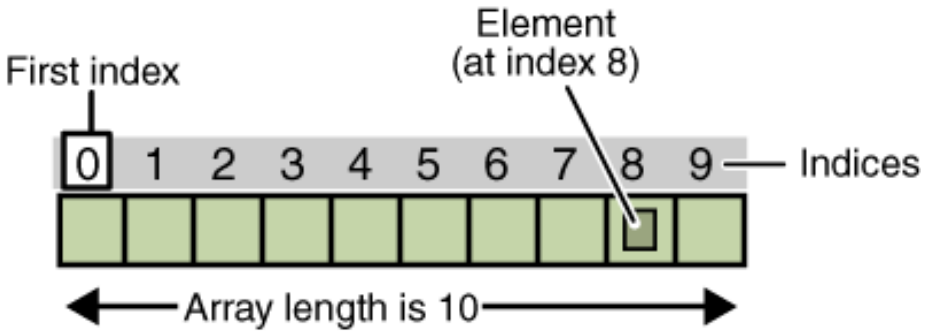
Array

- java provides a data structure Array, which stores a fixed-size sequential collection of elements of the same type
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type
- Each item in an array is called an element, and each element is accessed by its numerical index. As shown in the preceding illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.



characteristics of arrays in Java:

- Fixed length: Once an array is created, we cannot change its size. So consider using arrays when the numbers of elements are known and fixed. Otherwise, you should consider using another dynamic container such as `ArrayList`.
- Fast access: It's very fast to access any elements in an array (by index of the elements) in constant time: accessing the 1st element takes same time as accessing the last element. So performance is another factor when choosing arrays.
- An array can hold primitives or objects.
- An array of primitives stores values of the primitives.
- An array of objects stores only the references to the objects.
- In Java, the position of an element is specified by index which is zero-based. That means the first element is at index 0, the second element at index 1, and so on.
- An array itself is actually an object.

Declaring a Variable to Refer to an Array

- Declare an Array

  ```
  <DataType>[] ArrVarName;
  ```

- Like declarations for variables of other types, an array declaration has two components: the Data Type and the array's name.
- An array's type is written as type[], where type is the data type of the contained elements
- The size of the array is not part of its type (which is why the brackets are empty).
- An array's name can be anything you want, provided that it follows the rules and conventions

- It simply tells the compiler that this variable will hold an array of the specified type
- Example
    - byte[] arrayRefVar;
    - short[] arrayRefVar;
    - long[] arrayRefVar;
    - float[] arrayRefVar;
    - double[] arrayRefVar;
    - boolean[] arrayRefVar;
    - char[] arrayRefVar;
    - String[] arrayRefVar;

Creating Arrays

- You can create an array by using the new operator with the following 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];
```

Arrays declarations and initialization

Declare first and initialize later:

```
int [] numbers = new int[10];
```

This declares an array object to hold 10 integer numbers (primitive array).

When declaring an array of primitive type, remember these rules:

- All numbers are initialized to zeroes by default. That means the above array numbers contain 10 numbers which are all zeroes, even we haven't initialized the array yet.
- Boolean elements are initialized to false by default.

Then we initialize values for each element of the array like this:

```
numbers[0] = 10;
numbers[1] = 500;
numbers[2] = 1000;
…
```

The following statement declares an array of String objects:

```
String[] names = new String[5];
```

This array holds 5 String objects. And by default, all elements of `Object` type are initialized to null.

NOTE: In Java, you can place the brackets **[]** either after the type or after the variable name. Hence these declarations are both correct:

```
String[] names = new String[5];
String titles[] = new String[10];
```

However, it's recommended to use the **[]** after the type for readability: You can easily realize this is an array of Strings, or that is an array of integer numbers.

You can also declare and initialize elements of an array in one statement. For example:

```
int[]  numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

This creates an array with 10 integer numbers which are initialized up declaration. This is a handy shortcut for declaring arrays whose elements are already known at compile time.

This statement declares and initializes an array of Strings:

```
String[] columnNames = {"No", "Name", "Email", "Address"};
```

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];
  ```

Arrays Manipulation

We access elements in the array by index (remember 0-based):

```
String firstColumn = columnNames[0];
```

This statement takes value of the first element in the String array `columnNames` and assigns it to the variable `firstColumn`.

The following statement illustrates accessing an element in a 2D array:

```
String[] firstRow = sampleData[0];
```

This gets the first element in the `sampleData` array, which returns an array.

The following statement takes the element at 3rd row and 2nd column in the above 2D array:

```
String letter = sampleData[4][3];
```

And these examples show how to assign values to elements in arrays:

```
columnNames[2] = "Phone;
columnNames[4] = new String("City");
numbers[4] = 1024;
sampleData[4][2] = "xyz";
```

A common operation is iterating an array using a loop statement like the `for` statement. The following example uses the `for` loop to iterate over all elements in an array of integer numbers, and prints value of each element:

```
 int[]  numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

for (int i = 0; i < numbers.length; i++) {
    System.out.println(numbers[i]);
}
```

For arrays of objects you can use the *for each* syntax to iterate. For example:

```
String[] nameList = {"Tom", "Mary", "Peter", "John", "Adam",
"Justin"};

for (String aName : nameList) {
    System.out.println(aName);
}
```

Program N0. 52

Array Demo

```
public class ArrayDemo1_52
{
	public static void main(String[] args)
	{
		int[] arr;  //declaration of array reference variable
		arr = new int[5]; //creating array of 5 elements
		int[] brr;
		arr[0] = 10;
		arr[1] = 20;
		arr[2] = 30;
		arr[3] = 40;
		arr[4] = 50;
		System.out.println("\n\t 1st Value = " + arr[0]);
	}
}
```

Program N0. 53

Input an array of 5 elements and display using for/foreach loop

```
import java.util.Scanner;
public class ArrayDemo2_53
{
	public static void main(String[] args)
```

```
	{
		Scanner scan = new Scanner(System.in);
		int[] arr = new int[5];
		System.out.println("\n\t Length of arr = " + arr.length);
		for(int i=0; i<arr.length; i++)
		{
			System.out.print("\n\t Enter Value :");
			arr[i] = scan.nextInt();
		}
		System.out.println("\n\t Array Values are :");
		for(int i=0; i<arr.length; i++)
		{
			System.out.print("\t" + arr[i]);
		}
	}
}
```

Program N0. 54

Input an array of 5 elements and display all odd and even numbers

```
import java.util.Scanner;
class Java54
{
	public static void main(String []args)
	{
		Scanner scan = new Scanner(System.in);

		int size,i;
		System.out.print("\n\t Enter Size of Array :");
		size = scan.nextInt();
		int arr[] = new int[size];
		System.out.print("\n\t Enter "+ size +" Array Values :");
		for(i=0;i<size;i++)
		{
			arr[i] = scan.nextInt();
		}
		System.out.print("\n\t Even Array values are :");
		for(i=0;i<size;i++)
		{
			if(arr[i]%2==0)
```

