MODULE-2

Arrays and Strings: Declaration, Initialization and accessing values, One-Dimensional Arrays, Multi-dimensional arrays, Alternative Array Declaration Syntax, var-arg methods, Wrapper Classes. String, String Buffer and StringBuilder classes.

1.Define an array. Explain how to declare, create and initialize an array with example program.

Arrays :

An array is a collection of similar type of elements which has contiguous memory location. 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.

Declaring Array Variables

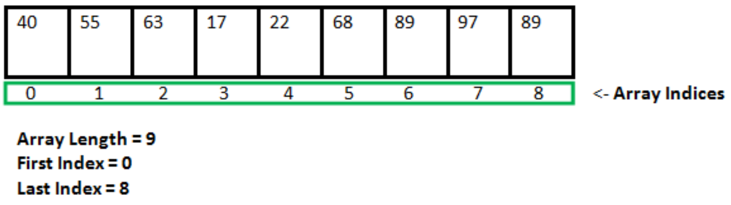
To use an array in a program, declare a variable to reference the array, and must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

Syntax

dataType[] arrayRefVar;
or
dataType arrayRefVar[];

Example: The following code snippets are examples of this syntax:

double[] myList;
or
double myList[];



Creating Arrays: Create an array by using the new operator with the following syntax −

Syntax

arrayRefVar = new dataType[arraySize];

The above statement does two things −

- It creates an array using new dataType[arraySize].
- It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below −

dataType[] arrayRefVar = new dataType[arraySize];

Alternatively can create arrays as follows:

dataType[] arrayRefVar = {value0, value1, ..., valuek};

The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayRefVar.length-1.

Example: Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList −

double[] myList = new double[10];

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.

Processing Arrays: When processing array elements, often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

Example

```
public class TestArray {
  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};
```

```
      // Print all the array elements
      for (int i = 0; i < myList.length; i++) {
        System.out.println(myList[i] + " ");
      }
      // Summing all elements
      double total = 0;
      for (int i = 0; i < myList.length; i++) {
        total += myList[i];
      }
      System.out.println("Total is " + total);
       // Finding the largest element
      double max = myList[0];
      for (int i = 1; i < myList.length; i++) {
        if (myList[i] > max)
        max = myList[i];
      }
      System.out.println("Max is " + max);
    }
  }
```

This will produce the following result −

Output

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

Advantages

- Code Optimization: It makes the code optimized; we can retrieve or sort the data efficiently.
- Random access: We can get any data located at an index position.

Disadvantages

Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Multi dimensional arrays (2D, 3D, … arrays)

Multi dimensional arrays represent 2D, 3D, …arrays which are the combinations of several earlier types of arrays. For example, a two dimensional array is a combination of 2 or more one dimensional arrays. Similarly a three dimensional array is a combination of 2 or more two dimensional arrays.

Two dimensional arrays (2D arrays):

A two dimensional array represents several rows and columns of data. For example, the marks obtained by a group of students in five different subjects can be represented by a 2D array.

Creating a two dimensional array: There are some ways of creating a two dimensional array as:

➢ We can declare a dimensional array and directly store elements at the time of its declaration, as :

```
int marks[][]={{50, 60, 70, 80, 90},
               {85, 65, 45, 96, 32},
               {56, 89, 95, 100, 94}};
```

So, the JVM creates 3*5=15 blocks of memory as there are 15 elements being stored into the array.

➢ Another way of creating a two dimensional array is by declaring the array first and then allotting memory for it by using new operator.

```
int marks[][];
```

```
        marks=new int[3][5];
```

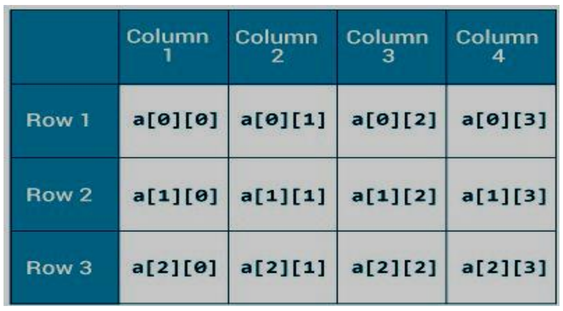
While writing a two dimensional array, we can write the two pairs of square braces before or after the array name, as:

```
int marks[][]=new int[3][5];
int[][] marks=new int[3][5];
```

Ex:

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

Created a multidimensional array named a. It is a 2-dimensional array, that can hold a maximum of 12 elements,



