GANPAT UNIVERSITY

U. V. PATEL COLLEGE OF ENGINEERING

2CEIT302

OBJECT ORIENTED PROGRAMMING

UNIT 4

ARRAYS AND STRINGS

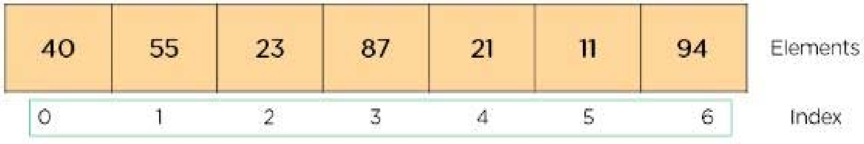
Prepared by: Prof. Y. J. Prajapati (Asst. Prof in IT Dept. , UVPCE)

Outline

- One-Dimensional Array
- Multi-Dimensional Array
- String Class and methods
- StringBuffer Class and methods

Array

- An array is a collection of similar types of data. It is a container that holds data (values) of one single type.
- **Java array** is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.
- It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.



40	55	23	87	21	11	94	Elements
0	1	2	3	4	5	6	Index

Features of an array

- Dynamic allocation: In arrays, the memory is created dynamically, which reduces the amount of storage required for the code.
- Elements stored under a single name: All the elements are stored under one name. This name is used any time we use an array.
- Occupies contiguous location: The elements in the arrays are stored at adjacent positions. This makes it easy for the user to find the locations of its elements.

Array

- ## Advantages of Arrays
- Java arrays enable you to access any element randomly with the help of indexes
- It is easy to store and manipulate large data sets
- ## Disadvantages of Arrays in Java
- The size of the array cannot be increased or decreased once it is declared—arrays have a fixed size
- Java cannot store heterogeneous data. It can only store a single type of primitives

How to declare an array?

dataType[] arrayName;

dataType[] arrayName; (or)

dataType []arrayName; (or)

dataType arrayName[];

- dataType - it can be primitive data types like int, char, double, byte, etc. or Java objects
- arrayName - it is an identifier

Example :

```
double[] data;
```

Here, data is an array that can hold values of type double.

But, how many elements can array this hold?

The memory will define the number of elements that the array can hold.

```
data = new Double[10];
```

Here, the size of the array is 10. This means it can hold 10 elements
(10 double types values). The size of an array is also known as the length of an array.

Array

- **Let's take another example:**

  int[] age;

  age = new int[5];

  Here, age is an array. It can hold 5 values of int type.

- In Java, we can declare and allocate memory of an array in one single statement.

  For example,

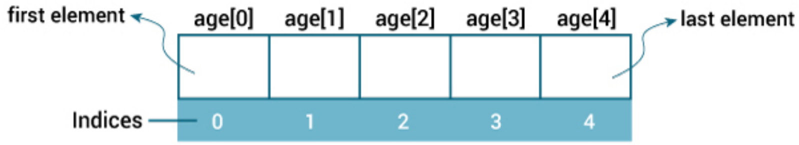
  int[] age = new int[5];

- **Java Array Index**

- In Java, each element in an array are associated with a number. The number is known as an array index. We can access elements of an array by using those indices.

Example:

int[] age = new int[5];



Array age **of length 5**

Array

- Example

```
class ArrayExample
{
          public static void main(String[] args)
         {
         // create an array of length 5
         int[] age = new int[5];
         // access each element of the array using the index number
         System.out.println(age[0]);
         System.out.println(age[1]);
         System.out.println(age[2]);
         System.out.println(age[3]);
         System.out.println(age[4]);
         }
}
```

Output:

```
0
0
0
0
0
```

- we have created an array named age. However, we did not assign any values to the array. Hence when we access the individual elements of the array, the default values are printed to the screen.
- Here, we are individually accessing the elements of the array. There is a better way to access elements of the array using a loop (generally for-loop).

```
class ArrayExample
{
        public static void main(String[] args)
        {
                // create an array of length 5
                int[] age = new int[5];
                // accessing elements using for loop
                for (int i = 0; i < 5; ++i)
                {
                            System.out.println(age[i]);
                }
        }
 }
```

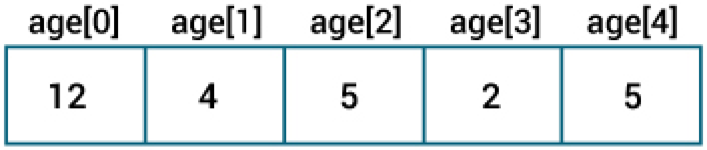
Array

- **How to initialize arrays in Java?**

Here's how you can initialize an array during declaration.