```
                        System.out.print(" " + arr[i]);
            }
            System.out.print("\n\t Odd Array values are :");
            for(i=0;i<size;i++)
            {
                  if(arr[i]%2==1)
                        System.out.print(" " + arr[i]);
            }
      }
}
```

Program N0. 55

Input an array of 5 elements and display all max and min numbers

```
import java.util.Scanner;
class Java55
{
      static Scanner scan;
      public static void main(String []args)
      {
            scan = new Scanner(System.in);
            int size,arr[];

            System.out.print("\n\t Enter Size of Array:");
            size = scan.nextInt();

            arr = new int[size];

            createArray(arr,size);
            showArray(arr);
            System.out.println("\n\t Max Value = " + showMax(arr));

            System.out.println("\n\t Min Value = " + showMin(arr));
      }

      public static void createArray(int x[],int size)
      {
            scan = new Scanner(System.in);
            System.out.print("\n\t Enter "+ size +" Array values :");
            for(int i=0;i<size;i++)
```

```
            {
                  x[i] = scan.nextInt();
            }
            System.out.println("\n\t Array Created.");
      }

      public static void showArray(int x[])
      {
            System.out.print("\n\t Array values are :");
            for(Object obj:x)
            {
                  System.out.print(" " + obj);
            }
      }

      public static int showMax(int x[])
      {
            int max = x[0];   //10
            for(int i=0;i<x.length;i++)
            {
                  if(x[i]>max)
                  {
                        max = x[i];
                  }
            }
            return max;
      }

      public static int showMin(int x[])
      {
            int min = x[0];   //10
            for(int i=0;i<x.length;i++)
            {
                  if(x[i]<min)
                  {
                        min = x[i];
                  }
            }
```

```
            return min;
        }
}
```

Program N0. 56

Input an array of 5 elements and search the given number

Program N0. 57

Input an array of 5 elements and count the occurrences of given number

Program N0. 58

Input an array of Strings and store names

Array Methods

Besides the fundamental operations like getting and setting, Java provides various functions to manipulate arrays in the **`Arrays`** class.

The `Arrays` is a utility class which can be found in the `java.util` package. Here are some noteworthy methods it provides:

- **`asList()`**: returns a fixed-size list backed by an array.
- **`binarySearch()`**: searches for a specific value in an array. Returns the index of the element if found, or -1 if not found. Note that the array must be sorted first.
- **`copyOf()`**: copies a portion of the specified array to a new one.
- **`copyOfRange()`**: copies a specified range of an array to a new one.
- **`equals()`**: compares two arrays to determine if they are equal or not.
- **`fill()`**: fills same values to all or some elements in an array.
- **`sort()`**: sorts an array into ascending order.
- And other methods you can find in the `Arrays` class Javadoc.

In addition, the `System.arraycopy()` is an efficient method for copying elements from one array to another. Remember using this method instead of writing your own procedure because this method is very efficient.

Copying Arrays

The **`System`** class has an **`arraycopy`** method that you can use to efficiently copy data from one array into another:

```
public static void arraycopy(Object src, int srcPos,
                             Object dest, int destPos, int length)
```

The two **`Object`** arguments specify the array to copy *from* and the array to copy *to*. The three **`int`** arguments specify the starting position in the source array, the starting position in the destination array, and the number of array elements to copy.

The following program, **ArrayCopyDemo**, declares an array of **char** elements, spelling the word "decaffeinated." It uses the **System.arraycopy** method to copy a subsequence of array components into a second array:

```
class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                            'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

Program N0. 59

Passing Arrays to Methods:

Just as you can pass primitive type values to methods, you can also pass arrays to methods

```
import java.util.Scanner;

public class ArrayDemo59
{
      static void showArray(int[] args)
      {
        System.out.println("\n\t Inside Method:Values are :");
        for(int value:args)
         {
            System.out.print("\t" + value);
         }
      }

      public static void main(String[] args)
      {
            Scanner scan = new Scanner(System.in);
            int[] arr= new int[5];

            System.out.print("\n\t Enter 5 Values :");
            for(int i=0; i<arr.length; i++)
            {
                  arr[i] = scan.nextInt();
            }
            showArray(arr);
```

```
	}
}
```

Program N0. 60

Returning an Array from a Method: A method may also return an array.
Program to return reverse of array using method

```
import java.util.Scanner;

public class ArrayDemo60
{
	static int[] reverseArr(int[] list)
	{
		int[] result = new int[list.length];
		int j = result.length-1;
		for(int i=0; i<list.length; i++)
		{
			result[j] = list[i];
			j--;
		}
		return result;
	}
	public static void main(String[] args)
	{
		Scanner scan = new Scanner(System.in);
		int[] arr= new int[5];

		System.out.print("\n\t Enter 5 Values :");
		for(int i=0; i<arr.length; i++)
		{
			arr[i] = scan.nextInt();
		}
		arr = reverseArr(arr);
		System.out.println("\n\t Reverse :");
		for(int value : arr)
		{
			System.out.print("\t" + value);
		}
	}
```

```
}
```

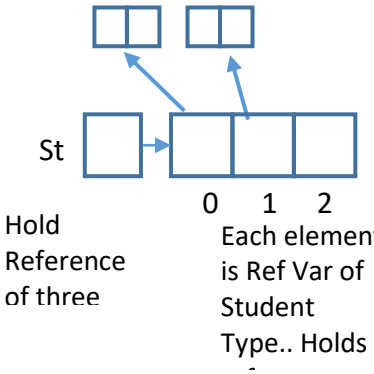
Array of Objects

Arrays are capable of storing objects also. For example, we can create an array of Strings which is a reference type variable.

However, using a String as a reference type to illustrate the concept of array of objects isn't too appropriate due to the immutability of String objects.

Therefore, for this purpose, we will use a class Student containing a single instance variable marks

```
class Student
{
	int roll;
	String name;
	void getStud(int r, String n)
	{
		roll = r;
		name = n;
	}
	void showStud()
	{
		System.out.print("\n\t" + roll + "\t" + name);
	}
}
```



`Student[] stud;`

- The statement creates the array which can hold references to Student objects. It doesn't create the Student objects themselves.
- They have to be created separately using the constructor of the Student class. The student Array contains three memory spaces in which the address of three Student objects may be stored.

```
stud = new Student[3];
Stud[0] = new Student();
Stud[1] = new Student();
Stud[2] = new Student();
```

