8.STRING HANDLING

❖ The string constructors

- The **String** class supports several constructors.
- To create an empty **String**, you call the default constructor.
- For example,
  - String s = new String();
    will create an instance of **String** with no characters in it
- You can construct a **String** object that contains the same character sequence as another **String** object using this constructor:
  - String(String strObj) Here, strObj is a **String** object.
- **Example:**

```
class test
{
        public static void main(String args[])
        {
                char c[] = {'J', 'a', 'v', 'a'};
                String s1 = new String(c);
                System.out.println(s1);

        }
}
```

  - **OUTPUT:** Java

❖ String length

- **length()**-To find a length of string.
- **Syntax**:- String object.length ( );
- **Example:-**

```
class length
{
        public static void main(String[] args)
        {
                String s1=new String ("HELLO");
                System.out.print(s1.length());
        }
}
```

  - **OUTPUT:5**

❖ Special string operation

- **string literals**, **concatenation** of **multiple String** objects by use of the + operator, and the conversion of other data types to a string representation.
- **String literals**
  - For each string literal in your program, Java automatically constructs a **String** object.
  - Thus, you can use a string literal to initialize a **String** object.
  - **Example:-**

```
class literal
{
```

```
        public static void main(String[] args)
        {
                String s2 = "KAP"; // use string literal
                System.out.println("KAP".length());
        }
}
```

- **OUTPUT:3**
- Because a **String** object is created for every string literal, you can use a string literal any place you can use a **String** object.

- **String Concatenation(+ Operator)**
  - In general, Java does not allow operators to be applied to **String** objects.
  - The one exception to this rule is the **+** operator, which concatenates two strings, producing a **String** object as the result.
  - This allows you to chain together a series of + operations
  - **Example:**

```
class concat
{
        public static void main(String[] args)
        {
                String name = "Hello " + "World";
                System.out.println(name);
        }
}
```

❖ Character Extraction

- **charAt ( ) :-** To abstract single character from a string.
  - **Syntax** :- String object.chartAt (Position of character);
  - **Example**

```
class strcharat
{
        public static void main(String[] args)
        {
                String s1=new String("Hello");
                System.out.println(s1.charAt(3));
        }
}
```

- **getBytes( )**
  - There is an alternative to **getChars( )** that stores the characters in an array of bytes.
  - This method is called **getBytes( )**, and it uses the default character-to-byte conversions provided by the platform.
  - **Syntax:**
  - ✓ **byte[ ] getBytes( )**

- **getChars( )**

  - If you need to extract more than one character at a time, you can use the **getChars( )** method.
  - **Syntax**
    - ✓ **void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)**
    - ✓ sourceStart:- specifies the index of the beginning of the substring
    - ✓ sourceEnd:- specifies an index that is end of the desired substring.
    - ✓ The array that will receive the characters is specified by **target.**
    - ✓ The index within target at which the substring will be copied is passed in targetStart.
  - **Example:**

```
class getCharsDemo
{
        public static void main(String args[])
        {
                String name= "This is a demo of the getChars method.";
                int s=10;
                int e=14;
                char n[] = new char[e - s];
                name.getChars(s, e, n, 0);
                System.out.println(n);
        }
}
```

- **toCharArray( )**
  - If you want to convert all the characters in a **String** object into a character array, the easiest way is to call **toCharArray( )**.
  - It returns an array of characters for the entire string.
  - **Syntax:**
    - ✓ **char[ ] toCharArray( )**
  - This function is provided as a convenience, since it is possible to use **getChars( )** to achieve the same result.

❖ String comparison

- **equals ( )**
  - The equal's methods compare the character inside string of.
  - It returns true if the string contains same character in the same order & otherwise false.
  - The comparison is case sensitive.
  - **Syntax**
    - ✓ **String object1.equals (string object2);**
  - **Example:-1**

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

- **Example 2:-**

```
class equals1
{
        public static void main(String[] args)
        {
                String s1=new String ("Hello");
                String s2=new String ("Hello");
                if(s1.equals(s2))
                {
                        System.out.println ("strings are equal");
                }
                else
                {
                        System.out.println ("strings are not equal");
                }
        }
}
```

- **compare To( )**
  - It compares weather two string are less then, greater than or equal.
  - Returns negative if s1<s2 Returns positive if s1>s2 Returns zero if s1=s2.
  - **Syntax**
    - ✓ String object.compareTo (string object2);
  - **Example:-**

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

❖ Searching strings

- The **String** class provides two methods that allow you to search a string for a specified character or substring:
  - **indexOf( )** Searches for the first occurrence of a character or substring.
  - **lastIndexOf( )** Searches for the last occurrence of a character or substring.

