UNIT-II: Array, String and I/O

INTRODUCTION:

- We know that, the concept of variable is introduced **to store the data**.
- But single variable can store only one value at a time, this is the drawback of variable. To overcome the drawback of single variable the concept of *Array* is introduced.
- That is *array* is also single variable but it has an ability to store multiple (more than one) value at a time.

Definition:

- "An Array is collection of elements or items having **same data type** referred by common name"
- **OR**
- "An Array is collection of homogeneous (having same type) elements or items referred by common name"

- In an array the individual element is accessed with the help of integer value and is called **subscript or index.** Also all elements of array are stored in **continuous memory** allocation.
- *Note that:*
  - *We know that, In C/C++ language, memory for array is get allocated at compile time (i.e. static memory allocation)*
  - *But, in JAVA everything is dynamic i.e. for variable, array, objects etc. memory is get allocated at run time (Dynamic memory allocation) by JVM*

Types of Array:

Depending on number of subscripts used in array, array having three types:

1) One Dimensional array: (1 D array)

The array having **only one subscript** is called as "*One dimensional array*"

Declaration Syntax:

```
datatype    array_name[ ]=new datatype [Size];
                     OR
datatype    [ ]array_name=new datatype [Size];
```

Here;
datatype is any valid *data type* in JAVA language
array_name is name of array which is an identifier.
'*size*' is integer value that denotes total number of elements stored in array
'*new*' is an operator.
e.g.

1) int x[ ]=new int[5];*//declares array 'x' & allocates memory for 5 integers*
   OR
   int [ ]x =new int[5];

   here ;
   'x' is array which can holds(stores) **five** integers at a time.

2) int marks[ ]; *//declares array 'marks'*
   marks=new int[5]; *//allocates memory for 5 integers*

Initialization of One dimensional array:

One dimensional array can be initialized as fallow;
e.g. int p[ ] = { 10 , 20 , 30 , 40 , 50 } ;

Here;
'p' is integer type array which stores five integers at a time. The storage of all elements in array 'p' is shown in following figure:

Index →	*0*	*1*	*2*	*3*	*4*
Elements of 'P'	**10**	**20**	**30**	**40**	**50**
Address →	110	114	118	122	126

In above figure the elements 10, 20, 30, 40, 50 are stored in array 'p' at individual index. That is the element 10 is stored at index 0, 20 is stored at index 1 like that, 50 is stored at index 4. Also all elements are stored in continuous memory location.

Note:

Index is integer value which is nothing but position of the element in an array.

2) Two Dimensional array: (2D array)

The array having **two subscripts** is called as "Two dimensional array or matrix"

The two dimensional array is used to perform matrix operations.

Declaration Syntax:

```
datatype        array_name[ ][ ]=new  datatype [Size1][Size2];
                                OR
datatype        [ ][ ]array_name=new  datatype [Size1][Size2];
```

Here;
datatype is any valid *data type* in Java language
array_name is name of array which is an identifier.
'*Size1'* is integer value that denotes total number of rows in array
'*Size2'* is integer value that denotes total number of columns in array
E.g.

1) int x[ ][ ]=new int[3][2];
   OR
   int [ ][ ] x=new int[3][2];

here ;
'*x'* is array which can holds total 6 integers at a time.
3 is *rowsize* i.e. there are 3 rows in matrix 'x'
2 is *columnsize* i.e. there are 2 columns in matrix 'x'

Initialization of Two dimensional array:

Two dimensional array can be initialized as follow;

e.g. 1) int p[ ][ ]={{3,4},{2,9},{7,6}};

```
2)      int         z[ ][ ] ={  {  5, 3 , 6 } ,
                                {  1, 7 , 8 } ,
                                {  9, 4 , 2 }  };
```

Here;
'z' is integer type two dimensional array which stores nine integers at a time. The storage of all elements in array 'z' is shown in following figure:

row index ↓	**0**	**1**	**2** ← column index
0	5	3	6
1	1	7	8
2	9	4	2

In above figure the individual element of array ‘z’ is also accessed by following way:

The element 8 can be accessed as z[1][2]
The element 7 can be accessed as z[1][1]
The element 9 can be accessed as z[2][0]   etc.

like that we can access all individual elements of matrix ‘z’

<u>3) Multi-Dimensional array:</u>

“The array having **more than two subscripts** is called as *multi-dimensional* array”

<u>Declaration Syntax:</u>

```
datatype array_name[ ][ ]……[ ]=new datatype [Size1][Size2]……[SizeN];
                                OR
datatype [ ][ ]……[ ]array_name=new datatype [Size1][Size2]……[SizeN];
```

here;

datatype is any valid *datatype* in Java language.
array_name is an identifier (name given by programmer)
size1, size2,………sizeN are **integer constants** and that denotes total number of elements stored in an array.

e.g.   1)   int   z[][][]=new int[2]**[4][3]**;

Above array is the multidimensional array having three subscripts and which stores 12 integer values at a time.

*That is above multi-dimensional array stores **2 sets of 4*3 matrices**.*

2)   float   x[ ][][][]=new float[2][3][2][2];

The above array ‘x’ is also multidimensional array having four subscripts and which stores 24 float values at a time.

<u>Initialization of Multi-dimensional array:</u>

Multi-dimensional array can be initialized as follows:

Example:

1) int x[][][]={{{1,2},{4,5},{7,8},{3,6}},
            {{11,12},{14,15},{17,18},{13,16}} };

In above multi-dimensional array, elements are stored in following manner;

X[0] ⇨

	0	*1*
0	**1**	**2**
1	**4**	**5**
2	**7**	**8**
3	**3**	**6**

x[1] ⇨

	0	*1*
0	**11**	**12**
1	**14**	**15**
2	**17**	**18**
3	**13**	**16**

From above, multi-dimensional array we can access individual element as follow:

```
X[0][1][0]   access  4
X[0][2][1]   access  8
X[0][3][0]   access  3
X[1][1][1]   access  15
X[1][2][0]   access  17
X[1][3][1]   access  16
```

'length' property of array:

- 'length' is property of an array *returns one integer value which is nothing but* ***size of array***.
- We use this property as follows:

```
int     var = arrayname.length;
```

Here, 'var' is an *int* type variable which stores size of array return by *arrayname.length* poperty.