Program N0. 61

Array of Objects

```
class Student
{
	int roll;
	String name;
```

```
	void getStud(int r, String n)
	{
		roll = r;
		name = n;
	}
	void showStud()
	{
		System.out.print("\n\t" + roll + "\t" + name);
	}
}
public class ArrayDemo61
{
	public static void main(String[] args)
	{
		Student[] stud;
		stud = new Student[3];

	/*	stud[0] = new Student();
		stud[1] = new Student();
		stud[2] = new Student();*/

		for(int i=0; i<stud.length; i++)
		{
			stud[i] = new Student();
		}
		stud[0].getStud(101, "Bhartesh");
		stud[1].getStud(102, "Payal");
		stud[2].getStud(103, "Pankaj");

		for(int i=0; i<stud.length; i++)
		{
			stud[i].showStud();
		}
	}
}
```

Program N0. 62

Array of Objects

Employee Class

```
import java.util.Scanner;
class Employee
{
	int empid;
	String name;
	double bsal;

	void getEmp(int id, String n, double sal)
	{
		empid = id;
		name = n;
		bsal = sal;
	}
	void processSal()
	{
		double da = bsal*20/100;
		double hra = bsal*40/100;
		double gsal = bsal+da+hra;

		System.out.println("\n\t " + empid + "\t" + name +
			"\t" + gsal);
	}
}
public class ArrayDemo62
{
	public static void main(String[] args)
	{
		int choice;
		Scanner scan = new Scanner(System.in);

		Employee[] emparr = new Employee[5];
		int cnt = 0;

		for(int i=0; i<emparr.length; i++)
		{
			emparr[i] = new Employee();
```

```
        }

    do
    {
        System.out.println("\n\t 1. Add Employee \n\t 2.
              Process Sal \n\t 3. Quit ");
        System.out.print("\n\t Enter your choice :");
        choice = scan.nextInt();

        switch(choice)
        {
            case 1:
              System.out.print("\n\t Enter id:");
              int id = scan.nextInt();

              System.out.print("\n\t Enter Name :");
              String name = scan.next();

              System.out.print("\n\t Enter Bsal :");
              double sal = scan.nextDouble();

              emparr[cnt].getEmp(id, name, sal);
              cnt++;
              break;
            case 2:
                  System.out.println("\n\n\t Salary
                       Processed");
                  for(int i=0; i<cnt;i++)
                  {
                        emparr[i].processSal();
                  }
               break;

            case 3:
               break;

            default:
                System.out.println("Invalid Input");
```

```
            }//switch ends
      }while(choice!=3);
      }
}
```

Program N0. 63

Array of Objects Bank Account Class

```
import java.util.Scanner;
class BankAcc
{
      static int number;
      int accno;
      String name;
      float balance;
      Scanner scan = new Scanner(System.in);

      void openAccount()
      {
            accno = generateAccNumber();
            System.out.println("\n\t Account Number = " + accno);

            System.out.print("\n\t Enter your Name :");
            name = scan.next();

            System.out.print("\n\t Enter Initial Amount :");
            balance = scan.nextFloat();
      }

      private int generateAccNumber()
      {
                  return ++number;
      }

      public void viewAccount()
      {
            System.out.println("\n\t Account Number = " + accno);
            System.out.println("\n\t Account Holder Name = " + name);
            System.out.println("\n\t Total Available Balance = Rs. " +
balance);
```

```
	}

}

class Java62
{
	static int cnt;
	static BankAcc acc[] = new BankAcc[10];

	public static void main(String []args)
	{
		int choice,accnumber,pos;
		float amt;
		Scanner scan = new Scanner(System.in);

		do
		{
			System.out.println("\n\n ** Menu **");
			System.out.println("\n 1. Open Acc \n 2. View Acc \n 3.
Deposite");
			System.out.println(" 4. Withdraw \n 5. Exit");

			System.out.print("\n\t Enter Choice :");
			choice = scan.nextInt();

			switch(choice)
			{
				case 1:
					acc[cnt] = new BankAcc();
					acc[cnt].openAccount();
					cnt++;
				break;

				case 2:
					System.out.print("\n\t Enter Account Number to
Search

:");
					accnumber = scan.nextInt();
```

```
                              pos = searchAcc(accnumber);
                              if(pos>=0)
                              {
                                    acc[pos].viewAccount();
                              }
                              else
                              {
                                    System.out.println("\n\t Accuont not
found.");
                              }
                        break;

                        case 3:
                              System.out.print("\n\t Enter Account Number to
Search :");
                              accnumber = scan.nextInt();

                              pos = searchAcc(accnumber);
                              if(pos>=0)
                              {
                                    System.out.print("\n\t Enter Amount :");
                                    amt = scan.nextFloat();

                                    acc[pos].balance = acc[pos].balance + amt;
                                    System.out.println("\n\t Amuont Rs. " + amt
+ " is Deposited Successfully.");
                              }
                              else
                              {
                                    System.out.println("\n\t Accuont not
found.");
                              }
                        break;
                        case 4:
                        break;
                        case 5:
                        break;
```

```
                        default:
                              System.out.println("\n\t Invalid Case.");
                  }
            }
            while(choice!=5);
      }
      // 1 2 3 4 5
      static int searchAcc(int accno)
      {
                  int flag=-1;
                  for(int i=0;i<cnt;i++)
                  {
                        if(accno==acc[i].accno)
                        {
                              flag = i;
                        }
                  }
                  if(flag>=0)
                        return flag;
                  else
                        return -1;
      }
}
```

Multidimensional Arrays

In some cases, we may want to divide the list in delimited sections. For example, if we create a list of names, we may want one part of the list to include family members and another part of the list to include friends.

Instead of creating a second list, we can add a second dimension to the list. In other words, we would create a list of a list, or one list inside of another list, although the list is still made of items with common characteristics

A multidimensional array is a series of arrays so that each array contains its own sub-array(s)

Two Dimensional Arrays

The most basic multidimensional array is made of two dimensions. This is referred to as two-dimensional. To create a two-dimensional array, inside of one pair of square brackets, use two pairs (of square brackets)

```
DataType[][] VariableName;
```

Two-dimensional arrays are used whenever the model data is best represented with rows and columns, or has two varying aspects (eg, gender and age, weight and height, ...)

Visualizing two-dimensional arrays

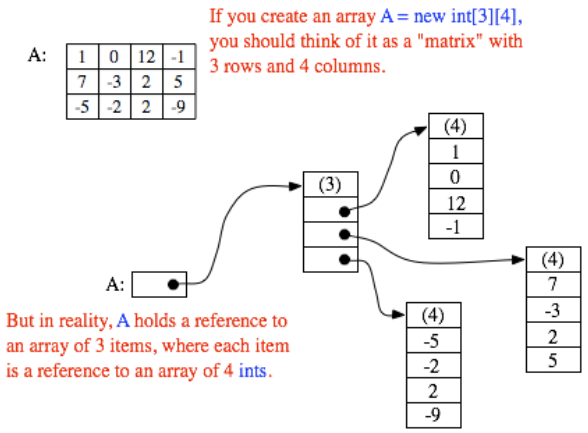
2-dimensional arrays are usually represented with a row-column "spreadsheet" style. Assume we have an array arr with two rows and four columns

```
int[][] a = new int[2][4];  // Two rows and four columns
```

a[0][0]	**a[0][1]**	**a[0][2]**	**a[0][3]**
a[1][0]	**a[1][1]**	**a[1][2]**	**a[1][3]**

```
String[][] members = new String[2][4];
```

In above declaration, the members variable contains two lists. Each of the two lists contains 4 elements. This means that the first list contains 4 elements and the second list contains 4 elements. Therefore, the whole list is made of 8 elements (2 * 4 = 8). Because the variable is declared as a String, each of the 8 items must be a string



Initialize values

- You can assign initial values to an array in a manner very similar to one-dimensional arrays, but with an extra level of braces.
- The dimension sizes are computed by the compiler from the number of values.

