Unit – 1: INTRODUCTION TO OOP AND JAVA		
Chapter No.	**Topic**	**Page No.**
1.1	Overview of OOP	1
1.2	Features/Characteristics of OOP	4
1.3	Java Buzzwords	8
1.4	Overview of Java	11
	1.4.1: Basic Java Terminologies	12
	1.4.2: Java Source File Structure	14
1.5	Java Data Types	19
1.6	Java Variables	21
1.7	Arrays	24
1.8	Operators	32
1.9	Control Flow Statements	42
1.10	Defining Classes and Objects	57
1.11	Methods	61
1.12	Constructors	63
	Types of Constructor	63
	'this' Keyword	68
	Constructor Overloading	70
	Constructor Chaining	71
1.13	Access Specifiers	73
1.14	Static Members	75
1.15	JavaDoc Comments	79
1.16	Additional Topics	86
	1.16.1: Java Comments	86
	1.16.2: Java Constants	87
	1.16.3: Java Identifiers	87
	1.16.4: Java Keywords	87
	1.16.5: Type Conversions and Casting	88
	1.16.6: Garbage Collection	90
	1.16.7: Using Command Line Arguments	92

UNIT 1 INTORDUCTION TO OOP AND JAVA

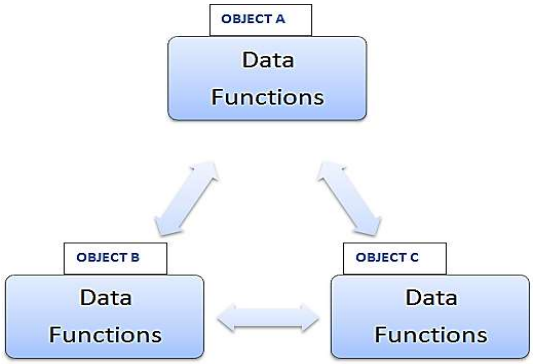
Overview of OOP – Object Oriented Programming Paradigms – Features of Object Oriented Programming – Java Buzzwords – Overview of Java – Data Types, Variables and Arrays – Operators – Control Statements – Programming Structures in Java – Defining Classes in Java – Constructors – Methods – Access Specifiers – Static Members – JavaDoc Comments.

1.1: Overview of OOP

OBJECT ORIENTED PROGRAMMING (OOP):

Object-Oriented Programming System (OOPs) is a programming paradigm based onthe concept of —objects that contain data and methods, instead of just functions and procedures.

- ✓ The primary **purpose** of object-oriented programming is to increase the flexibility and maintainability of programs.
- ✓ Object oriented programming brings together data and its behavior (methods) into a single entity (object) which makes it easier to understand how a program works.



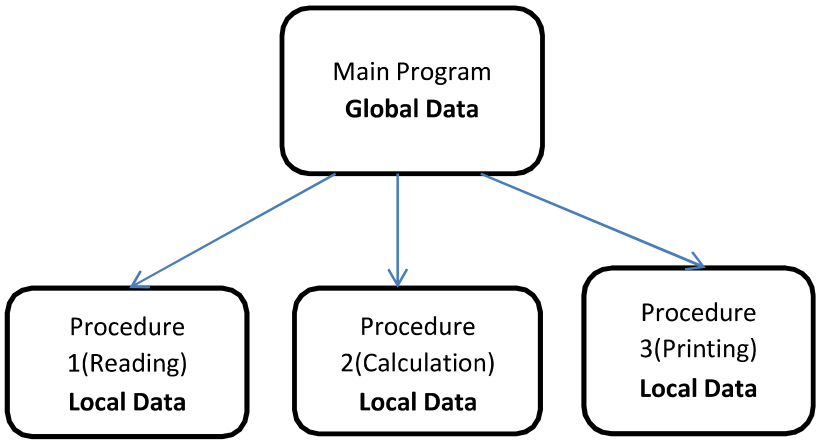
- **Features / advantages of Object Oriented Programming :-**
  1. It emphasis in own data rather than procedure.
  2. It is based on the principles of inheritance, polymorphism, encapsulation and dataabstraction.
  3. Programs are divided into objects.
  4. Data and the functions are wrapped into a single unit called class so that data ishidden and is safe from accidental alternation.
  5. Objects communicate with each other through functions.
  6. New data and functions can be easily added whenever necessary.
  7. Employs bottom-up approach in program design.

- **PROCEDURE-ORIENTED PROGRAMMING [POP]:**

Procedure-Oriented Programming is a conventional programming which consists of writing a list of instructions for the computer to follow and organizing these instructions into groups known as Functions (or) Procedures (or) subroutines (or)Modules.

Example: A program may involve the following operations:

- ✓ Collecting data from user (Reading)
- ✓ Calculations on collected data (Calculation)
- ✓ Displaying the result to the user (Printing)



Characteristics of Procedural oriented programming:-

1. It focuses on process rather than data.
2. It takes a problem as a sequence of things to be done such as reading, calculating and printing. Hence, a number of functions are written to solve a problem.
3. A program is divided into a number of functions and each function has clearly defined purpose.
4. Most of the functions share global data.
5. Data moves openly around the system from function to function.
6. Employs top-down approach in program design.

