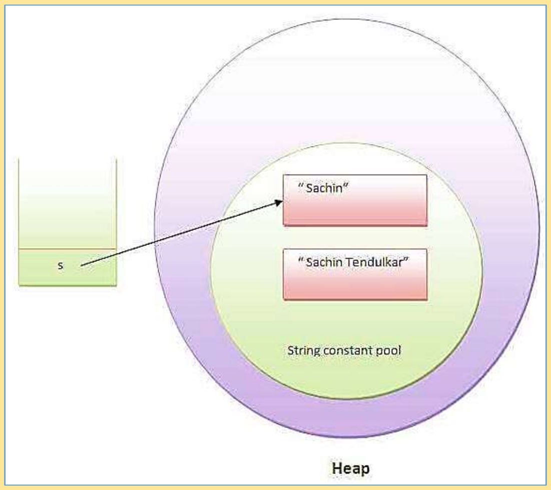
- In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable. Once string object is created its data or state can't be changed but a new string object is created. **Example**

```
class Testimmutablestring{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the end
   System.out.println(s);//will print Sachin because strings are immutable objects
 }
}
```

Output Sachin

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 52

It can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.



- As you can see in the figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".
- But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object. For example:

```
class Testimmutablestring1{
 public static void main(String args[]){
   String s="Sachin";
   s=s.concat(" Tendulkar");
   System.out.println(s);
 }
}
```

Output

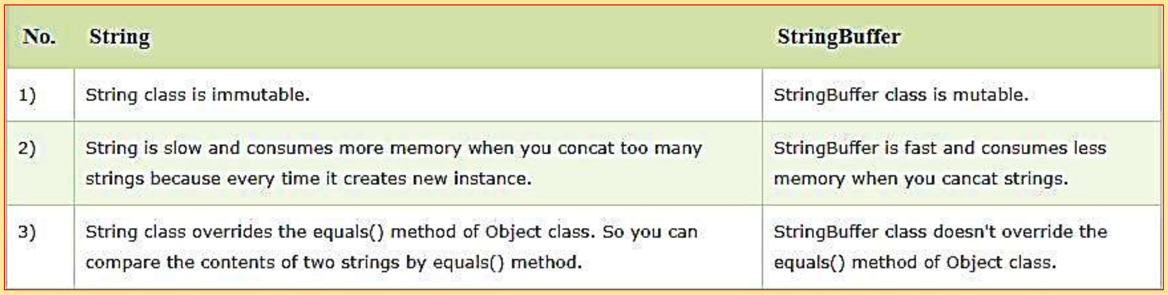
Sachin Tendulkar

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

❖Why string objects are immutable in java

- Because java uses the concept of string literal.
- Suppose there are 5 reference variables,all referes to one object "sachin".
- If one reference variable changes the value of the object, it will be affected to all the reference variables.
- That is why string objects are immutable in java.

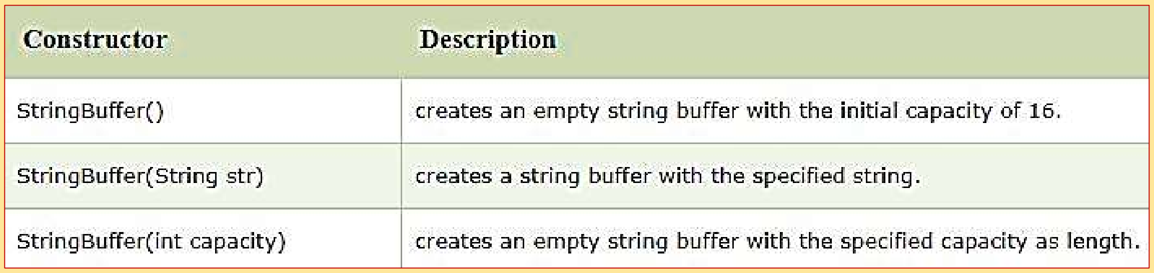
String and StringBuffer



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.

- Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

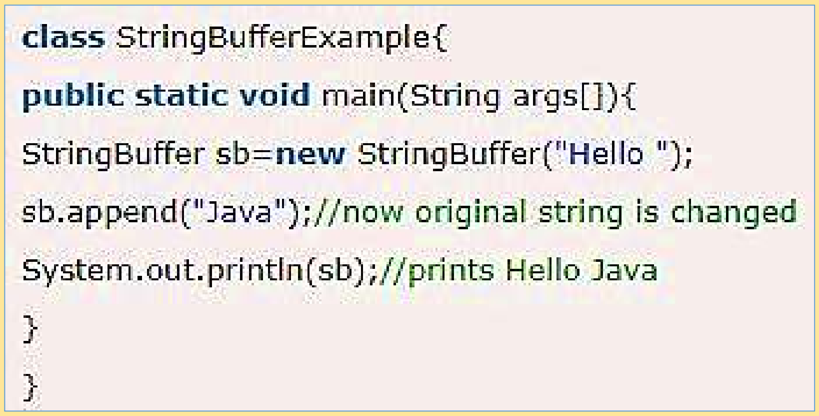
❖ Important Constructors of StringBuffer class



Constructor	Description
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.

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 append() method



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

❖StringBuffer insert() method

➢The insert() method inserts the given string with this string at the given position.

```
class StringBufferExample2{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

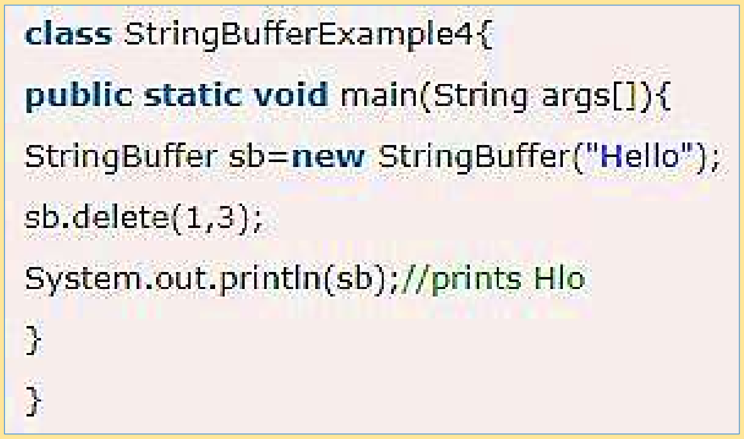
❖StringBuffer replace() method

➢The replace() method replaces the given string from the specified beginIndex and endIndex.

```
class StringBufferExample3{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

❖StringBuffer delete() method

➢The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

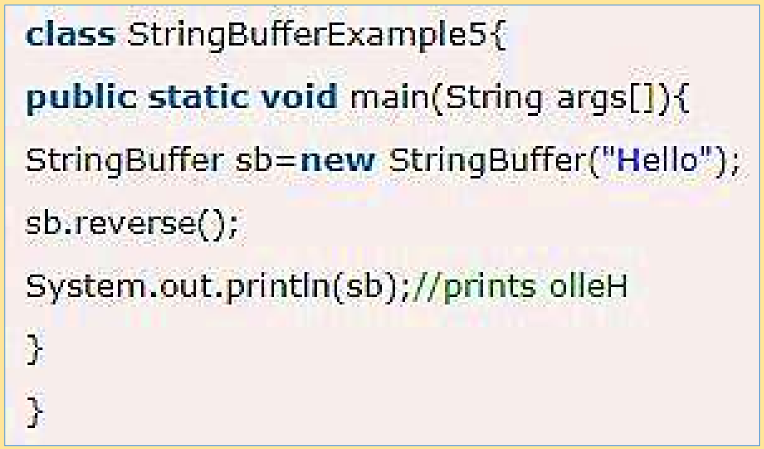


```
class StringBufferExample4{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 61

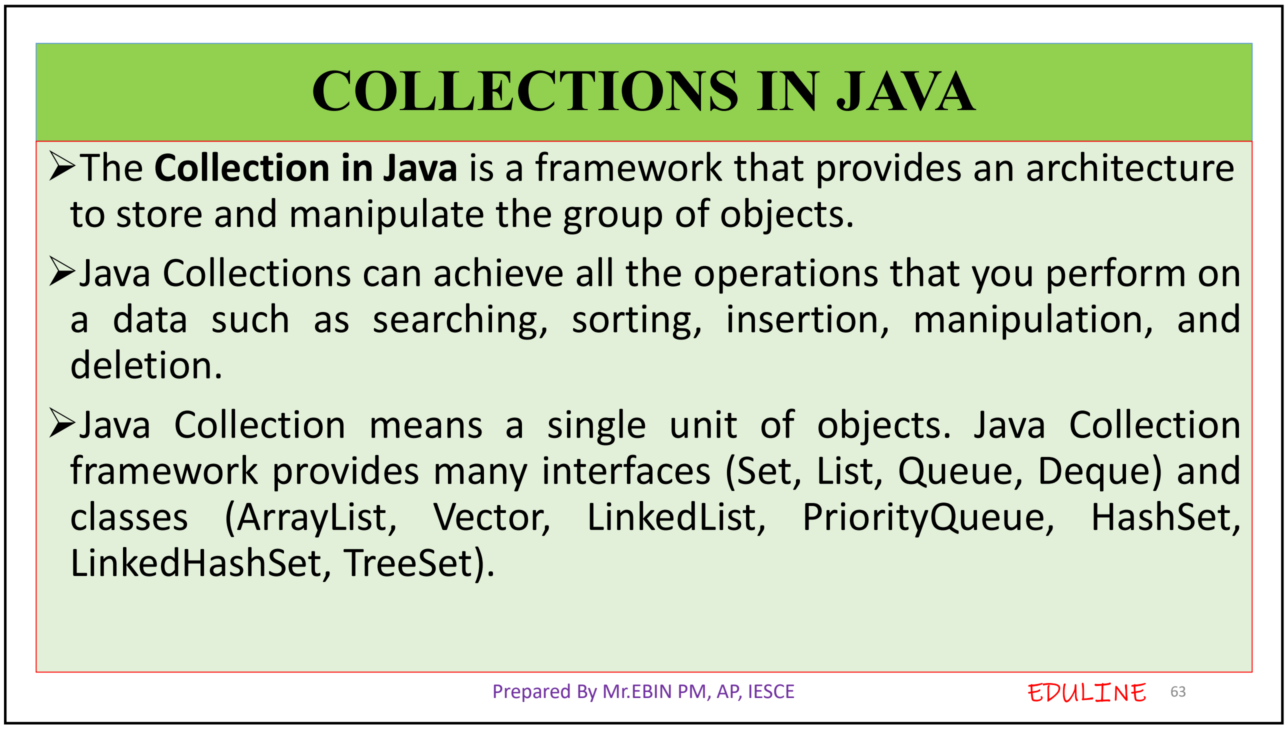
❖ StringBuffer reverse() method

➢The reverse() method of StringBuffer class reverses the current string.