Example

```
int[][] board = new int[][]{ {1,0,0}, {0,1,0}, {1,2,1} };
```

Program N0. 64

Simple Program Multidimensional Array

```
public class ArrayDemo64
{
      public static void main(String[] args)
      {
            int[][] arr;
            arr = new int[2][4];

            System.out.println("\n\t length of arr = " +
              arr.length);
            System.out.println("\n\t length of arr[0] = " +
              arr[0].length);
```

```
            System.out.println("\n\t length of arr[1] = " +
              arr[1].length);

            arr[0][0] = 10;
            arr[0][1] = 20;
            arr[0][2] = 30;
            arr[0][3] = 40;

            arr[1][0] = 50;
            arr[1][1] = 60;
            arr[1][2] = 70;
            arr[1][3] = 80;

            System.out.println("\n\t value of arr[0][0] = " + arr[0][0]);
            System.out.println("\n\t Values using for loop :");

            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("\n");
            }
      }
}
```

Program NO. 65

Input and Display roll no. and marks of 3 subjects

```
import java.util.Scanner;

public class ArrayDemo65
{
      public static void main(String[] args)
      {
```

```
            Scanner scan = new Scanner(System.in);
            int[][] stud = new int[5][4];
            int r;
            System.out.println("\n\t Enter Records :");
            for(int i=0; i<stud.length; i++)
            {
                  r = i+1;
                  System.out.print("\n\t Record No. : " + r);
                  System.out.print("\n\t Enter Roll No. :");
                  stud[i][0] = scan.nextInt();

                  System.out.print("\n\t Enter Sub1 :");
                  stud[i][1] = scan.nextInt();

                  System.out.print("\n\t Enter Sub2 :");
                  stud[i][2] = scan.nextInt();

                  System.out.print("\n\t Enter Sub3 :");
                  stud[i][3] = scan.nextInt();
            }

            System.out.println("\n\tRollNO\tSub1\tSub3\tSub4");
            System.out.println("\n\t--------------------\n\t");

            for(int i=0; i<stud.length; i++)
            {
                  for(int j=0; j<stud[i].length; j++)
                  {
                        System.out.print("\t" + stud[i][j]);
                  }
                  System.out.println();
            }
      }
}
```

Assignment

Program N0. 66	**Display the total number of accidents each year**

Classes and Objects

- Java is an object-oriented programming language therefore the underlying structure of all java programs is classes
- Everything to be represented in a java program must be encapsulated in a class that defines the state and behavior of objects
- A class is a user defined data type. Once the class type has been defined, we can create "variables" of that type. These "variables" are called instances or objects of classes
- An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tengible and intengible). The example of integible object is banking system.
- An object has three characteristics:
    - **state:** represents data (value) of an object.
    - **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
    - **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely
- For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.
- **Object is an instance of a class**
    - Class is a template or blueprint from which objects are created. So object is the instance(result) of a class
- A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.
- A class in java can contain:
    - **Data member**
    - **Method**
    - **Constructor**
    - **Block**
    - **Class and interface**

```
Syntax to declare a class:
class <class_name>
{
    datamember;
    method();
}
```

- Classes create objects and objects use methods.
- Classes provide a convenient method for packing together a group of logically related data items and functions.
- In java the data items are called fields and the functions are called methods.

- **Object -** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.
- **Class -** A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.

Encapsulation

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class, therefore it is also known as data hiding.

Program N0. 67

Class Student

```
class Student
{
	private int roll; //fields/data members/instance memberes
	private  String name;

	public void createStudent(int r, String n)
	{
		roll  = r;
		name = n;
	}
	public void showStudent()
	{
		System.out.println("\n\t Roll No. :" + roll);
		System.out.println("\n\t Name : " + name);
	}
}

public class ClassDemo1_67
{
	public static void main(String[] args)
	{
		Student stud1 = new Student();
		stud1.createStudent(101, "Kavita");
		stud1.showStudent();
	}
}
```

Methods

- A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println() method, for example, the system actually executes several statements in order to display a message on the console.

Creating Method

- Method definition consists of a method header and a method body

```
modifier returnType nameOfMethod (Parameter List)
{
 // method body
}
```

- **Modifier**: It defines the access type of the method and it is optional to use.
- **Return Type**: Method may return a value.
- **Name of method**: This is the method name. The method signature consists of the method name and the parameter list.
- **Parameter List**: The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- **Method body**: The method body defines what the method does with statements.

Program N0. 68

Class Employee

```
class Employee
{
	private int empId;
	private String empName;
	private double sal;

	public void getEmp(int id, String ename, double amt)
	{
		empId = id;
		empName = ename;
		sal = amt;
	}
	public void showEmp()
	{
		System.out.println("\n\t Emp Id = " + empId);
		System.out.println("\n\t Emp Name = " + empName);
		System.out.println("\n\t Salary = " + sal);
	}
}
public class ClassEmployee68
{
      public static void main(String[] args)
```

```
	{
		Employee emp1 = new Employee();
		emp1.getEmp();
		emp1.showEmp();

	}
}
```

Passing Parameters by Value

- Passing Parameters by Value means calling a method with a parameter. Through this the argument value is passed to the parameter
- While working under calling process, arguments is to be passed. These should be in the same order as their respective parameters in the method specification. Parameters can be passed by value or by reference

Program N0. 69

Class Product

```
class Product
{
	private int pid;
	private double price, qty, bamt;

	public void getProduct(int id, double pr, double q)
	{
		pid = id;
		price = pr;
		qty = q;
	}

	public void calcBamt()
	{
		bamt = price*qty;
		System.out.println("Product Id = " + pid);
		System.out.println("Price  = " + price);
		System.out.println("Quantity = " + qty);
		System.out.println("Bill Amount = " + bamt);
	}
```

```
}

public class ProductDemo
{
	public static void main(String[] args)
	{
		Product toy1 = new Product();
		toy1.getProduct(101, 350.50, 2.00);
		toy1.calcBamt();
	}
}
```