Drawback of POP

- ➢ Procedural languages are difficult to relate with the real world objects.
- ➢ Procedural codes are very difficult to maintain, if the code grows larger.
- ➢ Procedural languages do not have automatic memory management as like in Java. Hence, it makes the programmer to concern more about the memory management of the program.
- ➢ The data, which is used in procedural languages, are exposed to the whole

program.So, there is no security for the data.

- Examples of Procedural languages :
  - BASIC
  - C
  - Pascal
  - FORTRAN

Difference between POP and OOP:

	Procedure Oriented Programming	Object Oriented Programming
Divided Into	In POP, program is divided into smallparts called **functions**.	In OOP, program is divided into partscalled **objects**.
Importance	In POP, Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
Access Specifiers	POP does not have any accessspecifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely fromfunction to function in the system.	In OOP, objects can move and communicate with each other throughmember functions.
Expansion	To add new data and function in POPis not so easy.	OOP provides an easy way to add newdata and function.
Data Access	In POP, Most function uses Globaldata for sharing that can be accessed freely from function to function in the system.	In OOP, data cannot move easily from function to function, it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper wayfor hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Examples of POP are: C,VB, FORTRAN, and Pascal.	Examples of OOP are: C++, JAVA, VB.NET, C#.NET.

1.2: FEATURES / CHARACHTERISTICS OF OBJECT ORIENTED PROGRAMMING CONCEPTS

OOPs simplify the software development and maintenance by providing some concepts:

1. Class - Blue print of Object
2. Object - Instance of class
3. Encapsulation - Protecting our data
4. Polymorphism - Different behaviors at different instances
5. Abstraction - Hiding irrelevant data
6. Inheritance - An object acquiring the property of another object

1. Class:

A class is a collection of similar objects and it contains data and methods that operate on that data. In other words **- Class is a blueprint or template for a set of objects that share a common structure and a common behavior.** It is a logical entity.

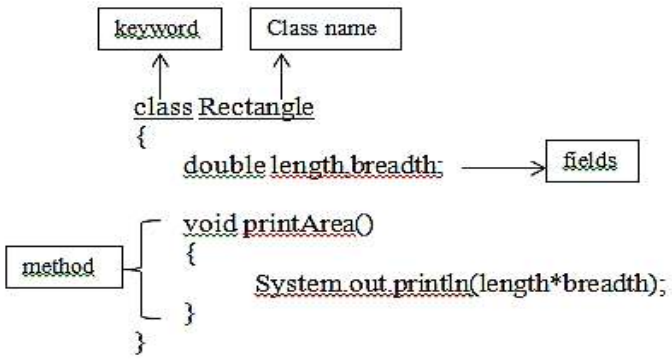
A class in Java can contain:

- **fields**
- **methods**
- **constructors**
- **blocks**
- **nested class and interface**

Syntax to declare a class:

Example:

```
class <class_name>
{
   field;
   method;
}
```



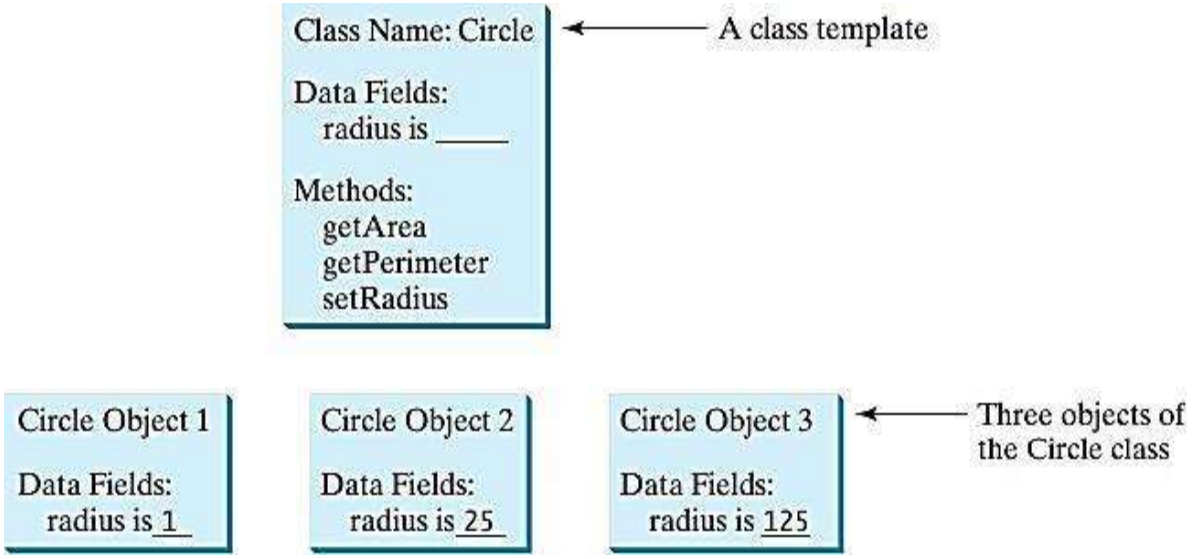
2. Object:

Any entity that has state and behavior is known as an object. **Object is an instance of a class.**

- ✓ For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.
- ✓ The object of a class can be created by using the **new** keyword in Java Programminglanguage.

```
class_name object_name = new class_name;
                (or)
class_name object_name;
object_name = new class_name();
```

Syntax to create Object in Java:



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 an unique ID 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.