Exampl:

```
1) int   arr[ ]=new  int[10];     //declares array with 10 size
          arr[0]=45;
          arr[1]=90;
   int    sz=arr.length;       //returns array size
   System.out.print("Array Size="+sz);

         OUTPUT:
                    Array Size=10
```

Note that:

- In above example 'arr' is array declared with 10 size. And arr[0] and arr[1] are initialized with values 45 and 90 respectively. But '*arr.length*' statement returns value 10 which is size of array (Since, *'length' property returns* ***size of array****. It* ***not*** *returns total number of array elements*)
- *Also, in case of 2D or Multi-dimensional array, 'length' property returns* ***number of rows of the array****.*

Following program shows use of 'length' property of array.

```
public   class   Myarr
{
   int     arr[]=new   int[5];
   int     arr1[][]=new  int[2][3];
   int     arr2[][][]=new   int[4][5][6];
   void get()
   {
    int   i=arr.length;
    int   j=arr1.length;
    int   k=arr2.length;
    System.out.println("Size of 1D="+i);
    System.out.println("Size of 2D="+j);
    System.out.println("Size of MD="+k);
   }
  public static void main(String []ar)
  {
    Myarr  p=new  Myarr();
    p.get();
  }

}
OUT PUT:
              Size of 1D=5
              Size of 2D=2
              Size of MD=4
```

STRING

Introduction:

- String is nothing but collection or group of characters.
- We know that, in case of C/C++ language string is nothing but *'array of characters'* & that terminates with '\0' character i.e. NULL character. But this is not true in Java language.
- In JAVA, *"String is nothing but Object of String class*" and which is not array of character. Also, in JAVA we can create array of character but it is not treated as string.
- While dealing with string, JAVA language has special class called "String" class which was found under "java.lang" package.

Creating Strings in Java:

We can create Sting in Java by three ways:

- We can create a String by assigning group of characters to a String type object:

```
String    str;    //declare string
str= "Hello";      //assign a group of characters to it
Above two statements can be combined as follow:
String    str= "Hello";
In this case, JVM creates an string reference & stores the string "Hello"
```

- We can create an object of String class by using 'new' operator:

```
String    str = new    String("Hello");
```

  In this case, First we create object '*str'* using *new* operator and then we store "Hello" string in it.

- Third way of creating String is by converting array of character into String:

```
char    arr[]={'H','e','l','l','o'};//create character array.
String    str=new    String(arr);//creating string 'str' by passing 'arr' to it.
```

Difference between object & Reference:

Object	Reference
1) The object is created using 'new' operator. e.g. String str=new String( );	1) Reference is not created using 'new'operator. e.g. String str;
2) Such type of object is stored in 'Heap' section by 'class loader sub system' of JVM.	2) Such type of referece is stored in 'Method area' section by 'class loader sub system' of JVM.
3) When such object is created then, Constructor is implicitly invoked.	3) When such reference is created then, Constructor is NOT implicitly invoked.
4) When such object is created then JVM allocated separate memory location to each object.	4) When such reference is created then JVM insert it into "*String constant pool*" (Special memory block where String references are stored)

String Class Methods:

While dealing with Strings, there are several methods belong to String class:

Note that: Following methods of String class are non-static therefore they are called with object of String class.

1) length ( ) :

- This method is used to find length of string.
- This method returns one integer value which is length (total characters) of string.

Syntax:

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

Here, *'s'* is an object or reference of String class and that contains the string.
'len' is integer variable used to store length of string.

E.g.

```
class        stringLen
{
    public static void main(String args[ ])
    {
            String    str=new    String("Hello");
            intlen=str.length( );
            System.out.println("Length="+len);
    }
}
OUTPUT: Length=5
```

2) concat ( ) :

- This method is used to concatenate two strings together.
- That is, it concatenates second string at the end of first string and returns concatenated string as a result.

Syntax:

```
String     str= s1.concat(s2);
```

Here, ‘s1’ is an object or reference of String class and that contains the first string.
‘s2’ is also object or reference of String class and that contains the second string.
‘str’ is also string that stores concatenated string.

E.g.

```
class     stringCat
{
      public static void main(String   args[ ])
      {
                  String    s1= “Delhi”;
                  String    s2= “Mumbai”;
                           String    str=s1.concat(s2);
                  System.out.println(str);
      }
}
OUTPUT:              DelhiMumbai
```

3) charAt ( ) :

- This method accepts one integer value and returns one character from string corresponding to passed integer value
- That is,it accepts index value (position of element) and returns corresponding character from string.

Syntax:

```
char      ch= s.charAt(pos);
```

Here, ‘s’ is an object or reference of String class and that contains the string.
‘pos’ is an integer value.
‘ch’ is char variable which stores returned character from string at position ‘*pos’*

E.g.

```
class        stringChar
{
      public static void main(String   args[ ])
      {
                  String    s= “Delhi”;
                  char  ch= s.charAt(3);
                  System.out.println(ch);
      }
}
OUTPUT:
                  h
```

4) equals( ) :

- This method is also used to compare two strings with each other for equality.
- This method returns boolean value depending upon Strings content.
- It returns boolean value ‘true’ if both strings are equal otherwise it returns ‘false’.

Syntax:

```
boolean      m = s1.equals(s2);
```

Here, ‘s1’ is an object or reference of String class and that contains first string.
‘s2’ is also an object or reference of String class and that contains second string.
‘m’ is boolean variable to store returned value.

E.g.

```
class          stringEqual
{
     public static void main(String   args[ ])
     {
               String    s1= “box”;
               String    s2= “BOX”;
               boolean  m= s1.equals(s2);
               if (m = = true)
                         System.out.print(“Both strings are equal”);
               else
                         System.out.print(“Strings are not equal”);
     }
}
OUTPUT:    Strings are not equal
```

5) equalsIgnoreCase( ) :

- This method is also used to compare two strings with each other for equality. But it ignores the case of characters present in strings. i.e. it treated “box” string same as “BOX”
- This method returns *boolean* value depending upon Strings contain.
  - ❖ It returns boolean value ‘true’ if both strings are equal otherwise it returns ‘false’.

Syntax:

```
boolean     m = s1.equalsIgnoreCase(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.
‘s2’ is also an object of String class and that contains second string.
‘m’ is boolean variable to store returned value.

E.g.