int[] age = {12, 4, 5, 2, 5};

- This statement creates an array named age and initializes it with the value provided in the curly brackets.
- The length of the array is determined by the number of values provided inside the curly braces separated by commas. In our example, the length of age is 5



age[0]	age[1]	age[2]	age[3]	age[4]
12	4	5	2	5

- Example

```
class ArrayExample
{
        public static void main(String[] args)
        {
        // create an array
        int[] age = {12, 4, 5, 2, 5};
        // access elements of the array
        for (int i = 0; i < 5; ++i)
        {
        System.out.println("Element at index " + i +": " + age[i]);
        }
        }
}
```

Output:

```
Element at index 0: 12
Element at index 1: 4
Element at index 2: 5
 Element at index 3: 2
 Element at index 4: 5
```

Arrays

- **How to access array elements?**

```
class ArrayExample
{
    public static void main(String[] args)
    {
            int[] age = new int[5];
            // insert 14 to third element
            age[2] = 14;
            // insert 34 to first element
            age[0] = 34;
            for (int i = 0; i < 5; ++i)
            {
            System.out.println("Element at index " + i +": " + age[i]);
            }
    }
}
```

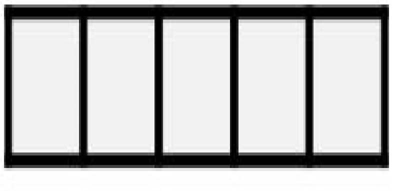
Output:

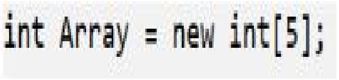
```
Element at index 0: 34
Element at index 1: 0
Element at index 2: 14
Element at index 3: 0
Element at index 4: 0
```

Types of Arrays

□ One-dimensional Array

- □ Also known as a linear array, the elements are stored in a single row.





```
int Array = new int[5];
```

Examples:

One dimensional array declaration of variable:

int[] a; // valid declaration

int b[]; // valid declaration

int[] c; // valid declaration

int a[5]; // invalid declaration -- If we want to assign size of array at the declaration time, it gives compile time error.

int a[], b[]; // valid declaration, both arrays are // one dimensional array.

int c[], [] d; // invalid declaration

int[] e, [] f; // invalid declaration

Examples:

// invalid, here size of array is not given

int[] a = new int[];

// valid, here creating 'b' array of size 5

int[] b = new int[5];

// valid

int[] c = new int[0];

// gives runtime error

int[] d = new int[-1];

One Dimensional Array-Example

- **One Dimensional Array Using Standard Method**

```
class Onedimensional
{
        public static void main(String args[])
        {
        int[] a=new int[3];//declaration
        a[0]=10;//initialization
        a[1]=20;
        a[2]=30;
        //printing array
        System.out.println("One dimensional array elements are");
        System.out.println(a[0]);
        System.out.println(a[1]);
        System.out.println(a[2]);
        }
}
```

OUTPUT :

```
One dimensional array elements are
10
20
30
```

- **Using Scanner**

```
import java.util.*;
class OnedimensionalScanner
{
  public static void main(String args[])
  {
int len;
Scanner sc=new Scanner(System.in);
System.out.println("Enter Array length : ");
len=sc.nextInt();
int a[]=new int[len];//declaration
System.out.print("Enter " + len + " Element to Store in Array :\n");
     for(int i=0; i<len; i++)
     {
       a[i] = sc.nextInt();
     }
     System.out.print("Elements in Array are :\n");
     for(int i=0; i<len; i++)
     {
       System.out.print(a[i] + "  ");
     }
  }
}
```

```
Enter Array length :
4
Enter 4 Element to Store in Array :
1
2
3
4
Elements in Array are :
1  2  3  4
```

One Dimensional Array-Example

- **Using For Loop – One Dimensional Array**

```
class OnedimensionalLoop
{
public static void main(String args[])
{
   int a[]={10,20,30,40,50};//declaration and initialization
   System.out.println("One dimensional array elements are :\n");
   for(int i=0;i<a.length;i++)
   {
System.out.println("a["+i+"]:"+a[i]);
   }
}
}
```

```
One dimensional array elements are
:
a[0]:10
a[1]:20
a[2]:30
a[3]:40
a[4]:50
```

