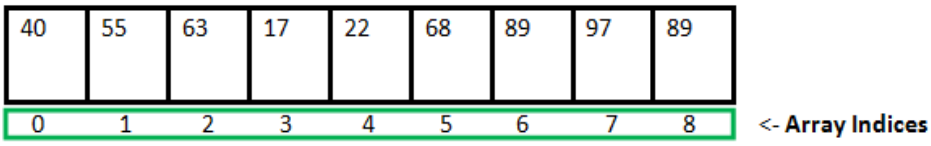
MCA FIRST SEMESTER- 2021
MCA20103-OBJECT ORIENTED PROGRAMMING USING JAVA
UNIT-2

1. Arrays in Java:

An array is a group of like-typed variables that are referred to by a common name. Arrays in Java work differently than they do in C/C++. Following are some important point about Java arrays.

- In Java all arrays are dynamically allocated.
- Since arrays are objects in Java, we can find their length using member length. This is different from C/C++ where we find length using sizeof.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered and each have an index beginning from 0.
- Java array can be also be used as a static field, a local variable or a method parameter.
- The size of an array must be specified by an int value and not long or short.

Array can contain primitives data types as well as objects of a class depending on the definition of array. In case of primitive's data types, the actual values are stored in contiguous memory locations.



Array Length = 9
First Index = 0
Last Index = 8

Creating, Initializing, and Accessing an Array

Arrays are classified into three types:

1. One Dimensional arrays
2. Two Dimensional arrays

1. One Dimensional array:

A variable name having list of items using only one subscript is called One Dimensional arrays. Arrays must be declared and created in the computer memory before they are used. Creation of an array involves three types

a) Declaring an array
b) Creating memory location
c) Putting values into the memory locations/Initialization of arrays

- **Declaration of an array:**

In java arrays can be declared in two ways

Syntax 1: Type ArrayName [ ];
Syntax 2: Type [ ] ArrayName;
Eg: int num[ ];
float[ ]avg;

- **Creating memory locations:**

After declaring an array we need to allocate memory to it. Java allows us to create arrays using new operator

Syntax: ArrayName=new type [size];

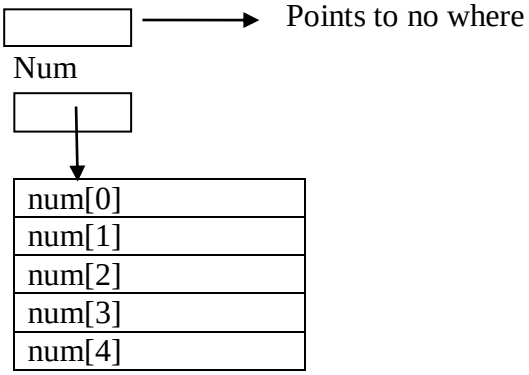
Eg: num =new int [5];

The variable num refers to an array of 5 integers. The conceptual view for the above example is

num[0]
num[1]
num[2]
num[3]
num[4]

We can combine declaration and creation into one step

Eg: int num [ ]=new int[5];

Statement	Result
int num[]	Num
num=new int[5];	

- **Initialization of arrays:** The final step is to insert values into the array. This process is called initialization. This is done using an array subscript or index

Eg: num [2]=35;

Array index starts from 0 and hence with "size-1".we can initialize arrays when they are declared

Syntax: Type ArrayName [ ]={list of values};

Eg: int num [ ]={35, 40,25,19};

- **Array length:** All arrays store the allocated sizes in a variable named " length".we can apply the length of the array num using "num.length"

Eg: int l=num.length;

2. Two Dimensional arrays:

A variable name having list of items using two subscripts is called "two dimensional arrays". Two dimensional arrays are in the form of rows and columns. The declaration and creation of two dimensional arrays is follows:

Syntax 1: type ArrayName [ ] [ ];
ArrayName=newtype[size1][size2];

Syntax2: type ArrayName[ ][ ]=new type[size1][size2];

Here size 1 specifies number of rows and size2 specifies number of columns

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

The conceptual view for the above example is

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

This example creates an array that can store 12 integer numbers in 3 rows and 4 columns

- **Initialization:**

The initialization of two dimensional arrays is as follows:

Eg: int a[ ] [ ] ={{6, 9, 4},{3,4,8}};

Camas is required after each brace that closes of a row, except in the case of last row. We can refer to a value stored in the third row and the second column in the array as a[2][1]

2. Three Dimensional Arrays:

Three-dimensional array is the collection of two-dimensional arrays in Java programming language. Three-dimensional array is also called the multidimensional array.