Difference between Object and Class

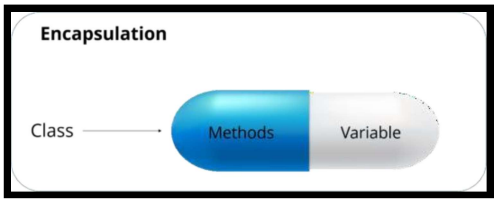
S.No.	Object	Class
1)	Object is an **instance** of a class.	Class is a **blueprint or template** from whichobjects are created.
2)	Object is a **real world entity** such aspen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a **group of similar objects**.
3)	Object is a **physical** entity.	Class is a **logical** entity.
4)	Object is created through **new keyword** mainly e.g. Student s1=new Student();	Class is declared using **class keyword** e.g.class Student{}
5)	Object is created **many times** as perrequirement.	Class is declared **once**.

6)	Object **allocates memory when it iscreated.**	Class **doesn't allocated memory when it iscreated.**
7)	There are **many ways to create object** in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only **one way to define class** in java using class keyword.

3. **Encapsulation:**

Wrapping of data and method together into a single unit is known as Encapsulation.

For example: capsule, it is wrapped with different medicines.



- ✓ **In OOP, data and methods operating on that data are combined together to form a single unit, this is referred to as a Class.**
- ✓ Encapsulation is the mechanism that binds together code and the data it manipulates and keeps both safe from outside interference and misuse.
- ✓ **The insulation of the data from direct access by the program is called —data hiding.** Since the data stored in an object cannot be accessed directly, the data is safe i.e.,the data is unknown to other methods and objects.

4. **Polymorphism:**

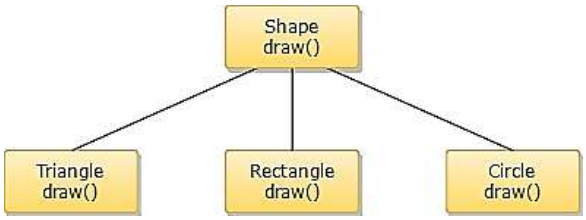
- ✓ Polymorphism is a concept by which we can perform a single action by different ways. It is the ability of an object to take more than one form.
- ✓ The word "poly" means many and "morphs" means forms. So polymorphism meansmany forms.
- ✓ An operation may exhibit different behaviors in different instances. The behavior depends on the data types used in the operation.
- ✓ For Example:- Suppose if you are in a classroom that time you behave like a student, when you are in the market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person present in different-different behaviors.

- Two types of polymorphism:

1. **Compile time polymorphism / Method Overloading: -** In this method, object is bound to the function call at the compile time itself.
2. **Runtime polymorphism / Method Overriding: -** In this method, object is bound to the function call only at the run time.

- **<u>In java, we use method overloading and method overriding to achievepolymorphism.</u>**
- **Example:**

1. draw(int x, int y, int z)
2. draw(int l, int b)
3. draw(int r)



5. <u>Abstraction:</u>

- ✓ Abstraction refers to the act of representing essential features without including the background details or explanations**. i.e., *Abstraction* means hiding lower-level details and exposing only the essential and relevant details to the users.**
- ✓ For Example: - Consider an ATM Machine; All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement…etc. but we can't know internal details about ATM.
- ✓ Abstraction provides advantage of code reuse.
- ✓ Abstraction enables program open for extension.
- ✓ **In java, abstract classes and interfaces are used to achieve Abstraction.**