	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]

Three dimensional arrays (3D array):

We can consider a three dimensional array as a combination of several two dimensional arrays. This concept is useful when we want to handle group of elements belonging to another group. For example, a college has 3 departments: electronics, computers and civil. We want to represent the marks obtained by the students of each department in 3 different subjects. To store all these marks, department wise, we need a three dimensional array as:

```
        int marks[][][]={{{50, 60, 70}, {85, 95, 75}},
                        {{70, 71, 72}, {85, 65, 75}},
                        {{85, 65, 75}, {84, 74, 94}}};
```

Other ways of creating three dimensional arrays:

```
                    int marks[][][]=new int[3][2][3];
                    int[][][] marks=new int[3][2][3];
```

***(Optional Question)**

Alternative Array Declaration Syntax

One Dimensional Array:

It is a collection of variables of same type which is used by a common name.
Examples:
One dimensional array declaration of variable:

```
import java.io.*;

class Abc {
   public static void main(String[] args)
   {
      int[] a; // valid declaration
      int b[]; // valid declaration
      int[] c; // valid declaration
   }
}
```

We can write it in any way.
Now, if you declare your array like below:

```
import java.io.*;

class Abc {
   public static void main(String[] args) {
      // invalid declaration -- If we want to assign
```

```
        // size of array at the declaration time,  it
        // gives compile time error.
        int a[5];

        // valid declaration
        int b[];
    }}
```

Now, suppose we want to write multiple declaration of array variable then we can use it like this.

```
import java.io.*;

class Abc {
    public static void main(String[] args) {
        // valid declaration, both arrays are
        // one dimensional array.
        int a[], b[];

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

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

When we are declaring multiple variable of same type at a time, we have to write variable first then declare that variable except first variable declaration. There is no restriction for the first variable.
Now, when we are creating array it is mandatory to pass the size of array; otherwise we will get compile time error.
You can use new operator for creating an array.

```
import java.io.*;

class Abc {
    public static void main(String[] args) {
        // 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];
    }
}
```

Printing array:/* A complete Java program to demonstrate working of one dimensional arrays */

```
class oneDimensionalArray {
    public static void main(String args[])    {
        int[] a; // one dimensional array declaration
        a = new int[3]; // creating array of size 3
        for (int i = 0; i < 3; i++) {
            a[i] = 100;
            System.out.println(a[i]);
        }
    }}
   Output:
                          100
                          100
                          100
```

Two Dimensional Arrays:

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:

```
// Java program to demonstrate different ways
// to create two dimensional array.
import java.io.*;

class Abc {
    public static void main(String[] args) {
        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.

```
// Java program to demonstrate multiple declarations
// of array variable
import java.io.*;

class Abc {
public static void main(String[] args) {

        // Here, 'a' is two dimensional array, 'b'
        // is two dimensional array
        int[] a[], b[];

          // Here, 'c' is two dimensional array, 'd'
        // is two dimensional array
        int[] c[], d[];

          // Here, 'e' is two dimensional array, 'f'
        // is three dimensional array
        int[][] e, f[];

          // Here, 'g' is two dimensional array,
        // 'h' is one dimensional array
        int[] g[], h;
    }
}
```

Creating one dimensional array and two dimensional arrays without new operator:/* Java program for one and two dimensional arrays.
   without new operator*/
class oneTwoDimensionalArray {

```
    public static void main(String args[])
    {
        int[] a[] = { { 1, 1, 1 }, { 2, 2, 2 },
                      { 3, 3, 3 } }, b = { 20 };

        // print 1D array
        System.out.println(b[0]);

        // print 2D array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                a[i][j] = 100;
                System.out.println(a[i][j]);
            }
        }
    }
}
```

Output:

```
20
100
100
100
100
100
100
100
100
100
```

Creating one dimensional array and two dimensional array using new operator:

```
/* Java program for one and two dimensional arrays.
   using new operator*/
class oneTwoDimensionalArray {

    public static void main(String args[])
    {
        int[] a[], b = { 20 };
        a = new int[3][3];
        b = new int[3];

        // print 1D array
        for (int i = 0; i < 3; i++)
            System.out.println(b[i]);

        // print 2D array
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                a[i][j] = 100;
                System.out.println(a[i][j]);
            }
        }
    }
}
```

Output:

```
0
0
0
100
100
100
100
100
100
100
100
100
```

2. What is var-arg methods and explain with example?

Var_arg Methods

Variable Arguments (Varargs) in Java is a method that takes a variable number of arguments. Variable Arguments in Java simplifies the creation of methods that need to take a variable number of arguments.