```
class       stringEqual
{
     public static void main(String   args[ ])
     {
               String    s1= “box”;
               String    s2= “BOX”;
               boolean  m= s1.equalsIgnoreCase(s2);
               if (m = = true)
                         System.out.print(“Both strings are equal”);
               else
                         System.out.print(“Strings are not equal”);
     }
}
OUTPUT:   Both Strings are equal
```

6) compareTo ( ) :

- This method is used to compare two strings with each other for equality.
- This method returns one integer value depending upon Strings contain.
  - ❖ It returned value is Zero then both strings are equal.
  - ❖ If returned value is positive then first string is greater than second string.
  - ❖ If returned value is negative then Second string is greater than first string.

Syntax:

```
int     m = s1.compareTo(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.

‘s2’ is also an object of String class and that contains second string.

‘m’ is integer variable to store returned value.

E.g.

```
class             stringCmp
{
    public static void main(String   args[ ])
    {
            String    s1= “abc”;
            String    s2= “abd”;
            int  m= s1.compareTo(s2);
            if (m = = 0)
                    System.out.print(“Both strings are equal”);
            else  if( m > 0)
                    System.out.print(“First string is greater”);
            else
                    System.out.print(“Second string is greater”);
    }
}
OUTPUT:         Second string is greater
```

7) compareToIgnoreCase( ) :

- This method is also used to compare two strings with each other for equality. But it ignores the case of characters present in strings. i.e. it treated “box” string same as “BOX”
- This method returns one integer value depending upon Strings contain.
    - ❖ If returned value is Zero then both strings are equal.
    - ❖ If returned value is positive then first string is greater than second string.
    - ❖ If returned value is negative then Second string is greater than first string.

Syntax:

```
int     m = s1.compareToIgnoreCase(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.
‘s2’ is also an object of String class and that contains second string.
‘m’ is integer variable to store returned value.

E.g.

```
class         stringCmp
{
    public static void main(String   args[ ])
    {
            String    s1= “box”;
            String    s2= “BOX”;
            int  m= s1.compareToIgnoreCase(s2);
            if (m = = 0)
                    System.out.print(“Both strings are equal”);
            else  if( m > 0)
                    System.out.print(“First string is greater”);
            else
                    System.out.print(“Second string is greater”);
    }
}
OUTPUT:
             Both strings are equal
```

8) startsWith( ) :

- This method returns *boolean* value ‘true’ if first string starts with second string otherwise it returns ‘false’.

Syntax:

```
boolean     m = s1.startsWith(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.
‘s2’ is also an object of String class and that contains second string.
‘m’ is boolean variable to store returned value.

E.g.

```
class           stringStart
{
    public static void main(String   args[ ])
    {
            String    s1= “Box is heavy”;
            String    s2= “Box”;
            boolean  m= s1.startsWith(s2);
            if (m = = true)
                    System.out.print(“First string start with second string”);
            else
                    System.out.print(“First string NOT start with second string”);
    }
}
OUTPUT: First string start with second string
```

9) endsWith( ) :

- This method returns boolean value ‘true’ if first string ends with second string otherwise it returns ‘false’.

Syntax:

```
boolean     m = s1.endsWith(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.
‘s2’ is also an object of String class and that contains second string.
‘m’ is boolean variable to store returned value.

E.g.

```
class          stringEnd
{
    public static void main(String   args[ ])
    {
            String    s1= “Box is heavy”;
            String    s2= “heavy”;
            boolean  m= s1.startsWith(s2);
            if (m = = true)
                    System.out.print(“First string ends with second string”);
            else
                    System.out.print(“First string NOT ends with second string”);
    }
}
OUTPUT: First string ends with second string
```

10) indexOf( ) :

- This method returns integer value which is nothing but first position of substring into main string.
- If substring is not found in main string then it returns **-1** negative value.

Syntax:

```
int    m = s1.indexOf(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.
‘s2’ is also an object of String class and that contains second string.
‘m’ is integer variable to store returned value.

E.g.

```
class          stringFirstIndex
{
    public static void main(String   args[ ])
    {
            String    s1= “Box is heavy and it is dirty”;
            String    s2= “is”;
            int m= s1.indexOf(s2);
            if (m < 0 )
                    System.out.print(“Substring Not found”);
            else
                    System.out.print(“Substring found at= ”+m);
    }
}
OUTPUT: Substring found at= 4
```

11) lastIndexOf( ) :

- This method returns integer value which is nothing but <u>last position </u>of substring into main string.
- If substring is not found in main string then it returns <u>**-1** negative</u> value.
- Syntax:

```
int    m = s1.lastIndexOf(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.
‘s2’ is also an object of String class and that contains second string.
‘m’ is integer variable to store returned value.

E.g.

```
class            stringLastIndex
{
    public static void main(String   args[ ])
    {
            String    s1= “Box is heavy and it is dirty”;
            String    s2= “is”;
            int m= s1.lastIndexOf(s2);
            if (m < 0 )
                    System.out.print(“Substring Not found”);
            else
                    System.out.print(“Substring found at= ”+m);
    }
}
OUTPUT: Substring found at= 20
```

12) replace( ) :

- This method replaces characters of existing string <u>with new given character</u>.

Syntax:

```
String    str = s.replace(old_char,new_char);
```

Here, ‘old_char’ is character to be replaced by ‘new_char’ of string.
‘str’ is String variable to store returned value.

E.g.

```
class          stringReplace
{
    public static void main(String   args[ ])
    {
            String    s= “Box is heavy”;
            String    str= s.replace(‘B’,‘D’);
            System.out.print(str);
    }
}
OUTPUT: Dox is heavy
```

13) substring( ) :

- This method has two forms:-

I) substring(int):

- This method returns a new string consisting of all characters from *given position* to the *end of string*.

Syntax:

```
String    str = s.substring(pos);
```

Here, ‘s’ is an object of String class and that contains string.
‘pos’ is an integer value from which new string is to be started.
‘str’ is string object to store resultant string.

E.g.