Program N0. 70

Class Rectangle

```
class Rectangle
{
	public int length, breadth, height;

	public void getData(int l, int b, int h)
	{
		length = l;
		breadth = b;
		height = h;
	}

	public int getArea()
	{
		int area;
		area = length*breadth;
		return area;
	}

	public int getVol()
	{
		return (length*breadth*height);
	}
}

public class ClassRectangle70
```

```
{
	public static void main(String[] args)
	{
		Rectangle rect1 = new Rectangle();
		rect1.getData(10,3,2);

		int area;
		area = rect1.getArea();
		System.out.println("\n\t Area of Rect1 = " + area);
		System.out.println("\n\t Vol of Rect1 = " +
rect1.getVol());
	}
}
```

assignments

Program N0. 71

Define a class BOOK with the following specifications

Private members of the class BOOK are
BOOK NO : integer type
BOOKTITLE : 20 characters
PRICE : float (price per copy)
TOTAL_COST() : A function to calculate the total cost for N number of copies where N is passed to the function as argument

Public members of the class BOOK are
INPUT() : function to read BOOK_NO. BOOKTITLE, PRICE
PURCHASE() : function to ask the user to input the number of copies to be purchased. It invokes TOTAL_COST() and prints the total cost to be paid by the user.

Note : You are also required to give detailed function definitions

Program N0. 72

Define a class in Java with following description:

Private Members
A data member Flight number of type integer
A data member Destination of type string
A data member Distance of type float
A data member Fuel of type float
A member function CALFUEL() to calculate the value of Fuel as per the following criteria

Distance	Fuel
<=1000	500
more than 1000 and <=2000	1100
more than 2000	2200

Public Members

A function FEEDINFO() to allow user to enter values for Flight Number, Destination, Distance & call function CALFUEL() to calculate the quantity of Fuel
A function SHOWINFO() to allow user to view the content of all the data members

Program N0. 73

Define a class in Java with following description:

Private members:
bcode : 4 digits code number
bname : String
innings, notout, runs : integer type
batavg : it is calculated according to the formula
: batavg =runs/(innings-notout)
calcavg() : Function to compute batavg
Public members:
readdata() : Function to accept value from bcode, name, innings, notout
and invoke the function calcavg()
displaydata() : Function to display the data members on the screen

What is Overloading?

Overloading refers to the process of having multiple methods or constructors using a common name but with different argument list. The methods can be in the same class or in both super class and sub class.

- When a class has two or more methods by same name but different parameters, it is known as method overloading
- Method overloading allows to define same name for two or more methods, providing that the number of parameters or the data type of parameters are different.
- Method overloading is one way to realize polymorphism, which means to allow one interface to be used with multiple methods.
- To declare more than one methods, all methods must be declared with different argument list.
- The methods would perform different operations depending on the argument list in the method call.
- The correct method to be invoked is determined by checking the number and type of arguments.

Example #1:

```
public class Dog {
    private String name;

    public Dog() {
        this.name = "Puppy";
    }
```

```
        public Dog(String name) {
            this.name = name;
        }
    }
```

Here, the class `Dog` has two overloaded constructors: the first one with no argument; and the second one with a `String` argument (for name). Then we have two ways for creating `Dog` objects:

```
Dog dog1 = new Dog();    // name is 'Puppy'
Dog dog2 = new Dog("Rex");    // name is 'Rex'
```

Example #2:

```
public class Cat {
    public void catches(Mouse mouse) {
        System.out.print("Catching a mouse...");
    }

    public void catches(Fish fish) {
        System.out.print("Catching a fish...");
    }
}
```

Here, the `Cat` class has two overloaded methods: one for catching a mouse; and another for catching a fish. Both has same name (`catches`) but argument type is different (`Mouse` and `Fish`). So we can write:

```
Cat myCat = new Cat();
myCat.catches(new Mouse());
myCat.catches(new Fish());
```

Example #3:

```
public class Lion extends Cat {

    public void catches(Buffalo buffalo) {
        System.out.print("Catching a buffalo...");
    }
}
```

As you can see, this `Lion` class is a sub type of the `Cat` class. The `Lion` class overloads the `catches()` method in order to catch a buffalo. In this case, the method in the super class is called overloaded method, and the method in the sub class is called overloading method.

We can write:

```
Lion lion = new Lion();
lion.catches(new Mouse());
lion.catches(new Buffalo());
```

Why is Overloading?

At low level, overloading is for re-using method names and constructor names depending on needs. For example, you may want to create a `Dog` with default name 'Puppy', or sometimes you want to create a `Dog` with your own name; Sometimes you may want your cat to catch a mouse, but other times he can catch a fish, etc.

At high level, overloading is for extending functionalities but keeping the flexibility of code.

Overloading Rules

Compared to overriding, overloading is simpler, thus the rules are more relaxed.

Here are the rules with regard to overloaded methods:

- Overloaded methods must have different argument list.
- Overloaded methods may have different return type (the argument list is different as well).
- Overloaded methods may have different access modifiers.
- Overloaded methods may throw different exceptions.

Program N0. 74

Arithmetic Operations

```
public class MethodOverloadDemo_74
{
	static void arithAdd(int a, int b)
	{
		System.out.println("\n\t This is Method No. 1");
		System.out.println("\n\t Num1 = " + a);
		System.out.println("\n\t Num2 = " + b);
		int sum = a + b;
		System.out.println("\n\t Addition = " + sum);
	}
	static void arithAdd(int a, double b)
	{
		System.out.println("\n\t This is Method No. 2");
		System.out.println("\n\t Num1 = " + a);
		System.out.println("\n\t Num2 = " + b);
		double sum = a + b;
		System.out.println("\n\t Addition = " + sum);
	}
	static double arithAdd(double a, int b)
	{
		System.out.println("\n\t This is Method No. 3");
		System.out.println("\n\t Num1 = " + a);
		System.out.println("\n\t Num2 = " + b);
```

```
            double sum = a+b;
            return sum;
      }
      public static void main(String[] args)
      {
            arithAdd(10,20);
      }
}
```

Program N0. 75

Class BankAccount

```
class BankAccount
{
      private int accId;
      private String custName;
      private double bal;

      public void openAccount()
      {
            System.out.println("\n\t Bank Acc Opened");
      }
      public void openAccount(int id, String name)
      {
            System.out.println("\n\t Bank Account opened with
Following
                                          Details :");
            accId = id;
            custName = name;
            bal = 0.00;
            System.out.println("\n\t Account Id = " + accId);
            System.out.println("\n\t Cust Name = " + custName);
            System.out.println("\n\t Balance = " + bal);
      }
      public void openAccount(int id, String name, double iniamt)
      {
      System.out.println("\n\t Bank Account opened with Initial
Amount:");
            accId = id;
            custName = name;
```

```
            bal = iniamt;
            System.out.println("\n\t Account Id = " + accId);
            System.out.println("\n\t Cust Name = " + custName);
            System.out.println("\n\t Balance = " + bal);
      }

}
public class BankAccountDemo_75
{
      public static void main(String[] args)
      {
            BankAccount SB1 = new BankAccount();
            SB1.openAccount();

            BankAccount SB2 = new BankAccount();
            SB2.openAccount(101, "Bhartesh");

            BankAccount SB3 = new BankAccount();
            SB3.openAccount(101, "Payal", 1000.00);
      }
}
```