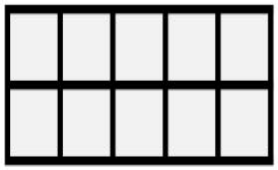
- **Using String**

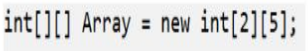
```
class OneDimensionString
{
    public static void main(String[] args)
    {
        //declare and initialize one dimension array
        String[] str = new String[]{"one", "two", "three", "four"};
        System.out.println("These are elements of one Dimensional array.");
        for (int i = 0; i < str.length; i++)
        {
            System.err.println(str[i] + "   ");
        }
    }
}
```

```
These are elements of one Dimensional array.
one
two
three
four
```

Multidimensional Array in Java

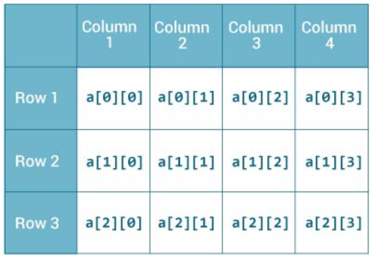
- data is stored in row and column based index (also known as matrix form).
- we will learn about the Java multidimensional array using 2-dimensional arrays and 3-dimensional arrays
- A multidimensional array is an array of arrays. Each element of a multidimensional array is an array itself. For example,





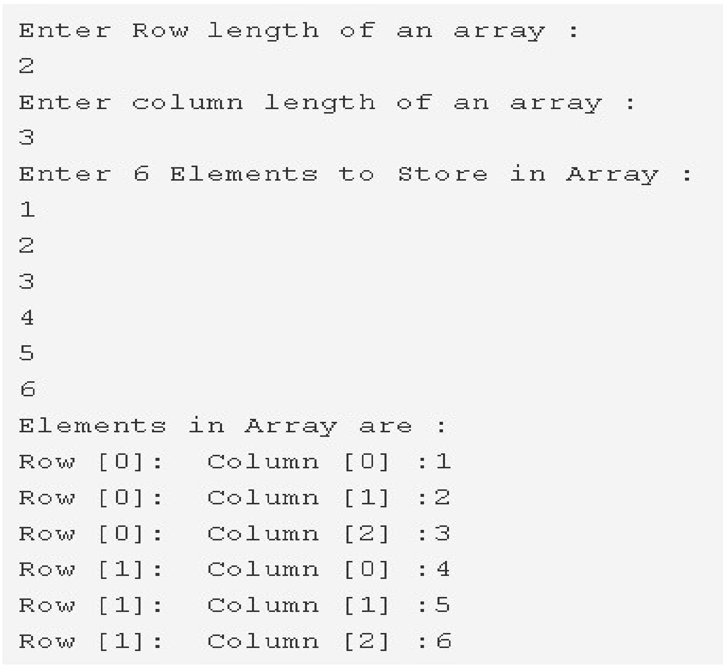
```
int[][] Array = new int[2][5];
```

int[][] a = new int[3][4];



	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

- Basically, you need to define both the rows and columns and then go ahead with declaring the elements in the respective locations or indexes.



```
Enter Row length of an array :
2
Enter column length of an array :
3
Enter 6 Elements to Store in Array :
1
2
3
4
5
6
Elements in Array are :
Row [0]:  Column [0] :1
Row [0]:  Column [1] :2
Row [0]:  Column [2] :3
Row [1]:  Column [0] :4
Row [1]:  Column [1] :5
Row [1]:  Column [2] :6
```

Declarations of Multidimensional array

- Suppose, you want to create two dimensional array of int type data. So you can declare two dimensional array in many of the following ways:

  int a[][]; // valid

  int[][] b; // valid

  int[][] c; // valid

  int[] d[]; // valid

  int[][] e; // valid

  int[] f[]; // valid

  [][] int g; // invalid

  [] int[] h; // invalid

- Now, Suppose we want to write multiple declarations of array variable then you can use it like this.

  int[] a[], b[];

  // Here, 'a' is two dimensional array, 'b'

  // is two dimensional array

  int[] c[], d[];

  // Here, 'c' is two dimensional array, 'd'

  // is two dimensional array

  int[][] e, f[];

  // Here, 'e' is two dimensional array, 'f'

  // is three dimensional array

  int[] g[], h;

  // Here, 'g' is two dimensional array,

  // 'h' is one dimensional array

Example

```
class TwodimensionalStandard
{
public static void main(String args[])
{
int[][] a={{10,20},{30,40}};//declaration and
      initialization
System.out.println("Two dimensional array elements are");
System.out.println(a[0][0]);
System.out.println(a[0][1]);
System.out.println(a[1][0]);
System.out.println(a[1][1]);
}
}
```

```
Two dimensional array elements are
10
20
30
40
```