- **Declaration of the three dimensional Array**

Syntax data_type[][][] arrayname;----1
or
data_type arrayname [][][] ;----2

we can choose any one method

Explanation of Three dimensional Array declaration

Syntax: int [][][]array;

- **Forming of three dimension array**

Syntax : data_type[][][]array_name=new data_type;

Example: int array[][][]=new int[2][3][3];

- ***How to initialize a 3d array in Java***
- **Method 1**

Syntax array_name[index_1][index_2][index_3]=value;
Example arr[0][0][0]=45; //initialize first elements of 3 d array
we can initialize every index, like this

- **Method 2**

```
Ex: int[][][] arr{
                {
                   {34,67,43},
                    {576,697,423},
                     {576,697,423}
                    },
                 {
                   {39,47,33},
                    {376,987,453},
                     {57,69,42}
                    },
```

Example program :

```
class threedarrayex{
public static void main(String args[])
{
int[][][]marks; //declaration of array
marks=new int[2][2][2];//initiation of array
//initiate elements
marks[0][0][0]=25;
marks[0][0][1]=15;
marks[0][1][0]=20;
marks[0][1][1]=05;
marks[1][0][0]=10;
marks[1][0][1]=13;
marks[1][1][0]=12;
marks[1][1][1]=30;
//Display elements from array
System.out.println(marks[0][0][0]);
System.out.println(marks[0][0][1]);
System.out.println(marks[0][1][0]);
System.out.println(marks[0][1][1]);
System.out.println(marks[1][0][0]);
System.out.println(marks[1][0][1]);
System.out.println(marks[1][1][0]);
System.out.println(marks[1][1][1]);
```

```
}
}
```

When the above code is compiled and executed, it will produce the following results

```
25
15
20
5
10
13
12
30
```

3. STRINGS:

String is a sequence of characters, for e.g. "Hello" is a string of 5 characters. In java, string is an immutable object which means it is constant and cannot be changed once it has been created.

1. Creating a String:-

There are two ways to create string in Java:

- ***String literal***
  String s = "GeeksforGeeks";
- **Using *new* keyword**
  String s = new String ("GeeksforGeeks");

- **String literal**

In java, Strings can be created like this: Assigning a String literal to a String instance:

```
String str1 = "Welcome";
String str2 = "Welcome";
```

What if we want to have two different object with the same string? For that we would need to create strings using **new keyword**.

- **Using New Keyword**

As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this:

```
String str1 = new String("Welcome");
String str2 = new String("Welcome");
```

In this case compiler would create two different object in memory having the same string**.**

2. String class methods:

String class provides number of methods to perform various operations on string objects the following of the most commonly used string methods.

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	String substring(int beginIndex)	returns substring for given begin index.
4	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
5	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
6	boolean equals(Object another)	checks the equality of string with the given object.
7	boolean isEmpty()	checks if string is empty.
8	String toLowerCase()	returns a string in lowercase.
9	String toUpperCase()	returns a string in uppercase.
10	String trim()	removes beginning and ending spaces of this string.

Ex:

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

```
  String s="Sachin";

  System.out.println(s.toUpperCase());//SACHIN

  System.out.println(s.toLowerCase());//sachin

  System.out.println(s);//Sachin(no change in original)

 }

}
```

Compile by: javac Testmethodofstringclass.java
Run by: java Testmethodofstringclass

```
SACHIN
sachin
Sachin
```

3. Compare two Strings in Java:

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

1. **Using user-defined function :** Define a function to compare values with following conditions :
    1. if (string1 > string2) it returns a **positive value**.
    2. if both the strings are equal lexicographically i.e.(string1 == string2) it returns **0**.
    3. if (string1 < string2) it returns a **negative value**.

2. **Using String.equals() :**In Java, string equals() method compares the two given strings based on the data/content of the string. If all the contents of both the strings are same then it returns true. If any character does not match, then it returns false.
   Syntax:
   str1.equals(str2);
   Here str1 and str2 both are the strings which are to be compared.