The Constructors

- A class constructor is a special member function of a class that is executed whenever an object of its associated class is created. It is called constructor because it constructs the values of data members of the class.
- A constructor function has exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.
- Java provides a default constructor which takes no arguments and performs no special actions or initializations, when no explicit constructors are provided.

```
Syntax
class <ClassName>
{
   …datamembers/fields;
   public <ClassName>(argument list)
   {
       …statements;
   }
}
```

Program N0. 76

Class Demo

```
class Demo
{
```

```
	int x, y;
	public Demo()		//Constructor Method
	{
		x = 10;
		y = 20;
	}
	void show()
	{
		System.out.println("\n\t x = " + x);
		System.out.println("\n\t y = " + y);
	}
}

public class ConstructorDemo_76
{
	public static void main(String[] args)
	{
		Demo obj1 = new Demo();
		obj1.show();
	}
}
```

Parameterized Constructor

- Constructors may include parameters of various types. When a constructor is parameterized the object declaration statement must pass the initial values to the parameters
- When the constructor is invoked using the new operator, the types must match those that are specified in the constructor definition

Program N0. 77

Class Rectangle - Parameterized Constructor

```
import java.util.Scanner;

class Rectangle
{
	public int length, breadth;

	public Rectangle(int l, int b)
	{
		length = l;
		breadth = b;
```

```
	}

	public void calcArea()
	{
		int area = length * breadth;
		System.out.println("\n\t Length =  " + length);
		System.out.println("\n\t Breadth = " + breadth);
		System.out.println("\n\t Area = " + area);
	}
}

public class RectangleDemo
{
	public static void main(String[] args)
	{
		Scanner scan = new Scanner(System.in);
		int l, b;

		System.out.print("\n\t Enter Length :");
		l = scan.nextInt();

		System.out.print("\n\t Enter Breadth :");
		b = scan.nextInt();

		Rectangle rect1 = new Rectangle(l, b);
		rect1.calcArea();
	}
}
```

Overloaded Constructors

- Like methods, constructors can also be overloaded. Since the constructors in a class all have the same name as the class, their signatures are differentiated by their parameter lists

Program N0. 78

Class Bank Account

```
class BankAccount
{
	int id;
	String name;
```

```
	double bal;

	public BankAccount()
	{
		System.out.println("Bank Acc Opened");
	}

	public BankAccount(int i, String n)
	{
		id = i;
		name = n;
		bal = 0.00;
		System.out.println("\n\t Bank Acc Opened with
		 Following Details :");
		System.out.println("\n\t id = " + id);
		System.out.println("\n\t Name = " + name);
		System.out.println("\n\t Balance = " + bal);
	}
	public BankAccount(int i, String n, double iniamt)
		{
			id = i;
			name = n;
			bal = iniamt;
			System.out.println("\n\t Bank Acc Opened with
			Following Details :");
			System.out.println("\n\t id = " + id);
			System.out.println("\n\t Name = " + name);
			System.out.println("\n\t Balance =  " + bal);
	}
}

public class BankAccountDemo_78
{
	public static void main(String[] args)
	{
		BankAccount SB1 = new BankAccount(101, "asdfjdsf",
		   1000.00);
	}
}
```

this() constructor

- "**this**" keyword in Java is a special keyword which can be used to represent current object or instance of any class in Java
- "**this**()" statement is used to call constructor of same class in Java (used to call overloaded constructor)
- It is called Explicit Constructor Invocation. If a class has two overloaded constructor one without argument and another with argument. Then this keyword can be used to call Constructor with argument from Constructor without argument

Program N0. 79

this() statement with Constructor

Class Goods

```
class Goods
{
	int id;
	String name;
	String description;
	public Goods()
	{
		System.out.println("\n\t Goods Created");
	}
	public Goods(int i, String n)
	{
		this();
		id = i;
		name = n;
		System.out.println("\n\t id = " + id);
		System.out.println("\n\t Name = " + name);
	}
	public Goods(int i, String n, String des)
	{
		this(i, n);
		description = des;
		System.out.println("\n\t Description = " +
		  description);
	}

}
```

```
public class ThisConstructorDemo
{
	public static void main(String[] args)
	{
		Goods g1 = new Goods(101, "Mouse", "Optical Mouse");
	}
}
```

Note

- this keyword can only be the first statement in Constructor
- A constructor can have either this or super keyword but not both
- this is a final variable in Java and you cannot assign value to this

"this" to refer current class instance variable

- If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity

Program N0. 80

this keyword to refer current class instance variable

```
class Goods
{
	int id;
	String name;
	String description;

	public Goods()
	{
		System.out.println("\n\t Goods Created");
	}
	public Goods(int id, String name)
	{
		this();
		this.id = id;
		this.name = name;
		System.out.println("\n\t id = " + id);
		System.out.println("\n\t Name = " + name);
	}
	public Goods(int id, String n, String des)
	{
```

```
            this(id, n);
            description = des;
            System.out.println("\n\t Description = " +
             description);
      }
}

public class ThisInstanceVar_80
{
      public static void main(String[] args)
      {

      }
}
```

"this" passed as an argument in the method

- The **this** keyword can also be passed as an argument in the method. It is mainly used in the event handling

Program N0. 81

"this" passed as an argument in the method

This keyword with Method

- this keyword can also be used inside methods to call another method from same class

Program N0. 82

"this" keyword with method

```
class Demo
{
      private void method1()
      {
            System.out.println("\n\t Method is Invoked");
      }

      public void method2()
      {
            this.method1();
      }
```

```
}

public class ThisMethod_82
{
	public static void main(String[] args)
	{
		Demo d = new Demo();
		d.method2();
	}
}
```

String

A Java string is a series of characters gathered together, like the word "Hello", or the phrase "practice makes perfect". Create a string in the code by writing its chars out between double quotes.