Using For Loop

```
class TwodimensionalLoop
{
public static void main(String args[])
{
int[][] a={{10,20},{30,40},{50,60}};//declaration and
      initialization
System.out.println("Two dimensional array elements are");
for (int i = 0; i < 3; i++)
{
         for (int j = 0; j < 2; j++)
   {
            System.out.println(a[i][j]);
         }
}
}
}
```

```
Two dimensional array elements
are
10
20
30
40
50
60
```

Examples

Using Scanner

```
import java.util.*;
class TwoDimensionalScanner
{
   public static void main(String args[])
   {
Scanner sc=new Scanner(System.in);
System.out.println("Enter Row length of an array : ");
int row=sc.nextInt();
System.out.println("Enter column length of an array : ");
int column=sc.nextInt();
int a[][]=new int[row][column];//declaration
System.out.print("Enter " + row*column + " Elements to Store in Array :\n");
        for (int i = 0; i < row; i++)
{
    for(int j = 0; j < column; j++)
    {
           a[i][j] = sc.nextInt();
    }
}
```

```
  System.out.print("Elements in Array are :\n");
       for (int i = 0; i < row; i++)
{
    for(int j = 0; j < column; j++)
    {
       System.out.println("Row ["+i+"]:  Column ["+j+"] :"+a[i][j]);
    }
}
   }
}
```

```
Enter Row length of an array :
2
Enter column length of an array :
3
Enter 6 Elements to Store in Array :
1
2
3
4
5
6
Elements in Array are :
Row [0]:  Column [0] :1
Row [0]:  Column [1] :2
Row [0]:  Column [2] :3
Row [1]:  Column [0] :4
Row [1]:  Column [1] :5
Row [1]:  Column [2] :6
```

Jagged Array

- If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.

```
class JaggedArray{
    public static void main(String[] args){
        //declaring a 2D array with odd columns
        int arr[][] = new int[3][];
        arr[0] = new int[3];
        arr[1] = new int[4];
        arr[2] = new int[2];
        //initializing a jagged array
        int count = 0;
        for (int i=0; i<arr.length; i++)
            for(int j=0; j<arr[i].length; j++)
                arr[i][j] = count++;
```

```
//printing the data of a jagged array
    for (int i=0; i<arr.length; i++)
   {
        for (int j=0; j<arr[i].length; j++)
   {
            System.out.print(arr[i][j]+" ");
        }
        System.out.println();//new line
    }
  }

}
```

Output:

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

Strings

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

How to create a string object?

- There are two ways to create String object:
- By string literal
- By new keyword

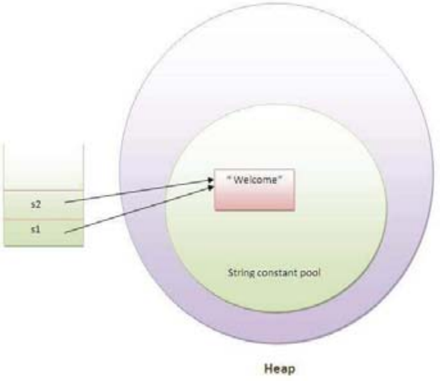
- **String Literal**
- Java String literal is created by using double quotes. For Example:

  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.

Strings

- **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).

```
 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
```

Java String Methods

- Java String provides various methods that allow us to perform different string operations. Here are some of the commonly used string methods.
- concat() : joins the two strings together
- equals() : compares the value of two strings
- charAt() : returns the character present in the specified location
- getBytes() : converts the string to an array of bytes
- indexOf() : returns the position of the specified character in the string
- length() : returns the size of the specified string
- replace() : replaces the specified old character with the specified new character
- substring() : returns the substring of the string
- split() : breaks the string into an array of strings
- toLowerCase() : converts the string to lowercase
- toUpperCase() : converts the string to uppercase

Examples

- **Java find string's length**

```
class Main
{
public static void main(String[] args)
{
    // create a string
    String X = "Hello! World";
    System.out.println("The string is: " + X);
    //checks the string length
    System.out.println("The length of the string: " + X.length());
}
}
```

Output

```
The string is: Hello! World
The length of the string: 12
```

- **Java join two strings using concat()**

```
class Main
{
    public static void main(String[] args)
    {
            // create string
            String greet = "Hello! ";
            System.out.println("First String: " + greet);
            String name = "World";
            System.out.println("Second String: " + name);
            // join two strings
            String joinedString = greet.concat(name);
            System.out.println("Joined String: " + joinedString);
     }
 }
```