```
class StringBufferExample5{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

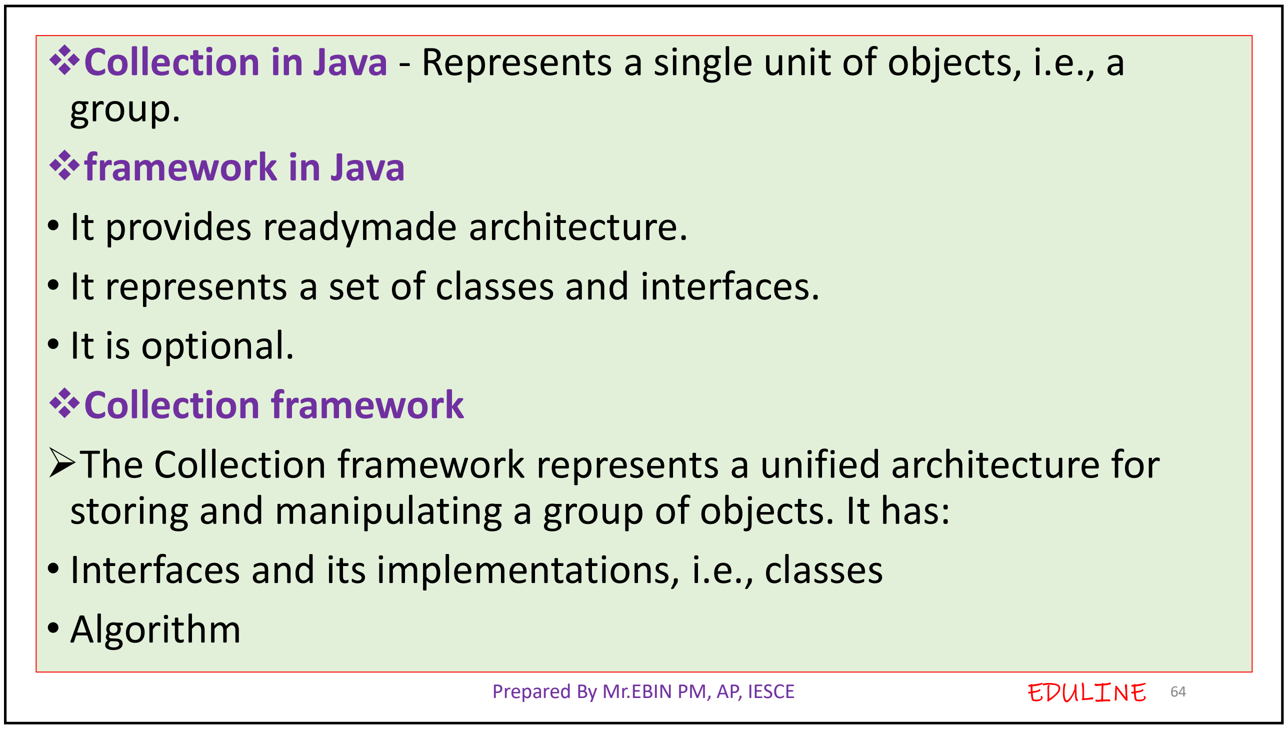
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 62