- Strings are a sequence of characters
- In the Java, strings are objects
- The Java platform provides the String class to create and manipulate strings
- Once a String object is created it cannot be changed. Stings are Immutable.
- There are various ways of creating String object
  - Direct method
  - Using String class constructors
  - Using toString() method defined in java.lang.Object

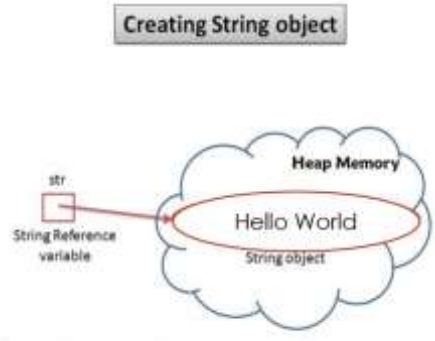
Creating Strings

- The most direct way to create a string is to write:

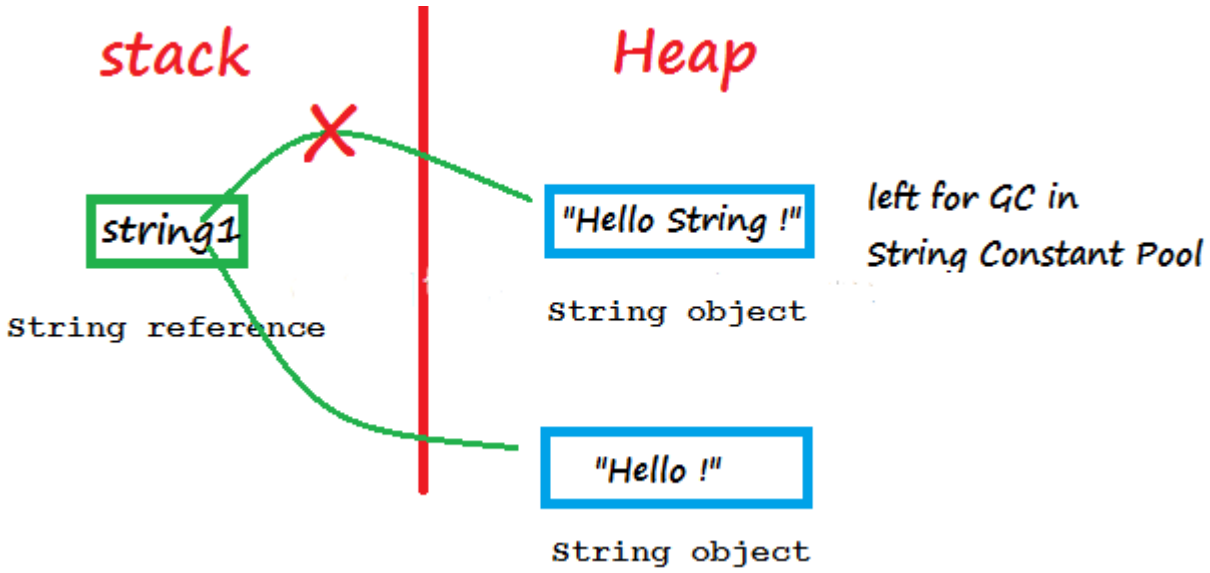
```
String str = "Hello World";
```

- Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

```
String str = "abc";
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```



- As with any other object, you can create String objects by using the new keyword and a constructor.
- The String class has eleven constructors that allow you to provide the initial value of the string using different sources, such as an array of characters.



Program N0. 83

String Demo

```
public class StringDemo_83
{
	public static void main(String[] args)
	{
		String str1 = "Hello World";
		System.out.println("\n\t str1 = " + str1);

		str1 = "Hello!";
		System.out.println("\n\t str1 = " + str1);
	}
}
```

Note:

The String class is immutable; so that once it is created a String object cannot be changed.

If there is a necessity to make a lot of modifications to Strings of characters, then you should use String Buffer & String Builder Classes

String Length

- Methods used to obtain information about an object are known as accessor methods.
- One accessor method that you can use with strings is the length() method, which returns the number of characters contained in the string object.

Program N0. 84

length() method

```
public class StringDemo_84
{
	public static void main(String[] args)
	{
		String str1 = "Hello World";
		System.out.println("\n\t str1 = " + str1);
		int len = str1.length();
		System.out.println("\n\t Length of str1 = " + len);

	}
}
```

Concatenating Strings

- The String class includes a method for concatenating two strings:

```
string1.concat(string2);
```

- This returns a new string that is string1 with string2 added to it at the end. You can also use the concat() method with string literals.

Program N0. 85

Concat() method

```
public class StringDemo_85
{
	public static void main(String[] args)

		String fname= "Vishal";
		String lname= "Shah";
		fname = fname.concat(" ");
		String fullName = fname.concat(lname);

		System.out.println("\n\t fname = " + fname);
		System.out.println("\n\t lname = " + lname);
		System.out.println("\n\t Fullname = " + fullName);

	}
}
```

String class Methods

charAt()

- public
  - Returns the character at the specified index. An index ranges from 0 to length() - 1.
  - The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing

```
char ch;
ch = "abc".charAt(1); // ch = "b"
```

getChars()

- Copies characters from this string into the destination character array
- public void **getChars**(int srcBegin, int srcEnd, char[] dst, int dstBegin)
  - **srcBegin** - index of the first character in the string to copy.
  - **srcEnd** - index after the last character in the string to copy.
  - **dst** - the destination array.
  - **dstBegin** - the start offset in the destination array

Program N0. 86

charAt()

getChars()

```
public class StringDemo_86
{
	public static void main(String[] args)
	{
		String src = "This is Source";
		char[] chararr = {'a','b',
		 'c','d','e','f','g','h','i','j','k'};

		src.getChars(0,3,chararr,5);
		String dest = new String(chararr);

		System.out.println("src = " + src);
		System.out.println("des = " + dest);

	}
}
```

compareTo()

- Compares two strings lexicographically
  - The result is a negative integer if this String object lexicographically precedes the argument string.
  - The result is a positive integer if this String object lexicographically follows the argument string
  - The result is zero if the strings are equal

- compareTo returns 0 exactly when the equals(Object) method would return true
- public int compareTo(String anotherString)
- public int compareToIgnoreCase(String str)

equals()

```
public boolean equals(Object anObject)
```

- Compares the invoking string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as the invoking object.

equalsIgnoreCase()

- Compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

```
public boolean equalsIgnoreCase(String anotherString)
```

Program N0. 87

compareTo()
Equals()
equalsIgnoreCase()

```
public class StringDemo_87
{
	public static void main(String[] args)
	{
		String str1 = "Abc";
		String str2 = "abc";

		int result = str1.compareTo(str2);

		System.out.println("str1 = " + str1);
		System.out.println("str2 = " + str2);

		System.out.println("Result = " + result);

		if(str1.equalsIgnoreCase(str2))
		   System.out.println("String are Same");
		 else
		     System.out.println("Strings are not same");

	}
```

```
}
```

startsWith()

```
public boolean startsWith(String prefix)
```

- Tests if this string starts with the specified prefix.

endsWith()

```
public boolean endsWith(String suffix)
```

- Tests if this string ends with the specified suffix.

Program N0. 88

startsWith()
endsWith()

```
public class StringDemo_88
{
	public static void main(String[] args)
	{
		String mob = "985081h100";

		if(mob.length()==10)
		{
			if(mob.startsWith("7") || mob.startsWith("8") ||
			   mob.startsWith("9"))
			{
			    System.out.println("Valid Mobile");
		    }
		    else
		    {
				System.out.println("Invalid");
			}
		}
		else
		{
			System.out.println("Invalid Mobile No.");
		}
	}
}
```

indexOf

- Searches for the first occurrence of a character or substring. Returns -1 if the character does not occur

```
public int indexOf(int ch)
```

- Returns the index within this string of the first occurrence of the specified character.