6. <u>Inheritance:</u>

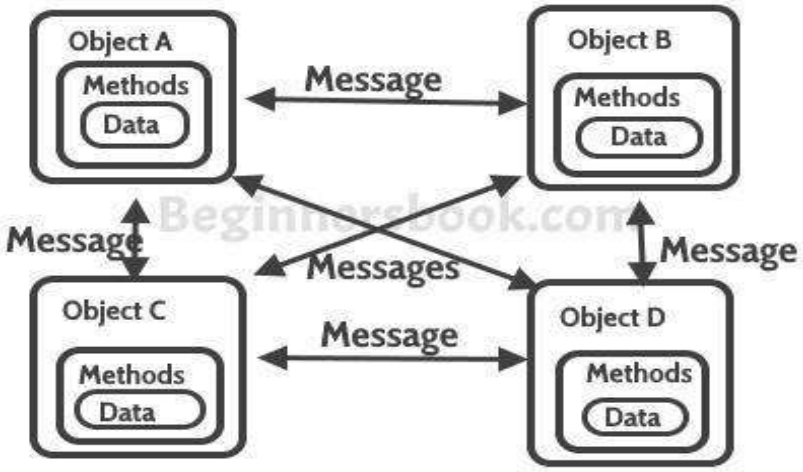
- ✓ **Inheritance in java** is a mechanism in which one object acquires all the propertiesand behaviors of another object.
- ✓ The idea behind inheritance in java is that we can create new classes that are built upon existing classes. When we inherit from an existing class, we can reuse methods and fields of parent class, and we can add new methods and fields also.
- ✓ Inheritance represents the **IS-A relationship**, also known **as *parent-child* relationship.**
- ✓ For example:- In a child and parent relationship, all the properties of a father areinherited by his son.
- ✓ **Syntax of Java Inheritance**

```
class Subclass-name extends Superclass-name
{
//methods and fields
}
```

7. **Message Passing:**

Message Communication:

- ✓ Objects interact and communicate with each other by sending ***messages*** to eachother. This information is passed along with the message as parameters.



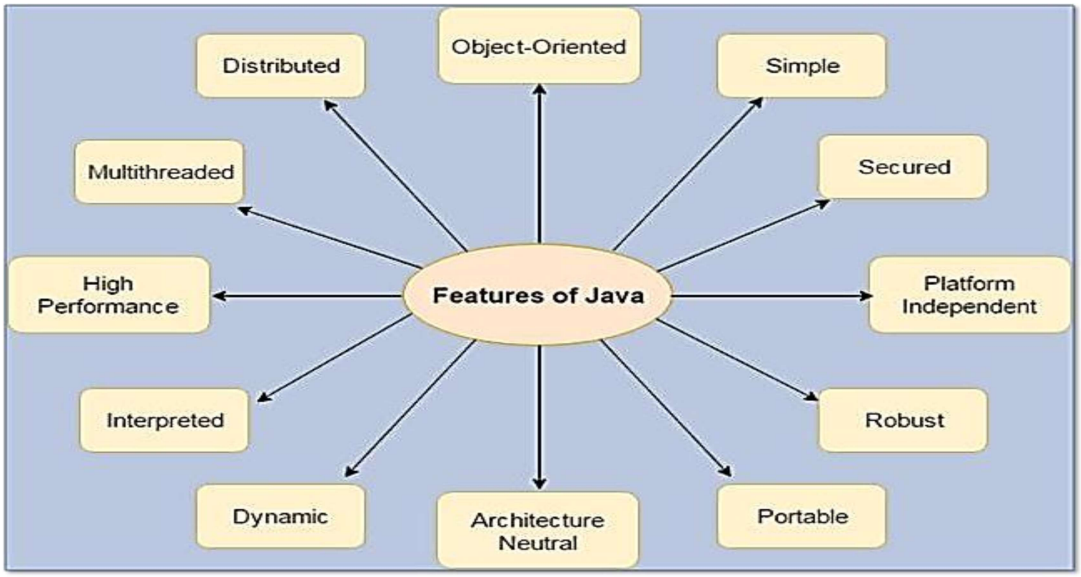
- ✓ A message for an object is a request for execution of a procedure and therefore will invoke a method (procedure) in the receiving object that generates the desired result.
- ✓ Message passing involves specifying the name of the object, the name of themethod (message) and the information to be sent.
- ✓ **Example:**

Employee.getName(name);
Where,
Employee – object name
getName – method name (message)
name - information

1.3 Java Buzzwords

The following are the features of the Java language:

1. Object Oriented
2. Simple
3. Secure
4. Platform Independent
5. Robust
6. Portable
7. Architecture Neutral
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed



1. **Object Oriented:**

- ✓ Java programming is pure object-oriented programming language. Like C++, Javaprovides most of the object oriented features.
- ✓ Though C++ is also an object oriented language, we can write programs in C++ without aclass but it is not possible to write a Java program without classes.
- ✓ Example: **Printing "Hello" Message.**

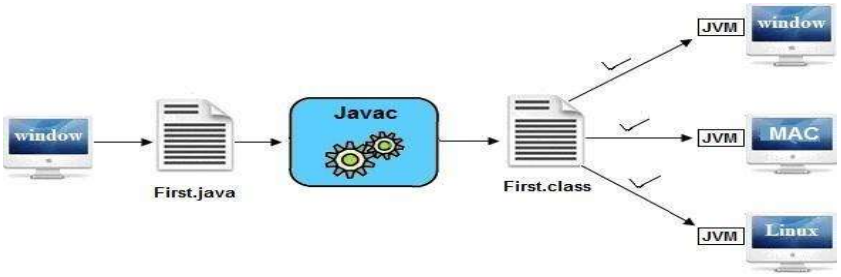
C++ (can be without class)	**Java – No programs without classes and objects**

With Class:
#include<iostream.h>
class display {
public:
void disp()
{
cout<<"Hello!";
}
};
main()
{
display d;
d.disp();
}
 	
Without class: #include<iostream.h>
void main()
{
clrscr();
cout<<"\n Hello!";
getch();
}
	With class:
	import java.io.*;
	class Hello {
	public static void main(String args[])
	{
	System.out.println("Hello!");
	}
	}
	
	Without class is not possible

2. **Simple:**
   - ✓ Java is Easy to write and more readable and eye catching.
   - ✓ Most of the concepts are drew from C++ thus making Java learning simpler.