COLLECTIONS IN JAVA

- ➢The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.
- ➢Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- ➢Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 63



❖**Collection in Java** - Represents a single unit of objects, i.e., a group.

❖**framework in Java**

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

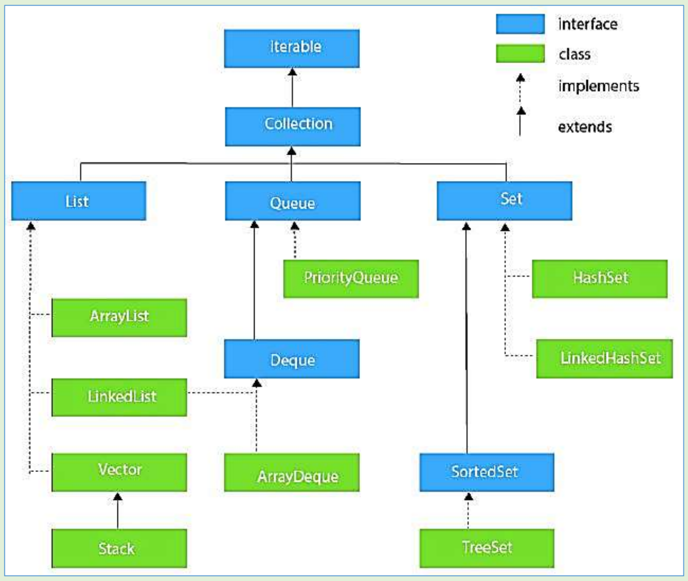
❖**Collection framework**

➢The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

- Interfaces and its implementations, i.e., classes
- Algorithm

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 64