Need of Java Varargs

- Until JDK 4, we can't declare a method with variable no. of arguments. If there is any change in the number of arguments, we have to declare a new method. This approach increases the length of the code and reduces readability.
- Before JDK 5, variable-length arguments could be handled in two ways. One uses an overloaded method(one for each), and another puts the arguments into an array and then passes this array to the method. Both of them are potentially error-prone and require more code.
- To resolve these problems, Variable Arguments (Varargs) were introduced in JDK 5. From JDK 5 onwards, we can declare a method with a variable number of arguments. Such types of methods are called Varargs methods. The varargs feature offers a simpler, better option.

Varargs is a short name for variable arguments. In Java, an argument of a method can accept arbitrary number of values. This argument that can accept variable number of values is called varargs.

The **syntax** for implementing varargs is as follows:

```
accessModifier methodName(datatype… arg)
{
   // method body
}
```

In order to define vararg, ... (three dots) is used in the formal parameter of a method. A method that takes variable number of arguments is called a variable-arity method, or simply a varargs method.

First, the example without using varargs:

```
class NoVararg {
   public int sumNumber(int a, int b){
      return a+b;
   }
   public int sumNumber(int a, int b, int c){
      return a+b+c;
   }
   public static void main( String[] args ) {
      NoVararg obj = new NoVararg();
      System.out.println(obj.sumNumber(1, 2));
      System.out.println(obj.sumNumber(1, 2, 3));
   }
}
```

Output:

```
3
6
```

2.Example: Working of varargs

```
class VarargExample {
   public int sumNumber(int ... args){
 System.out.println("argument length: " + args.length);
      int sum = 0;
      for(int x: args){
         sum += x;
      }
      return sum;
   }
   public static void main( String[] args ) {
      VarargExample ex = new VarargExample();
      int sum2 = ex.sumNumber(2, 4);
      System.out.println("sum2 = " + sum2);
```

```
            int sum3 = ex.sumNumber(1, 3, 5);
            System.out.println("sum3 = " + sum3);
            int sum4 = ex.sumNumber(1, 3, 5, 7);
            System.out.println("sum4 = " + sum4);
        }}
```

Output:

```
argument length: 2
sum2 = 6
argument length: 3
sum3 = 9
argument length: 4
sum4 = 16
```

Here, the sumNumber() method returns the sum of int parameters passed to it (doesn't matter the number of arguments passed).

3. What is the need of Wrapper Class in java? Write a program for implementation of the wrapper class in Java?

Wrapper Classes :

The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive. Since J2SE 5.0, autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically.

The automatic conversion of primitive into an object is known as **Autoboxing** and **vice-versa Unboxing.**

Use of Wrapper classes in Java: Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc.

- ***Change the value in Method:*** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- ***Serialization:*** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- ***Synchronization:*** Java synchronization works with objects in Multithreading.
- ***java.util package:*** The java.util package provides the utility classes to deal with objects.
- ***Collection Framework:*** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

The eight classes of the java.lang package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Primitive Type	Wrapper class	Primitive Type	Wrapper class
boolean	**Boolean**	int	**Integer**
char	**Character**	long	**Long**
byte	**Byte**	float	**Float**
short	**Short**	double	**Double**

Autoboxing: The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short. Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects.

Wrapper class Example: Primitive to Wrapper

```
//Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
 System.out.println(a+" "+i+" "+j);
}}
```

Output: 20 20 20

Unboxing: The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

Wrapper class Example: Wrapper to Primitive

```
//Java program to convert object into primitives
//Unboxing example of Integer to int
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally
   System.out.println(a+" "+i+" "+j);
}}
```