   Examples:
   Input 1: GeeksforGeeks
   Input 2: Practice
   Output: false

   Input 1: Geeks

Input 2: Geeks
Output: true

```
Input 1: geeks
Input 2: Geeks
Output: false
```

Ex Program:

```
// Java program to Compare two strings
// lexicographically
public class GFG {
    public static void main(String args[])
    {
        String string1 = new String("Geeksforgeeks");
        String string2 = new String("Practice");
        String string3 = new String("Geeks");
        String string4 = new String("Geeks");
        String string5 = new String("geeks");

        // Comparing for String 1 != String 2
        System.out.println("Comparing " + string1 + " and " + string2
                        + " : " + string1.equals(string2));

        // Comparing for String 3 = String 4
        System.out.println("Comparing " + string3 + " and " + string4
                        + " : " + string3.equals(string4));

        // Comparing for String 4 != String 5
        System.out.println("Comparing " + string4 + " and " + string5
                        + " : " + string4.equals(string5));

        // Comparing for String 1 != String 4
        System.out.println("Comparing " + string1 + " and " + string4
                        + " : " + string1.equals(string4));
    }
}
```

Output:

```
Comparing Geeksforgeeks and Practice : false
Comparing Geeks and Geeks : true
Comparing Geeks and geeks : false
Comparing Geeksforgeeks and Geeks : false
```

4. Immutable String in Java:

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.Once string object is created its data or state can't be changed but a new string object is created. Let's try to understand the immutability concept by the example given below:

Ex:
```
class Testimmutablestring
{
 public static void main(String args[])
{
 String s="Sachin";
 s.concat(" Tendulkar");//concat() method appends the string at the end
 System.out.println(s);//will print Sachin because strings are immutable objects
}
}
```

Test it Now

```
Output:Sachin
```

Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

Ex:
```
class Testimmutablestring1
{
  public static void main(String args[])
    {
        String s="Sachin";
        s=s.concat(" Tendulkar");
        System.out.println(s);
    }
}
```

Test it Now

```
Output:Sachin Tendulkar
```

5. String Buffer:

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

String class creates string objects of fixed length. String buffer creates string objects of flexible length that can be modified in terms of length and contents. We can insert characters and sub strings in the middle of the string or end of the string. String buffer class provides the following methods to perform different operations on string

Syntax: StringBuffer object=new StringBuffer();

Important methods of StringBuffer class:

1) StringBuffer 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
}
}
```

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

```
Ex: 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
}
}
```

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
Ex: class StringBufferExample3{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
Ex: class StringBufferExample4{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo  }  }
```

5) StringBuffer reverse() method

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

Ex:
```
class StringBufferExample5{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

6. Java String Builder Class:

Java StringBuilder class is used to create mutable string. The java string builder class is same as string buffer class except that it is non synchronized.

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 are 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.

Ex:
```
public class BuilderTest
  {
   public static void main(String[] args)
    {
       StringBuilder builder=new StringBuilder("hello");
        builder.append("java");
        System.out.println(builder);
    }
  }
```
Output: hellojava

Important methods of StringBuilder class

1) StringBuilder append() method

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

2) StringBuilder insert() method

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

3) StringBuilder replace() method

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

4) StringBuilder delete() method

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

5) StringBuilder reverse() method

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

7. Command Line Arguments

Sending arguments form the command line to the main method of the program while executing the program is called command line arguments. When we send command line arguments to the main() method

While interpreting the program, the JVM creates an array of strings with the arguments and send to main through the reference of array of strings we can access the barguments into the main. It helps to improve the flexibility in application development

Eg: >javac Demo.java

>java Demo hello I am the learner