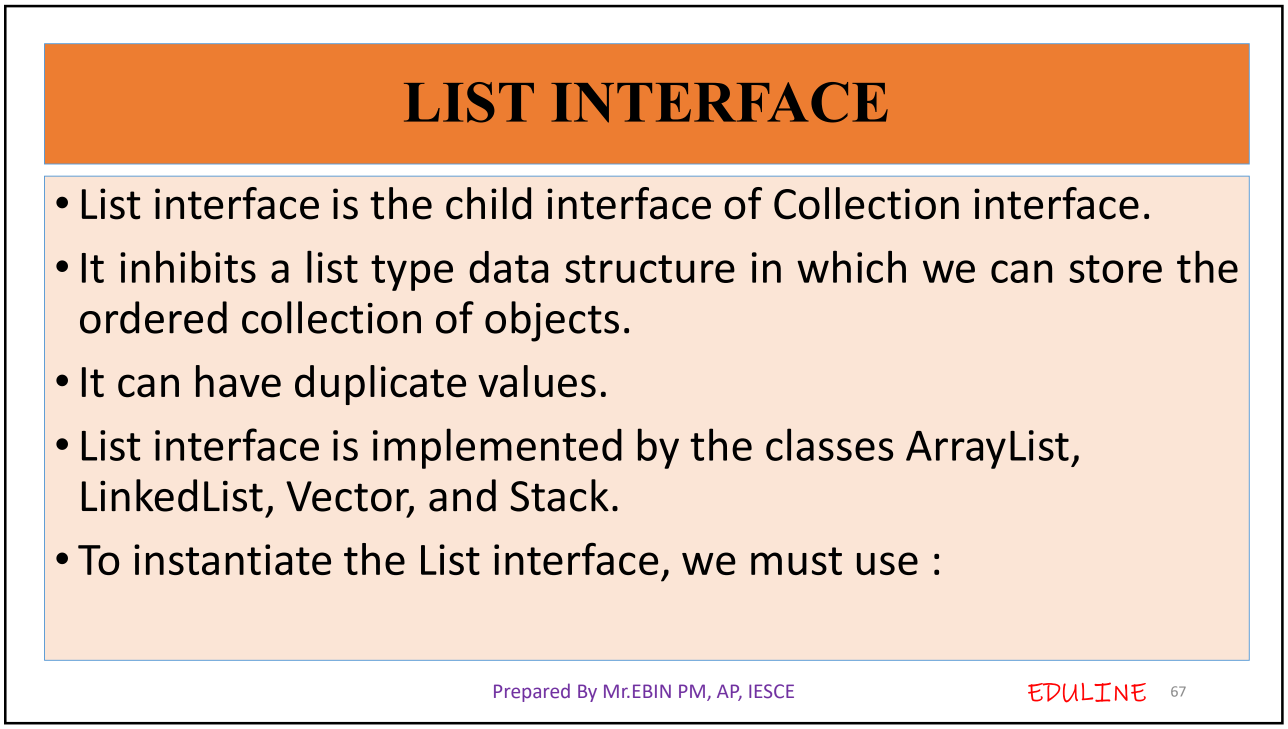
❖Hierarchy of Collection Framework



➢The **java.util** package contains all the classes and interfaces for the Collection framework.

❖Collection Interface

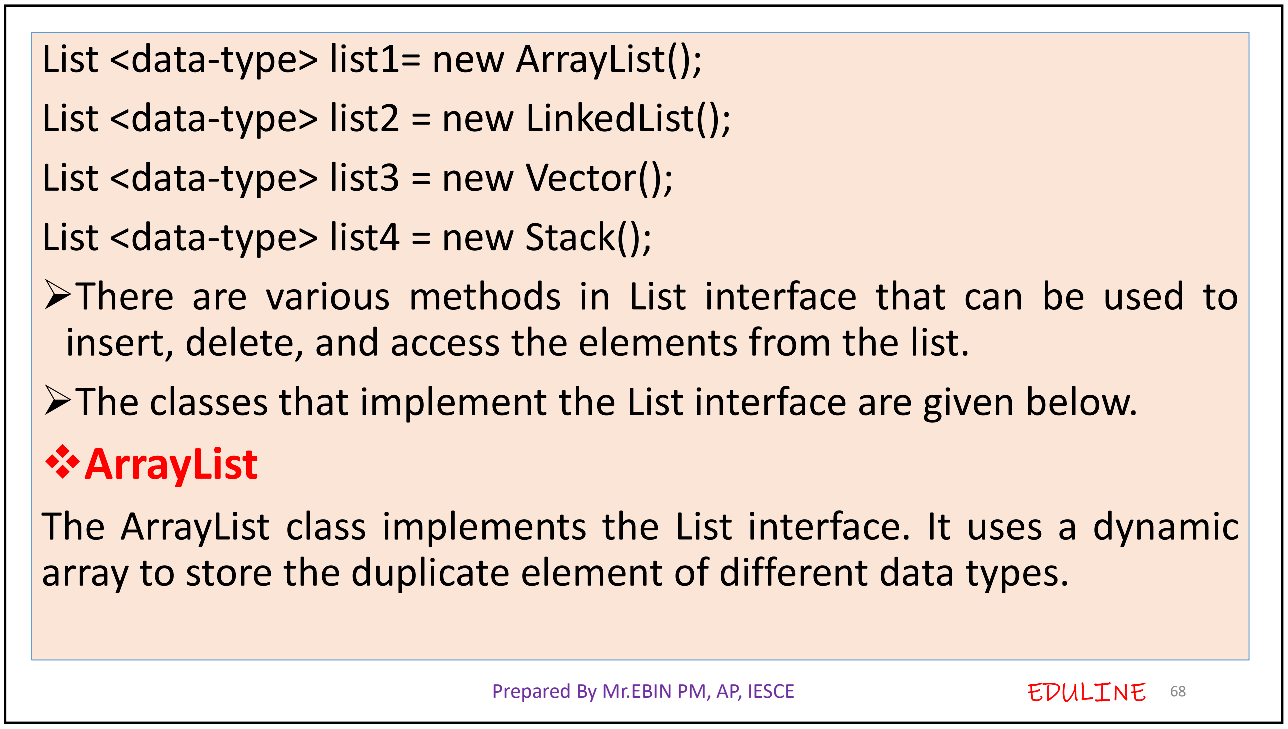
- The Collection interface is the interface which is implemented by all the classes in the collection framework.
- It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.
- Some of the methods of Collection interface are Boolean add (Object obj), Boolean addAll (Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.