- **indexOf ( )**
  - Gives the position of first occurrence of 'h' in the string s1.
  - **Syntax**
    - ✓ String object.indexOf('h');
  - **Example:-**

```
class indexof
{
    public static void main (String [] args)
    {
            String s1=new String ("Hello");
            System.out.println (s1.indexOf ('h'));
    }
}
```

❖ Modifying a string

- **substring( )**
  - You can extract a substring using **substring( )**. It has two forms.
  - **Syntax:**
    String substring(int startIndex)
    - ✓ Here, startIndex specifies the index at which the substring will begin.
    - ✓ This form returns a copy of the substring that begins at startIndex and runs to the end of the invoking string.
    - ✓ The second form of **substring( )** allows you to specify both the beginning and ending index of the substring:
    - ✓ **String substring(int startIndex, int endIndex)**
      - ♦ startIndex specifies the beginning index
      - ♦ endIndex specifies the stopping point.
  - **Example:-**

```
class substring
{
        public static void main(String[] args)
        {
                String s1=new String("Hello");
                System.out.println(s1.substring(2,5));
        }
}
```

- **concat ( )**
  - This method is use to concatenate two string s1 & s2.
  - **Syntax:**

✓ String object1.concat(string object2);

  - **Example:-**

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

- **replace ( )**
  - Replace method replaces all occurrence of one character in the string with another characters.
  - **Syntax**

✓ String object.replace (‘old character’,’ new character’);

  - **Example:-**

```
class replace
{
  public static void main(String[] args)
  {
        String s1=new String("Java");
        System.out.println(s1.replace('J','h'));
  }
}
```

- **trim( )**
  - Remove white space at beginning & end of the string.
  - **Syntax**
    - ✓ String object.trim( );
  - **Example:-**

```
class strtrim2
{
        public static void main(String[] args)
        {
                String s1=new String("      Hello");
                System.out.println (s1.trim());
        }
}
```

❖ toUpperCase ( )

- This method converts all character in a string from lower to upper.
- **Syntax:**
  String object.toUppercase();
- **Example:-**

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

❖ toLowerCase ( )

- This method converts all character in a string from upper to lower.
- **Syntax**
- String object.toLowercase();
- **Example:-**

```
class lower
{
    public static void main(String[] args)
    {
            String s1=new String ("HELLO");
            System.out.println (s1.toLowerCase());
    }
}
```

❖ valueOf( )

- String class valueOf method example:-
- This example demonstrates the working of valueOf method. this method returns String representations of all data types values
- **Syntax:-**
  - ✓ String **valueOf**(boolean b)
- **Java String class defines following methods to convert various Java primitives to Java String object.**
- **static String valueOf(int i)**
  - ✓ Converts argument int to String and returns new String object representing argument int.
- **static String valueOf(float f)**
  - ✓ Converts argument float to String and returns new String object representing argument float.
- **static String valueOf(long l)**
  - ✓ Converts argument long to String and returns new String object representing argument long.
- **static String valueOf(double i)**
  - ✓ Converts argument double to String and returns new String object representing argument double.

- **static String valueOf(char c)**
  - ✓ Converts argument char to String and returns new String object representing argument char.
- **static String valueOf(boolean b)**
  - ✓ Converts argument boolean to String and returns new String object representing argument boolean.
- **static String valueOf(Object o)**
  - ✓ Converts argument Object to String and returns new String object representing argument Object.
- **Example:-**

```
public class Test
{
    public static void main(String args[])
    {
            int i=10;
            float f = 10.0f;
            long l = 10;
            double d=10.0d;
            char c='a';
            boolean b = true;
            Object o = new String("Hello World");

            System.out.println( String.valueOf(i) );   // convert int to String
            System.out.println( String.valueOf(f) );   //convert float to String
            System.out.println( String.valueOf(l) );   // convert long to String
            System.out.println( String.valueOf(d) );  //convert double to String
            System.out.println( String.valueOf(c) );   //convert char to String
            System.out.println( String.valueOf(b) );   //convert boolean to String
            System.out.println( String.valueOf(o) );   //convert Object to String
    }
}
```