Output: 3 3 3

Java Wrapper classes Example

```
//Java Program to convert all primitives into its corresponding
//wrapper objects and vice-versa
public class WrapperExample3{
public static void main(String args[]){
byte b=10;
short s=20;
int i=30;
long l=40;
float f=50.0F;
double d=60.0D;
char c='a';
boolean b2=true;
  //Autoboxing: Converting primitives into objects
Byte byteobj=b;
Short shortobj=s;
Integer intobj=i;
Long longobj=l;
Float floatobj=f;
Double doubleobj=d;
Character charobj=c;
Boolean boolobj=b2;
  //Printing objects
System.out.println("---Printing object values---");
System.out.println("Byte object: "+byteobj);
System.out.println("Short object: "+shortobj);
System.out.println("Integer object: "+intobj);
System.out.println("Long object: "+longobj);
System.out.println("Float object: "+floatobj);
System.out.println("Double object: "+doubleobj);
System.out.println("Character object: "+charobj);
System.out.println("Boolean object: "+boolobj);
  //Unboxing: Converting Objects to Primitives
byte bytevalue=byteobj;
short shortvalue=shortobj;
int intvalue=intobj;
long longvalue=longobj;
float floatvalue=floatobj;
double doublevalue=doubleobj;
char charvalue=charobj;
boolean boolvalue=boolobj;
  //Printing primitives
System.out.println("---Printing primitive values---");
System.out.println("byte value: "+bytevalue);
System.out.println("short value: "+shortvalue);
System.out.println("int value: "+intvalue);
System.out.println("long value: "+longvalue);
System.out.println("float value: "+floatvalue);
System.out.println("double value: "+doublevalue);
System.out.println("char value: "+charvalue);
System.out.println("boolean value: "+boolvalue);   }}
```

Output:

```
                ---Printing object values---
        Byte object: 10
        Short object: 20
        Integer object: 30
        Long object: 40
        Float object: 50.0
        Double object: 60.0
        Character object: a
        Boolean object: true
                ---Printing primitive values---
        byte value: 10
        short value: 20
        int value: 30
        long value: 40
        float value: 50.0
        double value: 60.0
        char value: a
        boolean value: true
```

4.a) Define a string class.

b) Describe string methods with an example for each one.

a) String class :

The string is a sequence of characters. In Java, objects of String are immutable which means a constant and cannot be changed once created.

Creating a String

There are two ways to create string in Java:

1. String literal

String s=”Java”;

2. Using new keyword

String s=new String(“Java”);

b) Java String Class Methods :

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

No.	Method	Description
1	char charAt(int index)	It returns char value for the particular index
2	int length()	It returns string length
3	static String format(String format, Object... args)	It returns a formatted string.
4	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
5	String concat(String str)	It concatenates the specified string.
6	String replace(char old, char new)	It replaces all occurrences of the specified char value.
7	static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
8	String[] split(String regex)	It returns a split string matching regex.
9	int indexOf(int ch)	It returns the specified char value index.
10	int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
11	int indexOf(String substring)	It returns the specified substring index.
12	int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
13	String toLowerCase()	It returns a string in lowercase.
14	String toUpperCase()	It returns a string in uppercase.
15	String trim()	It removes beginning and ending spaces of this string.

Java String Compare

It is used in authentication (by equals() method), sorting (by compareTo() method), reference matching (by == operator) etc.

There are three ways to compare String in Java:

1. **By Using equals() Method**
2. **By Using == Operator**
3. **By compareTo() Method**

1) By Using equals() Method: The String class equals() method compares the original content of the string. It compares values of string for equality. String class provides the following two methods:

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

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

Teststringcomparison1.java

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

Output:

```
true
true
false
```

In the above code, two strings are compared using equals() method of String class. And the result is printed as boolean values, true or false.

Teststringcomparison2.java

```
class Teststringcomparison2{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="SACHIN";
   System.out.println(s1.equals(s2));//false
   System.out.println(s1.equalsIgnoreCase(s2));//true
 }
}
```

Output:

```
false
true
```

In the above program, the methods of String class are used. The equals() method returns true if String objects are matching and both strings are of same case. equalsIgnoreCase() returns true regardless of cases of strings.

2) By Using == operator

The == operator compares references not values.

Teststringcomparison3.java

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

Output:

```
true
false
```

3) String compare by compareTo() method

The above code, demonstrates the use of == operator used for comparing two String objects.

3) By Using compareTo() method

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

Suppose s1 and s2 are two String objects. If:

s1 == s2 : The method returns 0.

s1 > s2 : The method returns a positive value.

s1 < s2 : The method returns a negative value.