```
class          stringSubstr
{
     public static void main(String   args[ ])
     {
               String    s= “Box is heavy”;
               String    str= s.substring(4);
               System.out.print(str);
     }
}
OUTPUT:   is heavy
```

II) substring(int,int):

- This method returns a new string consisting of all characters from *given* ***first position*** to *the* ***last position -1***.

Syntax:

```
String    str = s.substring(pos1,pos2);
```

Here, ‘s’ is an object of String class and that contains string.
‘pos1’ is an integer value i.e. *first position*from which new string is to be started.
‘pos2’ is an integer value i.e. *last position*and new string ends at *last postion-1*.
‘str’ is string object to store resultant string.

E.g.

```
class    stringSubstr1
{
     public static void main(String   args[ ])
     {
               String    s= “Box is heavy”;
               String    str= s.substring(7,11);
               System.out.print(str);
     }
}
OUTPUT:   heav
```

14) toLowerCase( ) :

- This method converts all characters of string into *lower case* and returns that lower-cased string as result.

Syntax:

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

Here, ‘s’ is an object of String class and that contains string.
‘str’ is also an object of String that stores returned lower-cased string.

E.g.

```
class            stringLower
{
     public static void main(String   args[ ])
     {
               String    s= “BOX IS HEAVY”;
               String    str= s.toLowerCase( );
               System.out.print(str);
     }
}
OUTPUT:   box is heavy
```

15) toUpperCase( ) :

- This method converts all characters of string into *upper case* and returns that upper-cased string as result.

Syntax:

```
String    str = s.toUpperCase();
```

Here, ‘s’ is an object of String class and that contains string.
‘str’ is also an object of String that stores returned upper-cased string.

E.g.

```
class              stringUpper
{
    public static void main(String   args[ ])
    {
            String    s= “box is heavy”;
            String    str= s.toUpperCase( );
            System.out.print(str);
    }
}
OUTPUT:   BOX IS HEAVY
```

16) getChars( ) :

- This method copies characters from *string into character array*.
- Characters copied into character array from starting position ‘*pos1’* up to last position ‘*pos2-1*’ to location starting from ‘pos3’ in a character array.

Syntax:

```
s.getChars(pos1,pos2,arr,pos3);
```

Here, ‘s’ is an object of String class and that contains string.
‘pos1’ is an integer value represents *starting position* from which string copied.
‘pos2’ is an integer value represents *ending position*& string copied ends at <u>*pos2-1*</u> index.
‘arr’ is character array that stores copied characters from string ‘s’.
‘pos3’ is an integer value from which character array ‘arr’ stores the copied character.

E.g.

```
class           stringGetchar
{
    public static void main(String   args[ ])
    {
            String    s= "Box is heavy";
            char [ ]arr=new char[20];
            s.getChars(7,11,arr,0);
            for(char i:arr)
            {
               System.out.print(i);
            }
    }
}
OUTPUT:   heav
```

17) trim( ) :

- This method removes unwanted i.e. extra spaces which were found ***before and after*** the string.
- *Note that:*This method does <u>*not remove extra spaces between two words*</u> of string.

Syntax:

```
String    str = s.trim( );
```

Here, ‘s’ is an object of String class and that contains string.
‘str’ is also object of String class used to store resultant string.

E.g.

```
class    stringTrim
{
      public static void main(String   args[ ])
      {
                  String    s= "              Box is heavy           ";
                  String    str = s.trim( );
                  System.out.print(str);
      }
}

OUTPUT:   Box is heavy
```

18) split( ) :

- This method splits or cuts the given string into number of pieces (sub strings) corresponding to delimiter (specified character) and store it into array of string.

Syntax:

```
String  [  ] str = s.split(delimiter);
```

Here, ‘s’ is an object of String class and that contains string.
‘delimiter’ is string at which we specify splitting character
‘str’ is array of String that stores splitted strings at individual indices.

E.g.

```
class           stringSplit
{
      public static void main(String   args[ ])
      {
                  String    s= "Box is heavy,dirty,red and useless";
                  String [ ]str=s.split(",");
                  for(String i: str)
                  {
                              System.out.println(i);
                  }
      }
}
OUTPUT:
            Box is heavy
            dirty
            red and useless
```

String Comparison:

- We know that for comparison of two or more quantities with each other, we use relational operators like <, >, <=, >=, !=, ==.
- But for string comparison, relational operators are *not* suitable because these operators *compare reference (address) of object with each other*. They ***not** compare strings contents*.
- For that purpose, compareTo() or equals() methods are used to compare two strings. And *these functions compares strings contents for equality*.
  Consider following example;

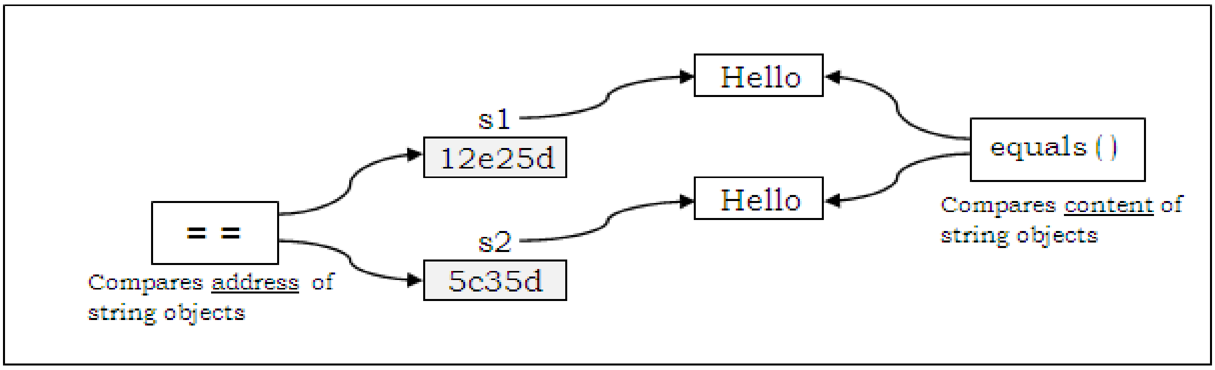
```
class          stringCmp
{
      public static void main(String   args[ ])
      {
                  String    s1=new    String(“Hello”);
                  String    s2=new    String(“Hello”);
                  if(s1= = s2)
                              System.out.println(“Strings are same”);
                  else
                              System.out.println(“Strings are NOT same”);
      }
}
OUTPUT:   Strings are NOT same
```

- In above example, we got output “Strings are NOT same” still content of strings are same. This is due to relational operator (= =) because it compares address (*address is in Hexadecimal number*) of both strings, it not compares strings content.
- When an object is created by JVM, it returns the memory address of the object as a hexadecimal number which is called “*object reference*” and this *object reference is separate for every object* i.e. whenever an abject is created, a new address number is allotted to it by JVM.
  Now, consider another example:

```
class       stringCmp
{
    public static void main(String   args[ ])
    {
            String    s1=new    String(“Hello”);
            String    s2=new    String(“Hello”);
            if(s1.equals(s2)==true)
                    System.out.println(“Strings are same”);
            else
                    System.out.println(“Strings are NOT same”);
    }
}

OUTPUT:   Strings are same
```

- In above example, we got output “Strings are same” which is right because *equals( )* method compares strings content, *not their addresses*.
  Above mentioned concepts shown in following figures:



s1
12e25d
Hello
Hello
equals ( )
s2
5c35d
= =
Compares content of string objects
Compares address of string objects

Now, consider One another example: There is slightly change in object creation of String. Here, object of String is created without using ‘new’ object:

```
class       stringCmp
{
    public static void main(String   args[ ])
    {
            String    s1=“Hello”;
            String    s2=“Hello”;
            if(s1==s2)
                    System.out.println(“Strings are same”);
            else
                    System.out.println(“Strings are NOT same”);
    }
}
OUTPUT:   Strings are same
```

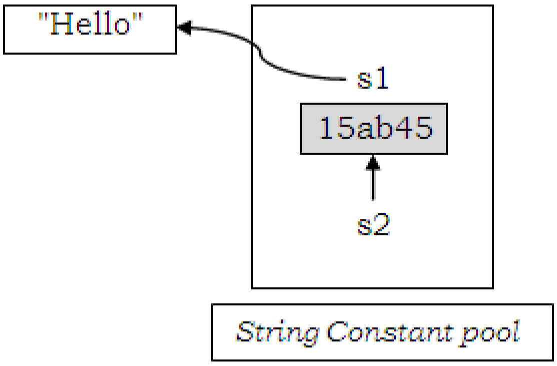
- In above example, we got output “Strings are same”. Still we are using relational operator (= =) for string comparison. How it is possible?

→

- ➢ Here, String objects are created without using ‘new’ operator at that time JVM uses separate memory block which is called *‘string constant pool’* and such objects are store there.
- ➢ When first statement i.e. String *s1= “Hello”* is executed then JVM insert object ‘*s1*’ into ‘*string constant pool’* with separate address.

- ➢ And when second statement i.e. String *s2 “Hello”* is executed then JVM, searches in ‘*string constant pool*’ to know whether the object with same content is already exist there or not?
  - ❖ If any object with same content is *found* in ‘*string constant pool*’ then *JVM just attach second object at reference (address) of first object*.
  - ❖ If any object with same content is *NOT found* in ‘*string constant pool*’ then JVM *allocate separate reference (address) to second object.*
- ➢ Here, content of String ‘s1’ and ‘s2’ are same therefore JVM just attach ‘s2’ at address of ‘s1’. And that’s made both objects addresses are same. And hence, we got output “Strings are same”

Above mentioned concept is shown in following fig.



Immutability of String:

- We know that, an ‘object’ is basic runtime entity that holds data or some content.
- And every Object can broadly classified into two categories viz: 1) Mutable object 2) Immutable object.

Let’s see them in details:

1) Mutable Object:

- The objects whose *content* can be changeable or modifiable are called ‘Mutable’ objects.

2) Immutable Object:

- The objects whose content can NOT be changeable or modifiable are called ‘Immutable’ Objects.
- And Object of String class is Immutable i.e. we cannot modify the content of String object. And hence, we can say that **‘String’ Class is Immutable class**.

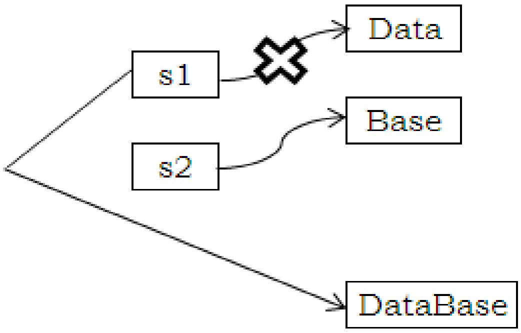
Consider following example to test immutability of String.

```
class      myString
{
    public static void main(String   args[ ])
    {
            String   s1=“Data”;
            String   s2=“Base”;
            s1=s1+s2;
            System.out.println(s1);
    }
}

OUTPUT: DataBase
```

- In above example we got output “DataBase”. It seems that, the content of ‘s1’ is modified. Because, earlier ‘s1’ has content “Data” and ‘s2’ has content “Base”. And after ‘s1+s2’ they are joined together and total string becomes “DataBase”. This result is assigned to ‘s1’. If ‘s1’ is mutable then it gets new string “DataBase” and we also got “DataBase” as result.
- But ‘s1’ is object of String class & we learned that ‘String objects are immutable’, then how we got result “DataBase”?

➔ Definitely String object ‘s1’ is immutable. Consider following figure that will explain this concept briefly:



- In the program, JVM creates two objects ‘s1’ and ‘s2’ separately, as shown in above figure.
- When ‘s1+s2’ is done then JVM creates new object and stores ‘DataBase’ in that created object.
- After creating new object, the reference ‘s1’ is just attached or assigned to newly created object.
- Remember, it ***does not modify existing content of ‘s1’***. And that’s why we are called *‘String objects are immutable.*’
- The old object that contains ‘Data’ has lost its reference. So it is called ‘*unreferenced object’* and garbage collector remove it from memory.

StringBuffer And StringBuilder

StringBuffer:

- In previous chapter, we learnt that Strings are immutable and cannot modified. To overcome this, Java language provide new String handling class called ‘StringBuffer’ which represents strings in such a way that their data can be modified.
- It means ‘StringBuffer’ objects are *mutable* i.e. we can modified their data. Also, the methods present in this class that directly manipulate and change the data of object very easily.
- *Also, there are several classes present in Java which are immutable such as Character, Byte, Integer, Float, Double, Long. . . . etc wrapper classes are immutable.*

Creating StringBuffer Object:

There are three ways of creating *StringBuffer* object and fill that object with string:

I) Creating s*tringBuffer* object by using *new* operator and pass String to that object:

```
StringBuffer     sb=new     StringBuffer(“Hello”);
```

Here, we are passing “Hello” string to *stringBuffer* object ‘sb’.

II) Creating *StringBuffer* object first using new operator & then storing string to created object:

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

Here, we create *StringBuffer* object ‘sb’ as an empty object and not passing any string to it.
In this case, a *StringBuffer* object will be created with *default capacity of 16 characters*.
After creating *StringBuffer* object, we can store string to created object using method append() or insert().
E.g.   sb.append(“Hello”);

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

Here, we create *StringBuffer* object ‘sb’ as an empty object with capacity of 50 characters.
But, we can store *more than 50 characters to ‘sb’ object because StringBuffer objects are mutable and can expand dynamically*.

III) First create object of String that holds one string and then pass that object to *StringBuffer*:

```
String        str= “Hello”;
StringBuffer    sb=new    StringBuffer(str);
```

Here, we create String object ‘str’ that has string “Hello” and pass it to *StringBuffer* object ‘sb’.
In this case, *StringBuffer* object stores data of *str i.e. “Hello”*

StringBuffer Class Methods:

There are several methods belonging to *StringBuffer* class and they are as follow:
Note that: Following methods of StringBuffer class are non-static therefore they are called with <u>*object*</u> *of StringBuffer class.*

1) append ( ) :

- This method is used to *append* given string into object of *StringBuffer* class.

Syntax:

```
sb.append(x);
```

Here, *'sb'* is an object of *StringBuffer* class.
*'x'*is argument accept by append( ) method that may be boolean, int, long, float, char, String or another *StringBuffer*.

E.g.

```
class     datastr
{
    public static void main(String args[ ])
    {
            StringBuffer     sb=new    StringBuffer("Uni");
            sb.append("versity");
            System.out.println(sb);
    }
}
OUTPUT:          University
```

2) insert ( ) :

- This method is used to *inserts* given string into object of *StringBuffer* class from given position.

Syntax:

```
sb.insert(pox,x);
```

Here, *'sb'* is an object of *StringBuffer* class.
'x' is argument accept by append( ) method that may be boolean, int, long, float, char, String or another *StringBuffer*.
'*pos'* is integer value which is position at which new string will be inserted.

E.g.

```
class     Mydemo
{
    public static void main(String args[ ])
    {
            StringBuffer     sb=new    StringBuffer("India Is My Country");
            sb.insert(9,"Great");
            System.out.println(sb);
    }
}
OUTPUT:          India Is GreatMy Country
```

In above program, "Great" string is inserted into '*sb'* object from index 9.

3) delete ( ) :

- This method is used to *deletes all the characters from first position 'i' to second postion 'j-1' p*osition.

Syntax:

```
sb.delete(i,j);
```

Here, *'sb'* is an object of *StringBuffer* class.
'*i'* is integer value which is first position
*'j'*is integer value which is second position.

E.g.

```
class      Mydemo
{
    public static void main(String     args[ ])
    {
            StringBuffer        sb=new    StringBuffer("University");
            sb.delete(3,6);
            System.out.println(sb);
    }
}
OUTPUT:           Unisity
```

In above program, delete( ) methoddeletes all character form position 3 to 5.

4) reverse ( ) :

- This method is used to *reverse all the characters of StringBuffer*object.

Syntax:

```
sb.reverse( );
```

Here, *'sb'* is an object of *StringBuffer* class that contains the string.

E.g.

```
class       Mydemo
{
    public static void main(String     args[ ])
    {
            StringBuffer         sb=new    StringBuffer("ABC");
            sb.reverse( );
            System.out.println(sb);
    }
}
OUTPUT:           CBA
```

5) length ( ) :

- This method is used to find number of characters present in *StringBuffer* object.
- This method returns one integer value which is total characters stored in *StringBuffer*.

Syntax:

```
intlen = sb.length( );
```

Here, *'sb'* is an object of *StringBuffer* class and that contains the string.
'len' is integer variable used to store length.

E.g.

```
class          stringLen
{
    public static void main(String args[ ])
    {
            StringBuffer      sb=new    StringBuffer("Hello");
            intlen=sb.length( );
            System.out.println("Length="+len);
    }
}
OUTPUT:
            Length=5
```

6) indexOf( ) :

- This method returns integer value which is nothing but <u>first position </u>of substring into *StringBuffer* object.
- If substring is not found in *StringBuffer* object then it returns **-1** <u>negative</u> value.

Syntax:

```
int    m = sb.indexOf(str);
```

Here, ‘sb’ is an object of *StringBuffer* class and that contains first string.
‘str’ is sub string .
‘m’ is integer variable to store returned value.

E.g.

```
class         stringFirstIndex
{
    public static void main(String   args[ ])
    {
            StringBuffer      sb=new  StringBuffer("Box is heavy and it is dirty");
            int     m= sb.indexOf("is");
            if (m < 0 )
                    System.out.print("Substring Not found");
            else
                    System.out.print("Substring found at= "+m);
    }
}
OUTPUT: Substring found at= 4
```

7) lastIndexOf( ) :

- This method returns integer value which is nothing but last position of substring into *StringBuffer* object.
- If substring is not found in *StringBuffer* object then it returns **-1** negative value.

Syntax:

```
int    m = sb.lastIndexOf(str);
```

Here, ‘sb’ is an object of *StringBuffer* class and that contains first string.
‘str’ is sub string.
‘m’ is integer variable to store returned value.

E.g.

```
class            stringLastIndex
{
    public static void main(String   args[ ])
    {
            StringBuffer      sb=new    StringBuffer(“Box is heavy and it is dirty”);
            int m= s1.lastIndexOf(“is”);
            if (m < 0 )
                    System.out.print(“Substring Not found”);
            else
                    System.out.print(“Substring found at= ”+m);
    }
}
OUTPUT: Substring found at= 20
```

8) replace( ) :

- This method replaces characters from ‘i' to ‘j-1’ position by given string of *StringBuffer* object

Syntax:

```
sb.replace(i,j, “str”);
```

Here, ‘i’ is first position from which character replaces
‘j’ is second position and up to “j-1” position character are replaced by string “str”

E.g.