```
class Demo
{
public static void main(String args[ ])
{
for(int i=0;i<args.lenth;i++)
System.out.print(args[i]+" ");
}
}
```

Output:

Hello
I
am
The
Learner

Individual elements of command line arguments are accessed from the main using the index of args[ ].

8. Procedure Oriented Programming (POP) :-

High level language such as C, COBOL,PASCAL etc is commonly known as POP languages. In procedure oriented programming the problem is viewed as sequence of things to be done, such as reading, calculating and printing. In procedural Oriented Programming the problem can be decomposes into small tasks knows as Functions. In Procedure Oriented Programming the primary focus is on function. The diagrammatic representation of multi-function programming is as follows

Advantages:

- Its relative simplicity, and ease of implementation of compilers and interpreters
- The ability to re-use the same code at different places in the program without copying it.
- An easier way to keep track of program flow.
- The ability to be strongly modular or structured.
- Needs only less memory.

Disadvantages:

- Data is exposed to whole program, so no security for data.
- Difficult to relate with real world objects.
- Difficult to create new data types reduces extensibility.
- Importance is given to the operation on data rather than the data.

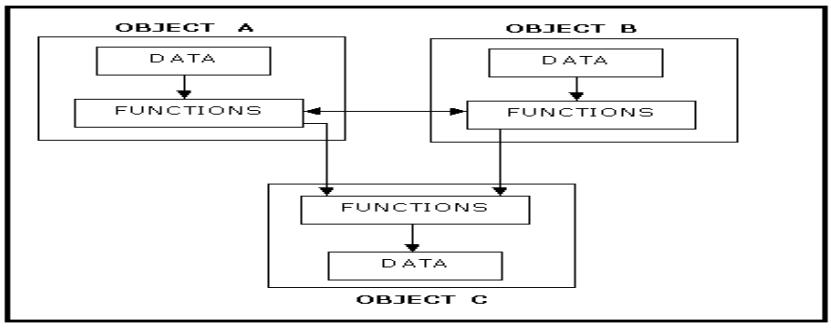
In POP global data may be accessed by all the functions. In a large program, it is very difficult to identify what data is used by which function. This is one of the biggest drawbacks in procedure-oriented programming. To solve the disadvantages encountered (find) in POP, developers develop a new concept Object Oriented Programming (OOPS).

9. Features of Object Oriented Programming:

Object-Oriented approach was introduced to overcome the drawbacks of Procedural approach. OOP treats data as an important element in the program development. Data cannot be moved freely around the system. It binds data more closely to the functions and protect from accidental modification from the outside functions.

OOPs decompose a problem into a number of entities called “Objects”. The Organization of data and functions in Object Oriented Programming as shown below.

Organization of OOPS



Basic concepts of OOPs (or) Basic Features of OOPs (or) Key concepts of OOPs (or) OOP Characteristics:-

Object oriented Programming is an approach that provides the way of modularizing programs by creating partition memory for both data and functions. The following are the basic elements of OOPs.

1. Objects:

- Objects are the basic run time entities in an object oriented programming.
- Memory will allocates only for objects not for classes.
- Objects are the variables of class.
- Each object contains data and functions. By using functions, we can manipulate data.

2. Classes:

- Classes are the basic building blocks of oops.
- Classes combined both data and function.
- Classes create new user defined data types.
- A class is a data type and object is a variable of class type.
- Once a class has been defined we can create number of objects belonging to that class.

3. Data abstraction:

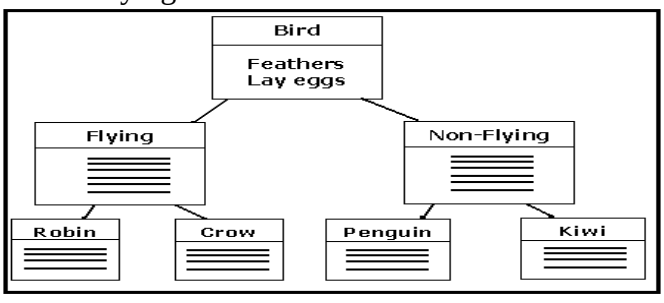
- Data abstraction means removing unnecessary data. i.e., representing the essential features without including background details.
- By the feature of data abstraction, it is possible to create user defined datatypes.

4. Data encapsulation:

- The process of combining both data and functions into a single unit is known as data encapsulation.
- Class is the example for data encapsulation. The Data is not accessible by outside class.
- By using encapsulation, we can accomplish data hiding.

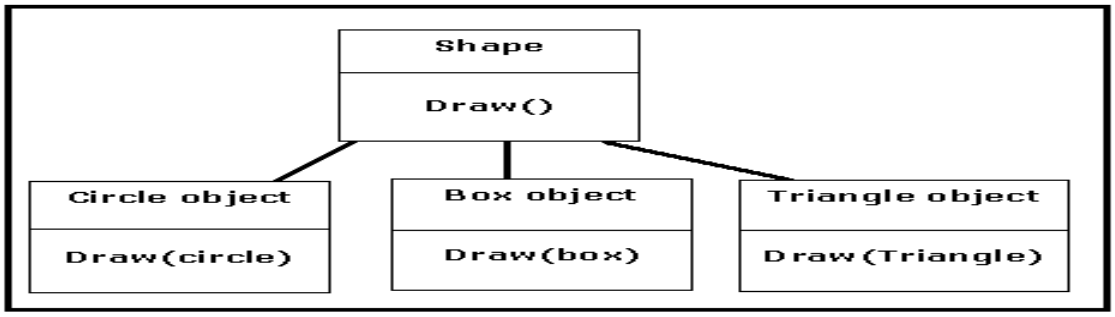
5. Inheritance:

- Inheritance is a process of creating a new class from existing class. The new class contains both the features new class and old class.
- Inheritance is very useful to add additional features to an existing class without modifying it.



6. Polymorphism: -

- Polymorphism means the ability to take more than one form with a single name.
- "Polymorphism" comes from the 'Greek' word. "Poly" means "many" and "morphism" means "forms". i.e., many forms.



7. Dynamic Binding:- Binding Refers to the linking of procedure called to the code to be executed in response to that call. Dynamic binding means that the code associated with the given procedure call is not known until runtime. It is achieved with virtual functions and inheritance in c++.

8. **Message Passing:-** A message for an object is a request for executing a procedure or a function. A message for an object will invoke a function that generates the desired results. Message passing involve specifying the name of object, name of the function and information to be send.

Ex:- **Emp.salary(sno);**

Here Emp is an object, salary is a message and sno is an information.

CLASSES AND OBJECTS:

Classes:

- **class:**

A class is a collection of related objects that share a common properties and methods. Classes are used to pack a group of data items and functions in java. The data items are called "fields" and functions are called "methods".

- **Defining a Class:**

classes are defined using the keyword "class"once a class is defined we can create any number of objects belong into that class. In java these variable called "instance of classes".

Syntax:

```
class classname
{
  Fields Declaration/variable Declaration
  Method Declaration
}
```

- **Fields Declaration/variable Declaration:** The variables which are declared inside a class are called "fields". These variables are also called instance variables they are created when ever an object of the class is instantiated.

Eg:

```
class Rectangle
```

```
{
int length;
int width;
}
```

The class rectangle contains two integer type instance variables no memory space is reserved for these instance variables

- **Methods Declaration:** Methods are necessary for manipulating the data contain that class. Methods are declared and defined inside the body of the class immediately after the declaration of instance of variable

syntax: return_type method_name(parameters list)

```
{
        Method body
}
```

Method declarations have 4 basic parts

1. The name of the method
2. The type of the value that the method returns
3. A list of parameters
4. The body of the method

Eg:

```
class Rectangle
{
 int length,width;
 void getData(int x,int y)
   {
   length=x;
   width=y;
  }
}
```

Objects:

- **Objects:**

An object is a block of memory that contains space to store all the instance variable

- **Creating objects:**

- Creating an object is also referred as instantiating an object
- In java objects are created using the operator new
- The new operator creates an object of the specified class and returns a reference to the object

syntax 1:

```
classname obj;
obj= new className();
```

syntax 2:

```
classname obj=new classname();
```

Eg: Rectangle r;
r= new Rectangle();

❖ **Accessing class members:**

We cannot access the instance variables and methods directly from outside the class. To access them from outside the class we must use the concerned object and the dot (.) operator

syntax:

objectname.variable=value;
objectname.methodName(parameters list);

Here object name is the name of the object method name is the name of the method that we wish to call

Eg:

```
class Rectangle
     {
    int length,width;
    void getData(int x, int y)
     {
      length=x;
      width=y;
     }

    int Area()
   {
   return length*width;
  }
 }
class RectArea
{
public static void main(String args[])
{
int area1,area2;
Rectangle r1=new Rectangle();
Rectangle r2=new Rectangle();
r1.length=12;
r1.width=10;
area1 =r1.length*r1.width;
r2.getData(20,13);
area2=r2.Area();
System.out.println("Area1="+area1);
System.out.println("Area2="+area2);
}
}
```

Output:

Area 1=120
Area 2=260

Variables in java:

There are three types of variables in java .

1.local variable:

The variable which is declared inside a method is called local variable.

2.instance variables:

A variable which is declared inside a class but outside a method is called instance variable. But it is not declared as static

3.static variables:

A variable which is declared as static is called static variable. It cannot be local.

Example to understand the types of variables in java

```
class A
{
int data=50;
static int m=100;
void method()
{
int n=90;
}
}
```

❖ **Initializing Instance variables:-**

- Instance variables are declared in class but outside a method
- Instance variables are created when an object is created with the use of the keyword new and destroyed when the object is destroyed
- Instance variables can be declared in a class level before or after use
- Access modifiers can be given for instance variables
- The instance variables are visible for all methods and constructors and block in the class. Normally it is recommended to make these variables private .however visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However with in static methods they should be called using the fully qualified name, objectreference.variablename.

Example for initializing instance variables:

```
public class Employee
{
public String name ;
```

```
private double salary;
public Employee(String empName)
{
name =empName;
}
public void setSalary(double empSal)
{
salary=empSal;
}
public void printEmp()
{
System.out.println("name:"+name);
System.out.println("salary:"+salary);
}
public static void main(String args[ ])
{
Employee empOne=new Employee("venkatesh");
empOne.setSalary(100000);
empOne.printEmp();
}
}
```

Output:

```
name:venkatesh
salary:100000
```

Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object.
Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.
There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

- **Rules for creating Java constructor**

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name

2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

❖ **Types of Java constructors**

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

1. Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

<class_name>(){}

1. Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1
{
Bike1(){System.out.println("Bike is created");}
public static void main(String args[]){
Bike1 b=new Bike1();
}
}
```

Output:

```
Bike is created
```

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

2. Example of default constructor that displays the default values

```
class Student3{
int id;
String name;
void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
Student3 s1=new Student3();
Student3 s2=new Student3();
s1.display();
s2.display();
}
}
```

Output:

```
0 null
0 null
```

In the above class, we are not creating any constructor so compiler provides a default constructor. Here 0 and null values are provided by default constructor.

2. Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.The parameterized constructor is used to provide different values to distinct objects. However, we can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

Ex:
```
class Student4{
  int id;
  String name;
  Student4(int i,String n){
  id = i;
  name = n;
  }
  void display(){System.out.println(id+" "+name);}
  public static void main(String args[]){
  Student4 s1 = new Student4(111,"Karan");
  Student4 s2 = new Student4(222,"Aryan");
  s1.display();
  s2.display();
 } }
```

Output:
```
111 Karan
222 Aryan
```

❖ Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

Ex:
```
class Student5{
  int id;
  String name;
  int age;
  Student5(int i,String n){
  id = i;
  name = n;
  }
  Student5(int i,String n,int a){
```

```
   id = i;
    name = n;
    age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan 0
222 Aryan 25
```

Access specifiers (or) visibility controls (or) Access modes

Sometimes it is necessary to restrict the access to some variables and methods from outside the class. In java it is achieved by applying access controls to instance variables and methods. These access controls are also called visibility controls. Java supports 3 different types of visibility controls

- 1.Public 2. Private 3. Protected

❖ **public access:** The members that are declared as public are visible and accessible anywhere. The scope of the public members is global.

Eg:
```
public int num;
public void sum()
 {
     …….
     …..…..
 }
```

❖ **private access:** The members that are declared as private are not accessible from outside the class they are accessible only within their own class by the methods of that class. They cannot accessible in sub class.

Eg :
```
private  int num;
private void sum()
{
…….
……..
}
```

❖ **protected access:** The members that are declared as protected are visible to all the classes and sub classes in the same package and also subclasses in other packages. The non sub class in other packages cannot access the protected members.

Eg:
```
protected int num;
protected void sum()
{……..}
```

Ex1:
```
class A
{
  public void display()
   {
      System.out.println("SoftwareTestingHelp!!");
   }
}
class Main
 {
     public static void main(String args[])
    {
        A obj = new A ();
         obj.display();
    }
 }
```
Output:
SoftwareTestingHelp!!

Ex2: **The below program demonstrates the usage of the Protected Access modifier in Java.**

```
class A
{
  protected void display()
   {
      System.out.println("SoftwareTestingHelp");
   }
}

class B extends A {}
class C extends B {}

class Main{
    public static void main(String args[])
  {
     B obj = new B();     //create object of class B
     obj.display();       //access class A protected method using obj
     C cObj = new C();    //create object of class C
     cObj.display ();     //access class A protected method using cObj
  }
}
```

Output: **SoftwareTestingHelp**

SoftwareTestingHelp

Ex3: The following Java program demonstrates the use of getter and setter methods for private variables in Java.

```
class DataClass {
   private String strname;

// getter method
   public String getName() {
      return this.strname;
   }
   // setter method
   public void setName(String name) {
      this.strname= name;
   }
}
public class Main {
   public static void main(String[] main){
      DataClass d = new DataClass();

 // access the private variable using the getter and setter
      d.setName("Java Programming");
      System.out.println(d.getName());
   }
}
```

Output:

```
 Java Programming
```