Teststringcomparison4.java

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

Output:

```
 0
 1
-1
```

5. Describe StringBuffer methods and StringBuilder methods

Java StringBuffer Class

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

StringBuffer is a peer class of **String** that provides much of the functionality of strings. The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences. **StringBuffer** may have characters and substrings inserted in the middle or appended to the end. It will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

- java.lang.StringBuffer extends (or inherits from) Object class.
- All Implemented Interfaces of StringBuffer class: Serializable, Appendable, CharSequence.
- public final class StringBuffer extends Object implements Serializable, CharSequence, Appendable.
- String buffers are safe for use by multiple threads. The methods can be synchronized wherever necessary so that all the operations on any particular instance behave as if they occur in some serial order.
- Whenever an operation occurs involving a source sequence (such as appending or inserting from a source sequence) this class synchronizes only on the string buffer performing the operation, not on the source.
- It inherits some of the methods from the Object class which such as clone(), equals(), finalize(), getClass(), hashCode(), notifies(), notifyAll().

Constructor	Description
StringBuffer()	It creates an empty String buffer with the initial capacity of 16.
StringBuffer(String str)	It creates a String buffer with the specified string..
StringBuffer(int capacity)	It creates an empty String buffer with the specified capacity as length.

Important methods of StringBuffer class

Modifier and Type	Method	Description
public synchronized StringBuffer	append(String s)	It 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.
public synchronized StringBuffer	insert(int offset, String s)	It 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.
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	reverse()	is used to reverse the string.
public int	capacity()	It is used to return the current capacity.
public void	ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.

public char	charAt(int index)	It is used to return the character at the specified position.
public int	length()	It is used to return the length of the string i.e. total number of characters.
public String	substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
public String	substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

Mutable String: A String that can be modified or changed is known as mutable String. StringBuffer and StringBuilder classes are used for creating mutable strings.

1) StringBuffer Class append() Method: The append() method concatenates the given argument with this String.

Ex:

```
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
}
}
```

Output: Hello Java

2) 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
}
}
```

Output: HJavaello

3) 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
}
}
```

Output: HJavalo

4) StringBuffer delete() Method: The delete() method of the 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
} }
```

Output: Hlo

5) StringBuffer reverse() Method: The reverse() method of the 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
}
}
```

```
Output: olleH
```

6) StringBuffer capacity() Method:

The capacity() method of the 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 (oldcapacity*2)+2
} }
```

```
Output:
        16
        16
        34
```

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

```
class StringBufferExample7{
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
sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34
sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70
}
}
```

```
Output:   16
          16
          34
          34
          70
```

Mutable Objects : The mutable objects are objects whose value can be changed after initialization. We can change the object's values, such as field and states, after the object is created.
Ex: Java.util.Date, StringBuilder, StringBuffer, etc.

Immutable Objects: The immutable objects are objects whose value can not be changed after initialization. We can not change anything once the object is created. For example, **primitive objects** such as int, long, float, double, **all legacy classes, Wrapper class, String class**, etc.

Java StringBuilder Class

The **StringBuilder** in Java represents a mutable sequence of characters. Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternative to String Class, as it creates a mutable sequence of characters. The function of StringBuilder is very much similar to the StringBuffer class, as both of them provide an alternative to String Class by making a mutable sequence of characters. However the StringBuilder class differs from the StringBuffer class on the basis of synchronization.

The StringBuilder class provides no guarantee of synchronization whereas the StringBuffer class does. Therefore this class is designed for use as a drop-in replacement for StringBuffer in places where the StringBuffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to StringBuffer as it will be faster under most implementations. Instances of StringBuilder are not safe for use by multiple threads. If such synchronization is required then it is recommended that StringBuffer be used. It is available since JDK 1.5.

Class Hierarchy:

java.lang.Object
↳ java.lang
↳ Class StringBuilder

Syntax:

public final class StringBuilder extends Object implements Serializable, CharSequence

Important Constructors of StringBuilder class:

Constructor	Description
StringBuilder()	It creates an empty String Builder with the initial capacity of 16.
StringBuilder(String str)	It creates a String Builder with the specified string.
StringBuilder(int length)	It creates an empty String Builder with the specified capacity as length.

Important methods of StringBuilder class

Method	Description
public StringBuilder append(String s)	It 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.
public StringBuilder insert(int offset, String s)	It 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.
public StringBuilder replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
public StringBuilder reverse()	It is used to reverse the string.
public int capacity()	It is used to return the current capacity.
public void ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	It is used to return the character at the specified position.
public int length()	It is used to return the length of the string i.e. total number of characters.
public String substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
public String substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

1) StringBuilder append() method: **The StringBuilder append() method concatenates the given argument with this String.**

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

Output: Hello Java

2) StringBuilder insert() method: **The StringBuilder insert() method inserts the given string with this string at the given position.**

```
class StringBuilderExample2{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