❖ String buffer

- String buffer is used to create modifiable(mutable) string.
- It allows character or substring to be inserted in the middle or append at the end.
- **length( )** : To find a length of string buffer.
  - **Syntax**
    - ✓ StringBufferobject.length();
  - **Example:-**

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

- **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.
  - **Syntax**
    - ✓ Stringbufferobject.Capacity();
  - **Example:-**

```
class capacityofstring
{
        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");
                System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2

        }
}
```

- **append()**
  - Append string s2 to s1 at the end.
  - **Syntax**
    - ✓ StringBufferobject.append.append();
  - **Example:-**

```
class A
{
        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
        }
}
```

- **insert ( )**
  - Insert the string s2 at the position of index s1.
  - 'N'=character of string of index.
  - **Syntax**
  - ✓ StringBufferobject.insert (index of charecter,"string");
  - **Example:-**

```
class insert
{
    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
    }
}
```

- **reverse( )**
  - You can reverse the characters within a **StringBuffer** object using **reverse( )**, shown here:
  - **Syntax:**
    - ✓ StringBuffer reverse( )
  - This method returns the reversed object on which it was called
  - **Example:-**

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

- **delete( )**
  - You can delete characters within a **StringBuffer** by using the methods **delete( )**
  - **Syntax:**
    - ✓ **StringBuffer delete(int startIndex, int endIndex)**
  - The **delete( )** method deletes a sequence of characters from the invoking object.
  - Here, startIndex specifies the index of the first character to remove, and endIndex specifies an index one past the last character to remove.
  - **Example:-**

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

- **replace( )**
  - You can replace one set of characters with another set inside a **StringBuffer** object by calling **replace( )**.
  - **Syntax:-**
    - ✓ **StringBuffer replace(int startIndex, int endIndex, String str)**
  - The substring being replaced is specified by the indexes startIndex and endIndex.
  - Thus, the substring at startIndex through endIndex–1 is replaced.
  - The replacement string is passed in str.
  - The resulting **StringBuffer** object is returned.
  - **Example:-**

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

```
        }
    }
```

❖ Vector Class

- Vector class is in java.util package of java.
- Vector is dynamic array which can grow automatically according to the required need.
- Vector does not require any fix dimension like String array and int array.
- Vector contains many useful methods.
- To add element in Vector, we can use add() method of vector class.
- To add elements at fix position, we have to use add(index, object) method.
- To get value from Vector, Vector provides get() method and Vector size() method.
- Size() method returns total number of elements in Vector.
- Vector is **synchronized**, ArrayList is not
- Syntax of some important method of **Vector** class. Given below.
  - **add(Object o) :** It adds the element in the end of the Vector.
  - **elements() :** It returns an enumeration of the element.
  - **elementAt(int index) :** It returns the element at the specified index.
  - **firstElement() :** It returns the first element of the vector.
  - **lastElement() :** It returns last element.
  - **removeElementAt(int index) :** It deletes the element from the given index.
  - **size() :** It returns total number of components available in vector.

- **Example:-1**

```
import java.util.*;
public class VectorExample
{
        public static void main(String[] args)
        {
                Vector<String> a=new Vector<String>();
                a.add("1");
                a.add("2");
                a.add("3");
                a.add("4");
                a.add("5");
                a.add(3, "Element at fix position");
                System.out.println("Vector Size :"+a.size());
                for(int i=0;i<a.size();i++)
                {
                    System.out.println("Vector Element "+a.get(i));
                }
                System.out.println("Vector Array:-"+a);
        }
}
```

- **Example-2:**

```
import java.util.Enumeration;
import java.util.Vector;

public class VectorExample3
{
    public static void main(String[] arr)
    {
        // creates a empty object of vector class
        Vector<Integer> a=new Vector<Integer>();
        // Add elements in vector
        a.add(1);
        a.add(2);
        a.add(3);
        a.add(4);
        a.add(5);
        System.out.println("Elements in vactor class.");
        // creates a enumeration object of vector class.
        Enumeration ob1=a.elements();
        while (ob1.hasMoreElements())
        {
            System.out.print(ob1.nextElement()+" ");
        }
    }
}
```

❖ **What is difference between string and string buffer?**

- Strings are immutable(unmodifiable). which means once u initialize the value for this? it can not be changed.
- StringBuffer is mutable(modifiable) and one can change the contents of it strings are immutable where as string buffer is a mutable, that means strings having the fixed size and stringbuffer size is varied(grow able).
- The significant performance difference between these two classes is that String Buffer is faster than String when performing simple concatenations.