LIST INTERFACE

- List interface is the child interface of Collection interface.
- It inhibits a list type data structure in which we can store the ordered collection of objects.
- It can have duplicate values.
- List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.
- To instantiate the List interface, we must use :

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 67



```
List <data-type> list1= new ArrayList();
List <data-type> list2 = new LinkedList();
List <data-type> list3 = new Vector();
List <data-type> list4 = new Stack();
```

➢There are various methods in List interface that can be used to insert, delete, and access the elements from the list.

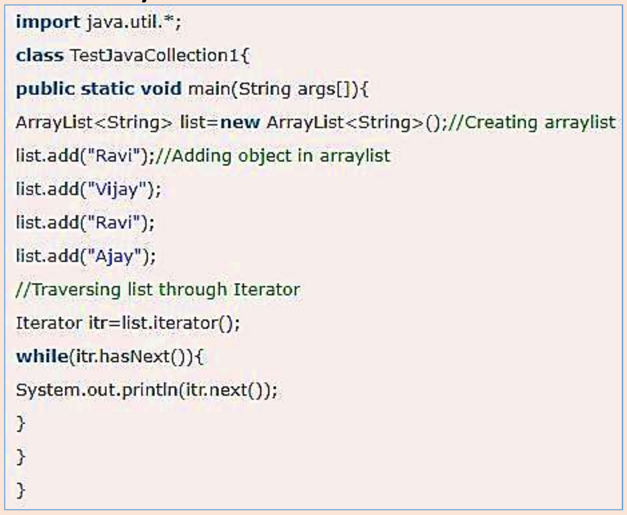
➢The classes that implement the List interface are given below.

❖ArrayList

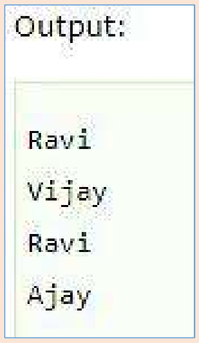
The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 68

- The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed. Consider the following example.



```
import java.util.*;
class TestJavaCollection1{
public static void main(String args[]){
ArrayList<String> list=new ArrayList<String>();//Creating arraylist
list.add("Ravi");//Adding object in arraylist
list.add("Vijay");
list.add("Ravi");
list.add("Ajay");
//Traversing list through Iterator
Iterator itr=list.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```



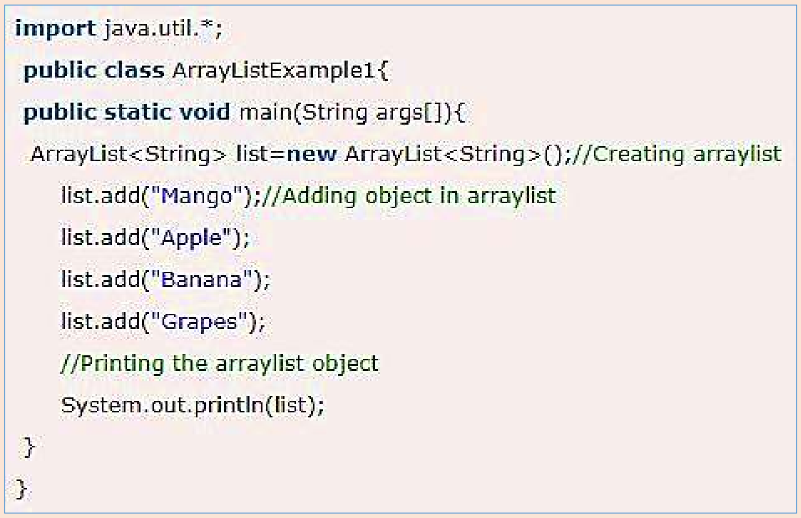
```
Output:

Ravi
Vijay
Ravi
Ajay
```

- ➢ Java ArrayList class uses a dynamic array for storing the elements.
- ➢ It is like an array, but there is no size limit. We can add or remove elements anytime.
- ➢ So, it is much more flexible than the traditional array. It is found in the java.util package. It is like the Vector in C++.
- ➢ The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of List interface here.
- ➢ The ArrayList maintains the insertion order internally.
- ➢ It inherits the AbstractList class and implements List interface.

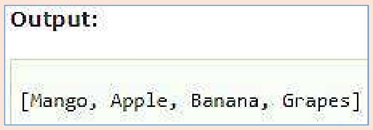
➢The important points about Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.
- In ArrayList, manipulation is little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.

ArrayList Example

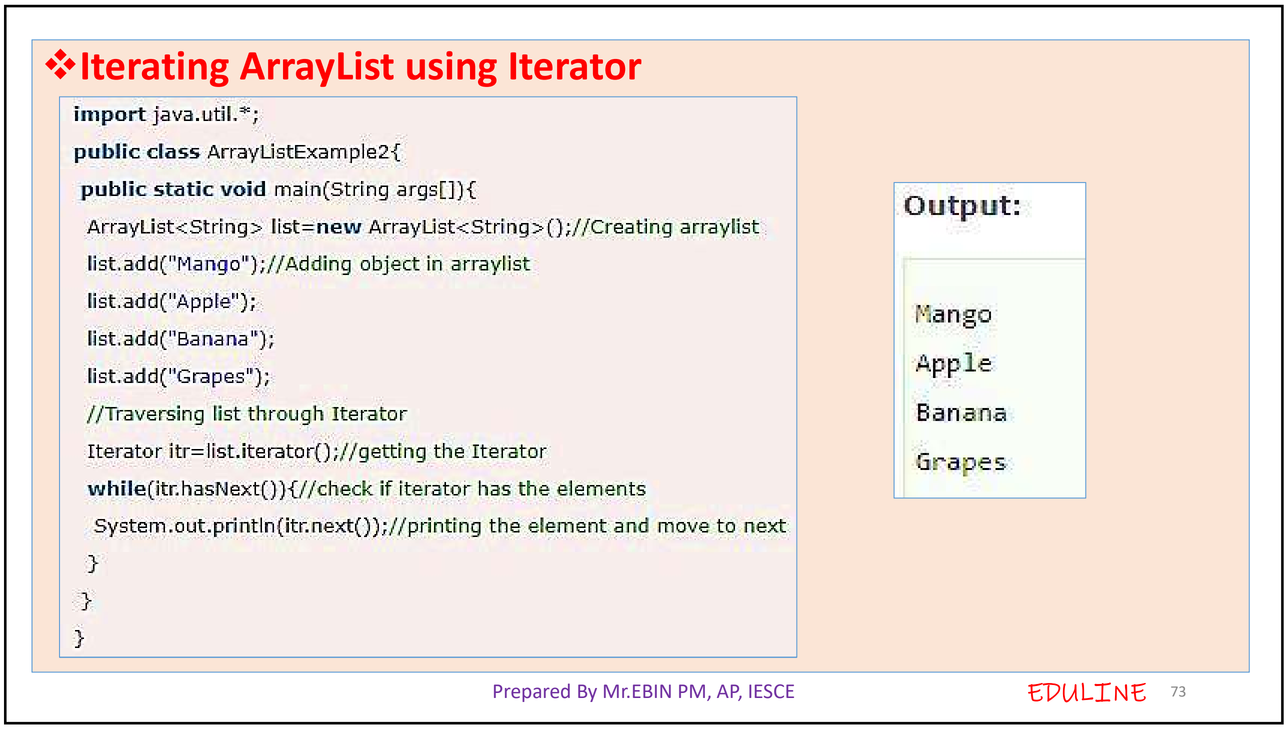


```
import java.util.*;
 public class ArrayListExample1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
      list.add("Mango");//Adding object in arraylist
      list.add("Apple");
      list.add("Banana");
      list.add("Grapes");
      //Printing the arraylist object
      System.out.println(list);
 }
}
```



Output:

```
[Mango, Apple, Banana, Grapes]
```



❖Iterating ArrayList using Iterator

```
import java.util.*;
public class ArrayListExample2{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Mango");//Adding object in arraylist
  list.add("Apple");
  list.add("Banana");
  list.add("Grapes");
  //Traversing list through Iterator
  Iterator itr=list.iterator();//getting the Iterator
  while(itr.hasNext()){//check if iterator has the elements
   System.out.println(itr.next());//printing the element and move to next
  }
 }
}
```

Output:

```
Mango
Apple
Banana
Grapes
```

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 73