Output:

```
16
16
34
34
```

Output: HJavaello

3) StringBuilder replace() method: **The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.**

```
class StringBuilderExample3{
      public static void main(String args[]){
      StringBuilder sb=new StringBuilder("Hello");
      sb.replace(1,3,"Java");
      System.out.println(sb);//prints HJavalo
      }
      }
```

Output: HJavalo

4) StringBuilder delete() method: **The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.**

```
class StringBuilderExample4{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

Output: Hlo

5) StringBuilder reverse() method: **The reverse() method of StringBuilder class reverses the current string.**

```
class StringBuilderExample5{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

Output: olleH

6) StringBuilder capacity() method: The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder 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 StringBuilderExample6{
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity());//default 16
```

Output:

```
16
16
34
```

```
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
}
}
```

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

```
class StringBuilderExample7{
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
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
sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34
sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70
}
}
```

6. Write a program to find the largest and smallest numbers from the given set of integers using arrays.

```
import java.util.Scanner;
public class MaxMin{
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements:");
        int n = scanner.nextInt();
        int[] array = new int[n];
        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            array[i] = scanner.nextInt();
        }
        int largest = array[0];
        int smallest = array[0];
        for (int i = 1; i < n; i++) {
            if (array[i] > largest) {
                largest = array[i];
            }
            if (array[i] < smallest) {
                smallest = array[i];
            }
        }
        System.out.println("Largest number is " + largest);
        System.out.println("Smallest number is " + smallest);
    }
            }
```

7. Write a java program to implement matrix multiplication with all validations.

```
import java.util.Scanner;

public class MatrixMultiplication {

public static void main(String[] args) {
  Scanner scanner = new Scanner(System.in);
  System.out.println("Enter number of rows and columns for first matrix:");
  int row1 = scanner.nextInt();
  int col1 = scanner.nextInt();
  int[][] matrix1 = new int[row1][col1];
  System.out.println("Enter elements of first matrix:");
  for (int i = 0; i < row1; i++) {
      for (int j = 0; j < col1; j++) {
          matrix1[i][j] = scanner.nextInt();
      }
  }
  System.out.println("Enter number of rows and columns for second matrix:");
  int col2 = scanner.nextInt();
  int row2 = scanner.nextInt();

  if (col1 != row2) {
      System.out.println("Matrix multiplication not possible. Incorrect dimensions.");
      return;
  }

  int[][] matrix2 = new int[row2][col2];
  System.out.println("Enter elements of second matrix:");
  for (int i = 0; i < row2; i++) {
      for (int j = 0; j < col2; j++) {
          matrix2[i][j] = scanner.nextInt();
      }
  }

  int[][] result = new int[row1][col2];
  for (int i = 0; i < row1; i++) {
      for (int j = 0; j < col2; j++) {
          result[i][j]=0;
          for (int k = 0; k < col1; k++) {
              result[i][j] += matrix1[i][k] * matrix2[k][j];
          }
      }
  }

  System.out.println("Product of entered matrices:");
  for (int i = 0; i < row1; i++) {
      for (int j = 0; j < col2; j++) {
          System.out.print(result[i][j] + " ");
      }
      System.out.println();
  }
 }
}
```

8.Write a java program to implement

a) Matrix addition
b) Matrix subtraction
c) Matrix transpose

a)Matrix addition

```
import java.util.Scanner;

public class MatrixAddition {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter number of rows and columns for first matrix:");
        int row1 = scanner.nextInt();
        int col1 = scanner.nextInt();

        int[][] matrix1 = new int[row1][col1];
        System.out.println("Enter elements of first matrix:");
        for (int i = 0; i < row1; i++) {
            for (int j = 0; j < col1; j++) {
                matrix1[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Enter number of rows and columns for second matrix:");
        int col2 = scanner.nextInt();
        int row2 = scanner.nextInt();

        int[][] matrix2 = new int[row2][col2];
        System.out.println("Enter elements of second matrix:");
        for (int i = 0; i < row2; i++) {
            for (int j = 0; j < col2; j++) {
                matrix2[i][j] = scanner.nextInt();
            }
        }

        int[][] result = new int[row1][col2];
        for (int i = 0; i < row1; i++) {
            for (int j = 0; j < col2; j++) {

                    result[i][j] += matrix1[i][j] + matrix2[i][j];