```
public int indexOf(String str)
```

- Returns the index within this string of the first occurrence of the specified substring.

```
public int indexOf(int ch, int fromIndex)
```

- Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

```
public int indexOf(String str, int fromIndex)
```

- Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

lastIndexOf()

- Searches for the last occurrence of a character or substring. The methods are similar to indexOf().

Program N0. 89

indexOf()

replace()

```
public String replace(char oldChar, char newChar)
```

- Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

trim()

```
public String trim()
```

- Returns a copy of the string, with leading and trailing whitespace omitted.
- **toLowerCase():** Converts all of the characters in a String to lower case.
- **toUpperCase():** Converts all of the characters in this String to upper case.

Program N0. 90

replace() **Trim()** **toLowerCase()** **toUpperCase()**

StringBuilder class

- The StringBuilder class is used to create mutable (modifiable) string.

- The StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK1.5

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

Commonly used methods of StringBuilder class

`public StringBuilder append(String s)`
- 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)`
- 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)`
- Used to replace the string from specified startIndex and endIndex.

`public StringBuilder delete(int startIndex, int endIndex)`
- Used to delete the string from specified startIndex and endIndex.

`public StringBuilder reverse()`
- Used to reverse the string.

`public int capacity()`
- Used to return the current capacity.

`public char charAt(int index)`
- used to return the character at the specified position.

`public int length()`
- used to return the length of the string i.e. total number of characters.

`public String substring(int beginIndex)`
- used to return the substring from the specified beginIndex.

`public String substring(int beginIndex, int endIndex)`
- used to return the substring from the specified beginIndex and endIndex

Program N0. 91

StringBuilder Class methods : Append() Insert() Replace() Delete()

```
public class StringBuilderDemo_91
{
	public static void main(String[] args)
	{
		StringBuilder sb = new StringBuilder("abcd");
```

```
            System.out.println("\n\t sb = " + sb);

            sb = sb.append("newstring");
            System.out.println("\n\t After Append: sb = "+ sb);

            sb = sb.insert(1, "www");
            System.out.println("\n\t After insert : sb = " + sb);

            sb = sb.replace(1, 4, "http");
            System.out.println("\n\t After REplace :sb = " + sb);

            sb = sb.delete(1,5);
            System.out.println("\n\t After Delete : sb = " + sb);
      }
}
```

Program N0. 92

StringBuilder Methods : Reverse() Capacity() Substring()

```
public class StringBuilderDemo_92
{
      public static void main(String[] args)
      {
            StringBuilder sb = new StringBuilder("abcdefghijk");
            System.out.println("\n\t sb = " + sb);

            sb = sb.reverse();
            System.out.println("\n\t Reverse sb = " + sb);

            System.out.println("\n\t Capacity = " +
            sb.capacity());

            System.out.println("\n\t Substring : " +
             sb.substring(0,4));
      }
}
```

StringBuffer Class

- The Java StringBuffer class is just like StringBuilder. The primary difference is that StringBuffer is synchronized. In other words, multiple threads can safely process StringBuffer objects.
- StringBuffer is slower than StringBuilder. You should use the StringBuffer class for string objects that will be changed by multiple threads of execution.

Immutable class

- There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable.
- We can also create immutable class by creating final class that have final data members

Program N0. 93

Immutable class : Employee

```
final class Employee
{
	final String panCardNo;

	public Employee(String no)
	{
		panCardNo = no;
	}

	public String getEmpPanNo()
	{
		return panCardNo;
	}
}

public class ImmutableClassDemo_93
{
	public static void main(String[] args)
	{
		Employee emp1 = new Employee("BLZPS8025A");
		System.out.println("Pan = " + emp1.getEmpPanNo());
	}
}
```

- The instance variable of the class is final i.e. we cannot change the value of it after creating an object.
- The class is final so we cannot create the subclass.

- There are no setter methods i.e. we have no option to change the value of the instance variable.

toString() method

- The toString() method returns the string representation of the object
- If you print any object, java compiler internally invokes the toString() method on the object.
- So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation
- By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code

Program N0. 94

Override toString() method

```
class Student
{
	int roll;
	String name;
	int marks;

	public Student()
	{
		roll = 101;
		name = "Bhartesh";
		marks = 45;
	}

	public String toString()
	{
		return roll + "\t" + name + "\t" + marks;
	}
}

public class ToStringMethod_94
{
	public static void main(String[] args)
	{
		Student s1 = new Student();
		System.out.println("\n\t " + s1);
	}
}
```

Java Regex

- The Java Regex or Regular Expression is an API to define pattern for searching or manipulating strings
- It is used to define constraint on strings such as password and email validation.

It provides following classes and interface for regular expressions. The Matcher and Pattern classes are widely used in java regular expression.

- MatchResult interface
- Matcher class
- Pattern class
- PatternSyntaxException class

Pattern class

It is the compiled version of a regular expression. It is used to define a pattern for the regex engine.

No.	Method	Description
1	static Pattern compile(String regex)	compiles the given regex and return the instance of pattern.
2	Matcher matcher(CharSequence input)	creates a matcher that matches the given input with pattern.
3	static boolean matches(String regex, CharSequence input)	It works as the combination of compile and matcher methods. It compiles the regular expression and matches the given input with the pattern.
4	String[] split(CharSequence input)	splits the given input string around matches of given pattern.
5	String pattern()	returns the regex pattern.

Matcher class

It implements MatchResult interface. It is a regex engine i.e. used to perform match operations on a character sequence.

No.	Method	Description
1	boolean matches()	test whether the regular expression matches the pattern.
2	boolean find()	finds the next expression that matches the pattern.
3	boolean find(int start)	finds the next expression that matches the pattern from the given start number

Program N0. 95

RegexDemo_95

```
import java.util.Scanner;
import java.util.regex.*;

public class RegexDemo_95
{
	public static void main(String[] args)
	{
		Scanner scan = new Scanner(System.in);
		System.out.println(Pattern.matches("[0-9]", "23434"));
	 }
}
```

Program N0. 96

RegexDemo_96

```
import java.util.regex.*;
public class RegexExample3
{
	public static void main(String args[])
	{
		boolean isValidMob;
		isValidMob = Pattern.matches("[0-9]{1,10}", "9ds0810100");
		System.out.println(isValidMob);
	}
}
```

Regex Character class

No.	Character Class	Description
1	[abc]	a, b, or c (simple class)
2	[^abc]	Any character except a, b, or c (negation)
3	[a-zA-Z]	a through z or A through Z, inclusive (range)
4	[0-9]	0 through 9