3. **Secure :**
   - ✓ Since Java is intended to be used in networked/distributed environments, lot of emphasishas been placed on security.
   - ✓ Java provides a secure means of creating Internet applications and to access web applications.
   - ✓ Java enables the construction of secured, virus-free, tamper-free system.

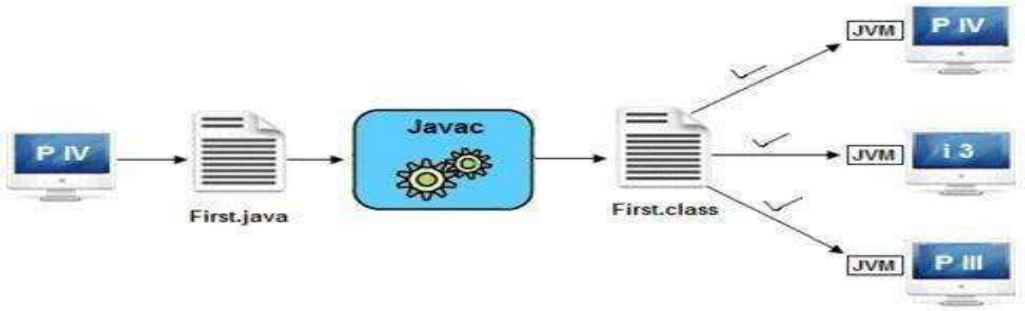
4. **Platform Independent:**
   - ✓ Unlike C, C++, when Java program is compiled, it is not compiled into platform-specific machine code, rather it is converted into platform independent code called **bytecode.**
   - ✓ The Java bytecodes are not specific to any processor. They can be executed in anycomputer without any error.
   - ✓ Because of the bytecode, Java is called as **Platform Independent.**



5. **Robust:**
   - ✓ Java encourages error-free programming by being strictly typed and performing run-time checks.

6. **Portable:**
   - ✓ Java bytecode can be distributed over the web and interpreted by **Java Virtual Machine (JVM)**
   - ✓ Java programs can run on any platform (Linux, Window, Mac)
   - ✓ Java programs can be transferred over world wide web (e.g applets)

7. **Architecture Neutral:**
   - ✓ Java is not tied to a specific machine or operating system architecture.
   - ✓ Machine Independent i.e Java is independent of hardware.
   - ✓ Bytecode instructions are designed to be both easy to interpret on any machine and easilytranslated into native machine code.



8. **Dynamic and Extensible:**
   - ✓ Java is a more dynamic language than C or C++. It was developed to adapt to an evolvingenvironment.
   - ✓ Java programs carry with them substantial amounts of run-time information that are usedto verify and resolve accesses to objects at run time.
9. **Interpreted:**
   - ✓ Java supports cross-platform code through the use of Java bytecode.
   - ✓ The Java interpreter can execute Java Bytecodes directly on any machine to which theinterpreter has been ported.

10. **High Performance:**
    - ✓ Bytecodes are highly optimized.
    - ✓ JVM can execute the bytecodes much faster.
    - ✓ With the use of Just-In-Time (JIT) compiler, it enables high performance.

11. **Multithreaded:**
    - ✓ Java provides integrated support for multithreaded programming.
    - ✓ Using multithreading capability, we can write programs that can do many tasks simultaneously.
    - ✓ The benefits of multithreading are better responsiveness and real-time behavior.

12. **Distributed:**
    - ✓ Java is designed for the distributed environment for the Internet because it handles TCP/IP protocols.
    - ✓ Java programs can be transmit and run over internet.

1.4: Overview of Java

- Java programming language was originally developed by Sun Microsystems which was initiated by **James Gosling** and released in **1995** as core component of Sun Microsystems' Java platform **(Java 1.0 [J2SE]).**

Java is a high-level object-oriented programming language, which provides developers with the means to create powerful applications, which are very small in size, platform independent, secure and robust.

- Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions ofUNIX.
- Java is mainly used for Internet Programming.
- Java is related to the languages C and C++. From C, Java inherits its syntax and from C++,Java inherits its OOP concepts.
- Ancestors of Java: - C, C++, B, BCPL.

Five primary goals in the creation of the Java language:

1. It should use the object-oriented programming methodology.
2. It should allow the same program to be executed on multiple operating systems.
3. It should contain built-in support for using computer networks.
4. It should be designed to execute code from remote sources securely.
5. It should be easy to use.