            }
        }

        System.out.println("Addition of entered matrices:");
        for (int i = 0; i < row1; i++) {
            for (int j = 0; j < col2; j++) {
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

b)Matrix subtraction

```
import java.util.Scanner;

public class MatrixSubtraction{

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter number of rows and columns for first matrix:");
        int row1 = scanner.nextInt();
        int col1 = scanner.nextInt();

        int[][] matrix1 = new int[row1][col1];
        System.out.println("Enter elements of first matrix:");
        for (int i = 0; i < row1; i++) {
            for (int j = 0; j < col1; j++) {
                matrix1[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Enter number of rows and columns for second matrix:");
        int col2 = scanner.nextInt();
        int row2 = scanner.nextInt();

        int[][] matrix2 = new int[row2][col2];
        System.out.println("Enter elements of second matrix:");
        for (int i = 0; i < row2; i++) {
            for (int j = 0; j < col2; j++) {
                matrix2[i][j] = scanner.nextInt();
            }
        }

        int[][] result = new int[row1][col2];
        for (int i = 0; i < row1; i++) {
            for (int j = 0; j < col2; j++) {

                    result[i][j] += matrix1[i][j] - matrix2[i][j];

            }
        }

        System.out.println("Subtraction of entered matrices:");
        for (int i = 0; i < row1; i++) {
            for (int j = 0; j < col2; j++) {
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

c)Matrix transpose

```
import java.util.Scanner;
public class MatrixTranspose {
    public static void main(String args[]) {
    System.out.println("Enter total rows and columns: ");
    Scanner s = new Scanner(System.in);
    int row = s.nextInt();
    int column = s.nextInt();
    int a[][] = new int[row][column];
    System.out.println("Enter values of a matrix:");
    for(int i = 0; i < row; i++)
    {
        for(int j = 0; j < column; j++)
            {
            a[i][j] = s.nextInt();
            }
    }
    System.out.println("The above matrix before Transpose is ");
    for(int i = 0; i < row; i++)
        {
            for(int j = 0; j < column; j++)
            {
            System.out.print(a[i][j]+" ");
            }
            System.out.println(" ");
        }
    System.out.println("The above matrix after Transpose is ");
    for(int i = 0; i < column; i++){
            for(int j = 0; j < row; j++) {
                System.out.print(a[j][i]+" ");
            }
            System.out.println(" ");
        }
    }
}
```

9.Write a program to sort the given strings in the ascending order using arrays

```
      import java.util.Arrays;
      public class StringSorter {
      public static void main(String[] args) {
        String[] str = {"hello", "world", "abc", "xyz", "java", "programming"};
        sortStrings(str);
        System.out.println(Arrays.toString(str));
     }
     public static void sortStrings(String[] str) {
        for (int i = 0; i < str.length - 1; i++) {
         for (int j = i + 1; j < str.length; j++) {
              if (str[i].compareTo(str[j]) > 0) {
                  String temp = str[i];
                  str[i] = str[j];
                  str[j] = temp;
              }
         }
        }
      }
    }
```

10. Write a java program to check the given string is palindrome or not.

```
  class Palindrome{
  public static void main(String[] args) {

    String str = "Radar", reverseStr = "";

    int strLength = str.length();

    for (int i = (strLength - 1); i >=0; --i) {
      reverseStr = reverseStr + str.charAt(i);
    }
    if (str.equals(reverseStr)) {
      System.out.println(str + " is a Palindrome String.");
    }
    else {
      System.out.println(str + " is not a Palindrome String.");
    }
  }
}
```

11.Write a java program to delete the number of characters from the string.

```
public class RemoveCharacters {
    public static void main(String[] args) {
        String str = "Hello, World!";
        int charsToRem = 7;
        String newStr = str.substring(0, str.length() - charsToRem);
        System.out.println(newStr);
    }
}
```

12.Write a java program to implement 2-D array given below:

1				
2	3			
4	5	6		
7	8	9	10	
11	12	13	14	15

```
public class Pattern {
  public static void main(String[] args) {
    int rows = 5, number = 1;
    for(int i = 1; i <= rows; i++) {
      for(int j = 1; j <= i; j++) {
        System.out.print(number + " ");
        ++number;
      }
      System.out.println();
    }
  }
}
```

***(Optional Question)**

Difference between String and StringBuffer Class

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

SNo.	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate t strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory

Difference between StringBuffer and StringBuilder Class

Java provides three classes to represent a sequence of characters: String, StringBuffer, and StringBuilder. The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable. There are many differences between StringBuffer and StringBuilder. The StringBuilder class is introduced since JDK 1.5.
A list of differences between StringBuffer and StringBuilder is given below:

No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

Java Enum ordinal() Method

The ordinal() method of Enum class returns the ordinal of this enumeration constant. This method is designed to be used by sophisticated enum-based data structures, like EnumSet and EnumMap.

Syntax