```
class         stringReplace
{
    public static void main(String   args[ ])
    {
            StringBuffer      sb=new StringBuffer("Box is heavy");
            sb.replace(0,3,"zzz");
            System.out.print(sb);
    }
}
OUTPUT: zzzis heavy
```

9) substring( ) :

- This method has two forms:-

I) substring(int):

- This method returns a new string consisting of all characters from *given position* to the *end of string from StringBuffer object*.

Syntax:

```
String    str = sb.substring(pos);
```

Here, 'sb' is an object of *StringBuffer* class and that contains string.
'pos' is an integer value from which new string is to be started.
'str' is string object to store resultant string.

E.g.

```
class          stringSubstr
{
    public static void main(String   args[ ])
    {
            StringBuffer         sb=new StringBuffer(“Box is heavy”);
            String    str= sb.substring(4);
            System.out.print(str);
    }
}
OUTPUT: is heavy
```

II) substring(int,int):

- This method returns a new string consisting of all characters from *given first position* to *the* ***last position -1*** from *StringBuffer* object.

Syntax:

```
String    str = sb.substring(pos1,pos2);
```

Here, 'sb' is an object of StringBuffer class and that contains string.
'pos1' is an integer value i.e. *first position* from which new string is to be started.
'pos2' is an integer value i.e. *last position* and new string ends at *last postion-1*.
'str' is string object to store resultant string.

E.g.

```
class    stringSubstr1
{
    public static void main(String   args[ ])
    {
            StringBuffer          sb=new StringBuffer("Box is heavy");
            String    str= sb.substring(7,11);
            System.out.print(str);
    }
}
OUTPUT: heav
```

StringBuilder Class:

- *StringBuilder* class has been added in JDK 1.5 which has same features of *StringBuffer* class.
- *StringBuilder* class objects are also mutable like *StringBuffer* class i.e. modification is allowed on object of *StringBuilder* class.

We can create object of *StringBuilder*class as follows:

❖ StringBuilder   sb=new StringBuilder(“Hello”);
❖ StringBuilder   sb=new StringBuilder( );
❖ StringBuilder   sb=new StringBuilder(50);

StringBuilder Class methods:

The following are the important methods of *StringBuilder* class whose functionality is same as *StringBuffer* class:

StringBuilder	append(x)
StringBuilder	insert(i,x)
StringBuilder	delete(i,j)
StringBuilder	reverse( )
String	toString( )
int	length( )
int	indexOf(String str)
int	lastIndexOf(String str)
StringBuilder	replace(i,j,Str)
String	substring(i);
String	substring(i,j);

Difference between *StringBuffer* and *StringBuilder* class:

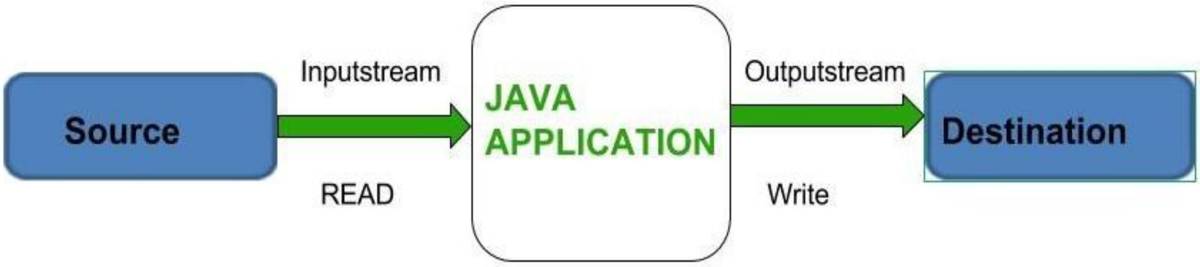
StringBuffer	StringBuilder
1) *StringBuffer* class is synchronized	1) *StringBuilder* class is not synchronized
2) This class is synchronized therefore multiple threads *cannot* act simultaneously onto object of *StringBuffer* class. i.e. threads act on object one after another.	2) This class is not synchronized therefore multiple threads *acts simultaneously onto object* of *StringBuilder* class. i.e. threads act on object at one time.
3) In this case, Threads act on single object one after another that gives reliable or correct result.	3) In this case, multiple threads act on single object at one time that gives unreliable or undependable or inaccurate result.
4) *StringBuffer* class will take more execution time. Because, when one thread acts on object at that time other threads are in waiting stage. After first thread completes its working then another thread start it work such process is happen for every thread.	4) *StringBuilder* class will take less execution time than *StringBuffer* class. Because, at one time multiple threads will act on same object of *StringBuilder* class. Here, no question for waiting of thread.

Input / Output Handling

- We know to solve any problem, we need an input and output operations. To perform input and output (I/O) in Java, it has *java.io* package that contains the classes which performs I/O.
- Such I/O operations done via streams that represents an input source and an output destination. The stream in the java.io package supports many data such as primitives, object, localized characters, etc.
- Java Program performs input / output through stream. So let's understand stream.

Stream:

- A stream can be defined as a sequence of bytes (data) which flows from input device to output device and vice-versa.
- There are two kinds of Streams:
- **InPutStream**: The InputStream is used to read data from a source i.e. from keyboard, file etc.
- **OutPutStream:** The OutputStream is used for writing data to a destination i.e. to monitor, file etc.
- **Following diagram shows I/O stream in Java-**



- To perform both Input stream and Output stream operations, Java language has two classes viz.-

1) Byte Stream (Binary Stream) class
2) Character Stream (Text stream) class

Before learning these classes, we have to know file and its type which is discussed below-

File:

- File is a place on secondary storage (Hard disk, CD, DVD, Pendrive etc.) where large amount of data can be stored permanently.

Types of file:

There are two types of files-
1) Binary files
2) Text Files
Let’s see these types in details:

1) Binary Files:

- The files which operates data in terms of binary stream (8 bit data) those files are called “Binary Files”
- Binary files used to operate binary data such as images, videos etc.
- Binary files are secured file because data in not directly interpreted by human being.

2) Text files:

- The files which operates data in terms of text streams (16 bit Unicode data) or characters those files are called “Text files”.
- Text files are Not secured file because its data directly interpreted by human being.

To deal with file in java, it has following classes:

1) Byte Streams class:

- Java byte streams are used to perform input and output in terms of 8-bit bytes.
- Though there are many classes related to byte streams but the most frequently used classes are, *FileInputStream* and *FileOutputStream*.
- Following program shows use of ByteStream classes.

Program 1) Program that reads content of file.