1.4.1: BASIC JAVA TERMINALOGIES:

1. **BYTECODE:**
   Byte code is an intermediate code generated from the source code by java compiler and it is platform independent.

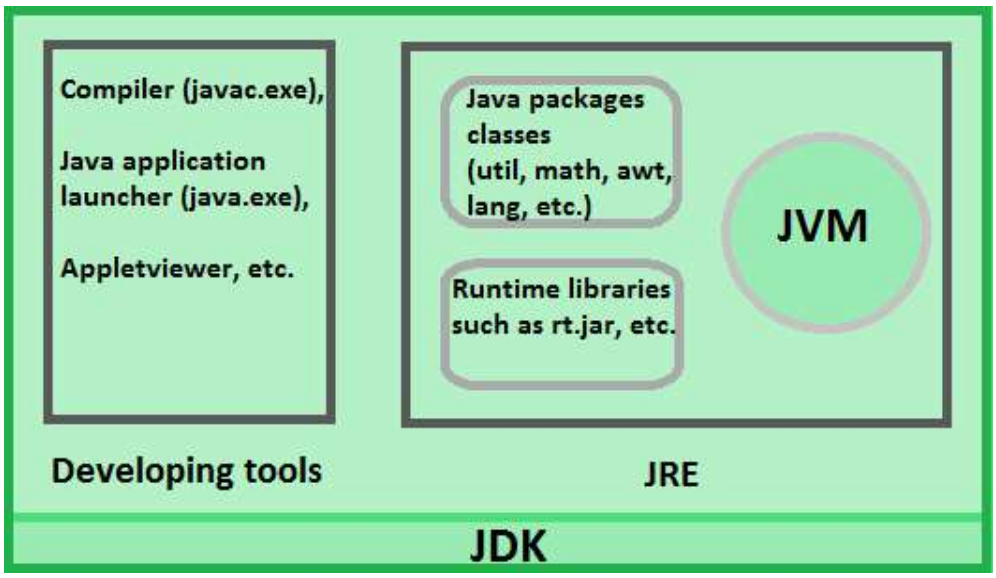
2. **JAVA DEVELOPMENT KIT (JDK):**

- The Java Development Kit (JDK) is a software development environment usedfor developing Java applications and applets.
- It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development.

3. **JAVA RUNTIME ENVIRONMENT (JRE):**
   JRE is used to provide runtime environment for JVM. It contains set of libraries +other files that JVM uses at runtime.

4. **JAVA VIRTUAL MACHINE (JVM):**

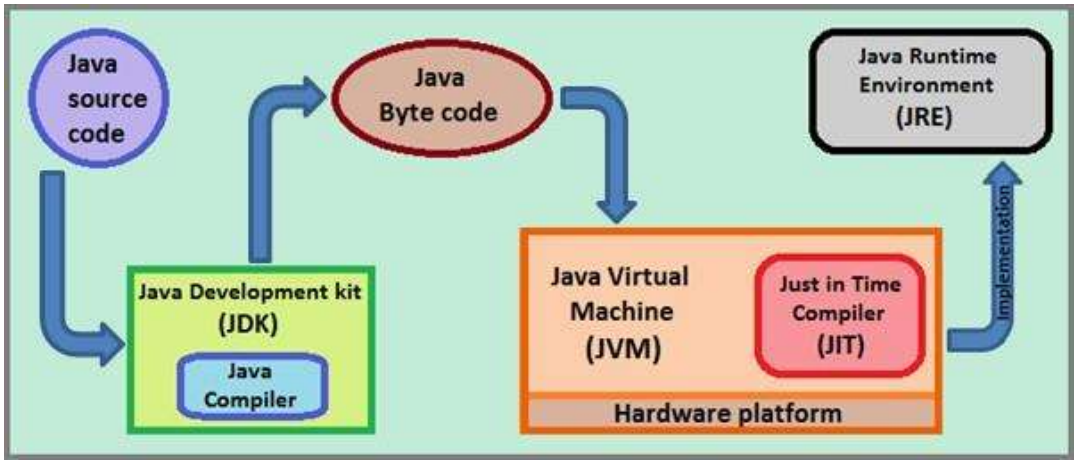
- JVM is an interpreter that converts a program in Java bytecode (intermediate language) into native machine code and executes it.
- JVM needs to be implemented for each platform because it will differ from platform to platform.



- The JVM performs following main tasks:
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment

5. JIT (JUST IN TIME) COMPILER:

It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time neededfor compilation.



Types of Java program:

In Java, there are two types of programs namely,

1. Application Program
2. Applet Program

1. Application Programs

Application programs are stand-alone programs that are written to carry out certain tasks on local computer such as solving equations, reading and writing files etc. The application programs can be executed using two steps:

1. Compile source code to generate Byte code using javac compiler.
2. Execute the byte code program using Java interpreter.