```
public final int ordinal()
```

Return Value: The ordinal() method returns this enumeration constant's ordinal.

Example

```
public class Enum_ordinalMethodExample1 {
  enum Colours{
  Red,Green,Brown,Yellow;
    }
  public static void main(String[] args) {
  Colours Red = Colours.Red;
  Colours Green = Colours.Green;
  Colours Brown = Colours.Brown;
  Colours Yellow = Colours.Yellow;
  System.out.println(" Red ordinal = "+Red.ordinal());
  System.out.println(" Green Ordinal = "+Green.ordinal());
  System.out.println(" Brown Ordinal = "+Brown.ordinal());
  System.out.println(" Yellow Ordinal = "+Yellow.ordinal());
} }
```

Output:

```
Red ordinal = 0
Green Ordinal = 1
Brown Ordinal = 2
Yellow Ordinal = 3
```

*(Optional Question)

BufferedReader and BufferedWriter Classes in Java

BufferedReader is a class in Java that reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, lines and arrays. The buffer size may be specified. If not, the default size, which is predefined, may be used.

In general, each read request made of a Reader causes a corresponding read request to be made of the underlying character or byte stream. It is therefore good practice to wrap a *BufferedReader* around any Reader whose ***read()*** operations may be costly, such as *FileReaders* and *InputStreamReaders*. For example,

FileReader reader = new FileReader("MyFile.txt");
BufferedReader br = new BufferedReader(reader);

will buffer the input from the specified file. Without buffering, each invocation of ***read()*** or ***readLine()*** could cause bytes to be read from the file, converted into characters, and then returned, which can be very inefficient.

A ***BufferedWriter*** on the other hand is a java class that writes text to a character-output stream, while buffering characters so as to provide for the efficient writing of single characters, strings and arrays.

A ***newLine()*** method is provided, which uses the platform's own notion of line separator as defined by the system property line.separator. Calling this method to terminate each output line is therefore preferred to writing a newline character directly.

In most occasions, a *Writer* sends its output immediately to the underlying character or byte stream. Unless prompt output is required, it is good practice to wrap a *BufferedWriter* around any *Writer* whose ***write()*** operations may be costly, such as *FileWriters* and *OutputStreamWriters*. For example,

File outputFile = new File("output.txt");
BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile));

Both *BufferedReader* and *BufferedWriter* are used in order to achieve greater efficiency through use of buffers. A data buffer is generally a region in memory that is temporarily used. Instructing a BufferedWriter to write on a file, it does not do it directly. Rather, it stores what you want it to write in a buffer and writes it onto the file when you tell it to perform a flush operation. Flushing is the operation that tells the BufferedWriter to write everything onto the output file. Using a buffer is what makes both *BufferedReader* and *BufferedWriter* fast and efficient.

*(Optional Question)

PROGRAMS ON ARRAYS

1.Java Program to illustrate how to declare, instantiate, initialize and traverse the Java array.

```
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
```

Output:

```
10
20
70
40
50
```

```
}}
```

2. //Java Program to print the array elements using for-each loop

```
class Testarray1{
public static void main(String args[]){
int arr[]={33,3,4,5};
//printing array using for-each loop
for(int i:arr)
System.out.println(i);
}}
```

Output:

```
33
3
4
5
```

3. Write a java program to perform matrix multiplication?

```
public class MatrixMultiplicationExample{
public static void main(String args[]){
int a[][]={{1,1,1},{2,2,2},{3,3,3}};
int b[][]={{1,1,1},{2,2,2},{3,3,3}};
int c[][]=new int[3][3];  //3 rows and 3 columns
  //multiplying and printing multiplication of 2 matrices
for(int i=0;i<3;i++)
{
      for(int j=0;j<3;j++)
     {
       c[i][j]=0;
       for(int k=0;k<3;k++)
      {
        c[i][j]+=a[i][k]*b[k][j];
      }//end of k loop
      System.out.print(c[i][j]+" ");  //printing matrix element
     }//end of j loop
System.out.println();//new line
}
}
}
```

OutPut:

```
6 6 6
12 12 12
18 18 18
```