```
import    java.io.*;
public   class   FileRead
{
  public  static  void  main(String   args[])   throws    IOException
 {
       FileInputStream in = new  FileInputStream("d:\\myfile\\input.txt");
       int c;
     try
    {
      while ((c = in.read()) != -1)   //While not end of File
      {
           System.out.print((char)c);
      }
     }
     catch(FileNotFoundException   ex)
     {
         System.out.println(ex);
     }
     finally
    {
        in.close();
     }
   }
}
```

Program 2) Program that write content into file.

```
import      java.io.*;
public   class   FileWrite
{
  public  static  void  main(String args[]) throws IOException
 {
       FileOutputStream   out = new FileOutputStream("d:\\myfile\\write.txt");
       String   str="Box is Heavy";
       int  len=str.length();
       int  i;
     try
    {
      for(i=0;i<=len-1;i++)
      {
        out.write(str.charAt(i));
      }
     }
     catch(FileNotFoundException   ex)
     {
           System.out.println(ex);
     }
     finally
    {
      out.close();
     }
  }
}
```

Program 3) Program that copies content of file into another file.

```
import     java.io.*;
public  class   FileCopy
{
  public    static   void   main(String args[])   throws   IOException
 {
        FileInputStream      in = new   FileInputStream("d:\\myfile\\input.txt");
        FileOutputStream   out = new  FileOutputStream("d:\\myfile\\output.txt");
        int   c;
      try
     {
        while ((c = in.read()) != -1)       //while there is not end of File
       {
          out.write(c);
       }
      }
      catch(FileNotFoundException     ex)
      {
            System.out.println(ex);
      }
      finally
     {
          in.close();
          out.close();
      }
   }
}
```

2) Character Streams class:

- Java Byte streams are used to perform input and output of 8-bit bytes, whereas Java Character streams are used to perform input and output for 16-bit Unicode.
- There are many classes related to character streams but the most frequently used classes are, *FileReader and FileWriter*.
- Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but there is **major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time**.
- Following programs shows use of CharacterStream classes:

Program 1) Program that read the content of file.

```
import    java.io.*;
public class   NewReadFile
{
   public static void main(String args[]) throws IOException
   {
      FileReader  in = new   FileReader("d:\\myfile\\read.txt");
       int   c;
```

```
    try
    {
        while ((c = in.read()) != -1)    //while not end of file
        {
            System.out.print((char)c);
        }
    }
    catch(FileNotFoundException ex)
    {
         System.out.println(ex);
    }
    finally
    {
        in.close();
    }
   }
  }
```

Program 2) Program to write the content in file.

```
import    java.io.*;
public   class   NewWriteFile
{
  public  static  void  main(String   args[])  throws   IOException
  {
    FileWriter    out = new   FileWriter("d:\\myfile\\NewWrite.txt");
    String    str="Box is Heavy";
    int    len=str.length();
    int    i;
    try
    {
        for(i=0;i<=len-1;i++)
        {
                  out.write(str.charAt(i));
        }
    }
    catch(FileNotFoundException ex)
    {
         System.out.println(ex);
    }
    finally
    {
        out.close();
    }
   }
  }
```

Program 3) Program that copies content of file into another file.

```
import     java.io.*;
public   class     NewCopyFile
{
  public static void main(String args[])  throws  IOException
  {
    FileReader    in =   new  FileReader("d:\\myfile\\read.txt");
    FileWriter     out = new  FileWriter("d:\\myfile\\write.txt");
    int     c;
    try
    {
        while ((c = in.read()) != -1)  //while not end of file
        {
           out.write(c);
        }
    }
    catch(FileNotFoundException   ex)
    {
         System.out.println(ex);
    }
    finally
    {
        in.close();
        out.close();
    }
   }
 }
```

Java finally block:

- It is a block used to execute important code such as closing the opened file, closing database connection, database backup code etc.
- Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

Theory Assignment No: 02

1) What is Array? Explain all types of array in details.
2) How array can be initialized? Explain with example.
3) What is String? How string is initialized?
4) Explain following String class methods with example:
   1) length( ) 2) concat( ) 3) charAt( ) 4) equals( ) 5) compareTo( ) 6) startsWith( )
   7) endsWith( ) 8) indexOf( ) 9) replace( ) 10) substring( ) 11) split( )

5) Explain immutability of string.

6) Explain following methods of StringBuffer class with example:
   1) append( ) 2) insert( ) 3) delete( ) 4) reverse( ) 5) length( ) 6) replace( )
   7) substring( )

7) Give the difference between StringBuffer and StringBuilder class.

8) What is File? How file manipulation is done in Java?

Practical Assignment No: 4

1) Write a program that accept any 10 numbers and print in reverse order.
2) Write a program that finds sum of all elements of array.
3) Write a program that finds sum of even numbers and sum of odd numbers in an array.
4) Write a program which check entered number is present in an array or not.
5) Write a program which find maximum and minimum number in an array.
6) Write a program which print given number into word.(e.g. 45=FourFive)
7) Write a program which convert decimal number into binary equivalent number.
8) Write a program which convert decimal number into octal equivalent number.
9) Write a program which find sum first and last digit of the entered number.
10) Write a program which sorts array element in ascending order.
11) Write a program which sorts array element in descending order.
12) Write a program which swaps two integer arrays.
13) Write a program which finds the sum of all elements of the matrix.
14) Write a program which prints only diagonal elements of the matrix.
15) Write a program which find sum of diagonal elements of the matrix.
16) Write a program which find sum of each row and each column of the matrix.
17) Write a program to calculate addition of two matrices.
18) Write a program to calculate subtraction of two matrices.
19) Write a program to calculate multiplication of two matrices.
20) Write a program which check entered string is Palindrome or not.
21) Write a program which count total number of vowels present in string.
22) Write a program to read the content of file.
23) Write a program to write the data into file.
24) Write a program to copy the content of one file into another.