2. Applet programs:

Applets are small Java programs developed for Internet applications. An applet located in distant computer can be downloaded via Internet and executed on a local computer using Java capable browser. The Java applets can also be executed in the command line using appletviewer, which is part of the JDK.

1.4.2: JAVA SOURCE FILE - STRUCTURE – COMPILATION

THE JAVA SOURCE FILE:

A Java source file is a plain text file containing Java source code and having .java extension. The .java extension means that the file is the Java source file. Java source code file contains source code for a class, interface, enumeration, or annotation type. There are some rules associated to Java source file.

Java Program Structure:

Java program may contain many classes of which only one class defines the main method.

A Java program may contain one or more sections.

Documentation Section
Package Statement
Import Statements
Interface Statements
Class Definitions

main Method Calss
{
Main Method Definition
}

Of the above Sections shown in the figure, the Main Method class is Essential part, Documentation Section is a suggested part and all the other parts are optional.

Documentation Section

- ✓ It Comprises a Set of comment lines giving the name of the program, the authorand other details.

- ✓ Comments help in Maintaining the Program.
- ✓ Java uses a Style of comment called *documentation comment.*
  /* * …… */
- ✓ This type of comment helps is generating the documentation automatically.
- ✓ **Example:**

```
/*
 * Title: Conversion of Degrees
 * Aim: To convert Celsius to Fahrenheit and vice versa
 * Date: 31/08/2000
 * Author: tim
*/
```

Package Statement

- ✓ The first statement allowed in a Java file is a package statement.
- ✓ It declares the package name and informs the compiler that the classes defined belong to this package.
- ✓ **Example :**

  package student;

  package basepackage.subpackage.class;
- ✓ It is an optional declaration.

Import Statements

- ✓ The statement instructs the interpreter to load a class contained in a particular package.
- ✓ Example :

  import student.test;

  Where, student is the package and test is the class.

Interface Statements

- ✓ An interface is similar to classes which consist of group of method declaration.
- ✓ Like classes, interfaces contain methods and variable.
- ✓ To link the interface to our program, the keyword **implements** is used.
- ✓ **Example:**

  public class xx extends Applet implements ActionListener

  where, xx – class name (subclass of Applet)Applet – Base class name
  ActionListener – interface Extends & implements - keywords
- ✓ It is used when we want to implement the feature of Multiple Inheritance in Java
- ✓ It is an optional declaration.

Class Definitions

- ✓ A Java Program can have any number of class declarations.

- ✓ The number of classes depends on the complexity of the program.

Main Method Class

- ✓ Every Java Standalone program requires a main method as its starting point.
- ✓ A Simple Java Program will contain only the main method class.
- ✓ It creates objects of various classes and uses those objects for performing various operations.
- ✓ When the end of main is reached the program terminates and the control transferred back to the Operating system.
- ✓ **Syntax for writing main:**
  public static void main(String arg[])
  where,
  public – It is an access specifier to control the visibility of class members. main() must be declared as public, since it must be called by code outside of its class when the program is started.
  static – this keyword allows main() method to be called without having to instantiate the instance of the class.
  void – this keyword tells the compiler that main() does not return any value.
  main() – is the method called when a Java application begins.
  String arg[] – arg is an string array which receives any command-line arguments present when the program is executed.

<u>Rules to be followed to write Java Programs:</u>

About Java programs, it is very important to keep in mind the following points.

- ***Case Sensitivity - Java is case sensitive, which means identifier Hello and hellowould have different meaning in Java.***
- ***Class Names - For all class names the first letter should be in Upper Case.***
  If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
  Example class MyFirstJavaClass
- ***Method Names - All method names should start with a Lower Case letter.***
  If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.
  Example public void myMethodName()
- ***Program File Name - Name of the program file should exactly match the classname.***
  When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match your program will not compile).
  Example : Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as

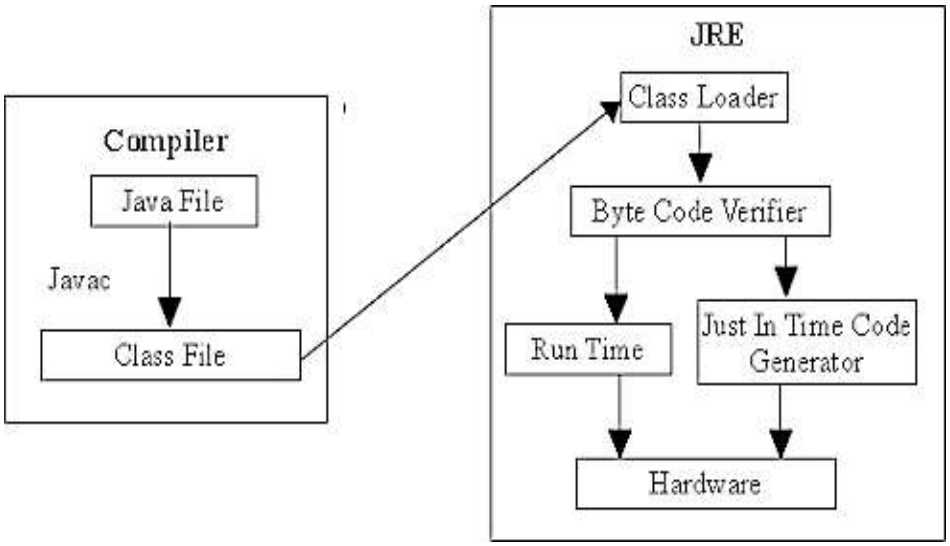
'MyFirstJavaProgram.java'