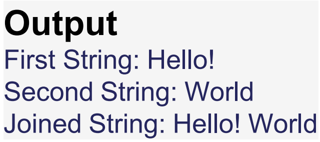
Output

```
First String: Hello!
Second String: World
Joined String: Hello! World
```

Examples

- **Java join strings using + operator**

```
class Main
{
      public static void main(String[] args)
      { // create string
                String greet = "Hello! ";
                System.out.println("First String: " + greet);
                String name = "World";
                System.out.println("Second String: " + name);
                // join two strings
                String joinedString = greet + name;
                System.out.println("Joined String: " + joinedString);
       }
 }
```

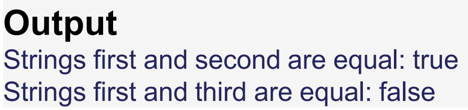


Output

```
First String: Hello!
Second String: World
Joined String: Hello! World
```

- **Java compare two strings**

```
class Main
{
       public static void main(String[] args)
      { // create strings
                String first = "java programming";
                String second = "java programming";
                String third = "python programming";
                // compare first and second strings
                boolean result1 = first.equals(second);
                System.out.println("Strings first and second are equal: " + result1);
                //compare first and third strings
                boolean result2 = first.equals(third);
                System.out.println("Strings first and third are equal: " + result2);
      }
}
```



Output

```
Strings first and second are equal: true
Strings first and third are equal: false
```

Examples

- **Java get characters from a string**

```
class Main
{
        public static void main(String[] args)
        {
        // create string using the string literal
        String greet = "Hello! World";
        System.out.println("The string is: " + greet);
        // returns the character at 3
         System.out.println("The character at 3: " + greet.charAt(3));
        // returns the character at 7
        System.out.println("The character at 7: " + greet.charAt(7));
        }
 }
```

Output

```
The string is: Hello! World
The character at 3: l
The character at 7: W
```

- **Java Strings other methods**

```
class Main
{
public static void main(String[] args)
 {
// create string using the new keyword
String example = new String("Hello! World");
// returns the substring World
System.out.println("Using the subString(): " + example.substring(7));
// converts the string to lowercase
System.out.println("Using the toLowerCase(): " + example.toLowerCase());
// converts the string to uppercase
System.out.println("Using the toUpperCase(): " + example.toUpperCase());
// replaces the character '!' with 'o'
System.out.println("Using the replace(): " + example.replace('!', 'o'));
}
}
```

Output

```
Using the subString(): World
Using the toLowerCase(): hello! World
Using the toUpperCase(): HELLO! WORLD
Using the replace(): Helloo World
```

StringBuffer class

- StringBuffer class is used to create a **mutable** string object. It means, it can be changed after it is created. It represents growable and writable character sequence.
- It contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.
- **Constructors of StringBuffer class**
  - **StringBuffer()**

  This constructs a string buffer with no characters in it and an initial capacity of 16 characters.

- **StringBuffer(CharSequence seq)**

  This constructs a string buffer that contains the same characters as the specified CharSequence.

- **StringBuffer(int capacity)**

  This constructs a string buffer with no characters in it and the specified initial capacity.

- **StringBuffer(String str)**

  This constructs a string buffer initialized to the contents of the specified string.

StringBuffer class methods

- **append(String s)**

  is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.

- **insert(int offset, String s)**

  is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

- **replace(int startIndex, int endIndex, String str)**

  is used to replace the string from specified startIndex and endIndex.

- **delete(int startIndex, int endIndex)**

  is used to delete the string from specified startIndex and endIndex.

- **reverse()**

  is used to reverse the string.

- **capacity()**

  is used to return the current capacity.

StringBuffer class methods

- **ensureCapacity(int minimumCapacity)**

  is used to ensure the capacity at least equal to the given minimum.

- **charAt(int index)**

  is used to return the character at the specified position.

- **length()**

  is used to return the length of the string i.e. total number of characters.

- **substring(int beginIndex)**

  is used to return the substring from the specified beginIndex.

- **substring(int beginIndex, int endIndex)**

  is used to return the substring from the specified beginIndex and endIndex.

Examples

- **StringBuffer append() method**

  The append() method concatenates the given argument with this string.

```
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
}
}
```

Examples

- **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
}
}
```

Examples

- **StringBuffer reverse() method**

  The reverse() method of StringBuilder 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
}
}
```

Examples

- **StringBuffer capacity() method**

- 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.

```
class StringBufferExample6
{
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 (old
      capacity*2)+2
}
}
```

THANK YOU!
ANY QUESTIONS?