- ***public static void main(String args[]) - Java program processing starts from the main() method which is a mandatory part of every Java program.***

Compiling and running a java program in command promptSTEPS:

1. Set the path of the compiler as follows (type this in command prompt):
   Set path="C:\Program Files\Java\jdk1.6.0_20\bin";
2. To create a Java program, ensure that the name of the class in the file is the sameas the name of the file.
3. Save the file with the extension .java (Example: HelloWorld.java)
4. To compile the java program use the command javac as follows:
   javac HelloWorld.java
   This will take the source code in the file HelloWorld.java and create the javabytecode in a file HelloWorld.class
5. To run the compiled program use the command java as follows:
   java HelloWorld
   (Note that you do not use any file extension in this command.)

   At compile time, java file is compiled by Java Compiler (It does not interact with OS) andconverts the java code into bytecode.



Class Loader : is the subsystem of JVM that is used to load class files.
Bytecode Verifier : checks the code fragments for illegal code that can violate access right to objects
Interpreter : read bytecode stream then execute the instructions.

Example 1: A First Java Program:

```
public class HelloWorld
{
public static void main(String args[])
{
System.out.println("Hello World");
}
}
```

Save: HelloWorld.java **Compile:** javac HelloWorld.java**Run:** java HelloWorld
Output: Hello World

Program Explanation:
public is the access specifier, **class** is a keyword and **HelloWorld** is the class name. **{** indicates the start of program block and **}** indicates the end of the program block. **System.out.println() –** is the output statement to print some message on the screen. Here, **System** is a predefined class that provides access to the system, **out** is the output stream that is connected to the console and **println()** is method to display the given string.

Example 2: A Second Java Program:

```
import java,util.Scanner; // Scanner is a class which contains necessary methods
                             to provide a user an access to the i/p console.
public class Example2                  // class declaration
{                                      // class definition starts
public static void main(String args[])
{
//main() definition starts
    int num=0,res;  // declares two integer with initial value 0
    Scanner in=new Scanner(System.in); //creating object of Scanner class toaccess the i/p stream.
    System.out.println("Enter a Number : ");
     num=in.nextInt();                 // to read the next integer value from the i/pstream
    res=num*2;                         // manipulation of the data
     System.out.println("The value of "+num+" * 2 = "+res);  //displaysresult
}
}
```

Save: Example2.java **Compile:** javac Example2.java**Run:** java Example2

Output:
Enter a Number: 25
The value of 25 * 2 = 50

1.5: JAVA – DATA TYPES

Data type is used to allocate sufficient memory space for the data. Data types specify the different sizes and values that can be stored in the variable.

- ***Java is a strongly Typed Language.***
- Definition: ***strongly Typed Language:***

Java is a strongly typed programming language because every variable must be declared with a data type. A variable cannot start off life without knowing the range of values it can hold, and once it is declared, the data type of the variable cannot change.

Data types in Java are of **two types**:

1. **Primitive data types (Intrinsic or built-in types) :- :** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types (Derived or Reference Types):** The non-primitive data types include Classes, Interfaces, Strings and Arrays.

1. Primitive Types:

Primitive data types are those whose variables allow us to store only one value and neverallow storing multiple values of same type. This is a data type whose variable can hold maximum one value at a time.

There are eight primitive types in Java:

Integer Types:

1. int
2. short
3. lomg
4. byte

Floating-point Types:

5. float
6. double

Others:

7. char
8. Boolean

- **Integer Types:**

The integer types are form numbers without fractional parts. Negative values are allowed.Java provides the four integer types shown below:

Type	Storage Requirement	Range	Example	Default Value
int	4 bytes	-2,147,483,648(-2^31) to 2,147,483,647 (2^31-1)	int a = 100000, int b = -200000	0
short	2 bytes	-32,768 (-2^15) to 32,767 (2^15-1)	short s = 10000, short r = -20000	0
long	8 bytes	-9,223,372,036,854,775,808 (-2^63) to 9,223,372,036,854,775,808 (2^63-1)	long a = 100000L, int b = -200000L	0L
byte	1 byte	-128 (-2^7) to 127 (2^7-1)	byte a = 100 , byte b = -50	0

- **Floating-point Types:**

The floating-point types denote numbers with fractional parts. The two floating-pointtypes are shown below:

Type	Storage Requirement	Range	Example	Default Value
float	4 bytes	Approximately ±3.40282347E+38F (6-7 significant decimal digits)	float f1 =234.5f	0.0f
double	8 bytes	Approximately ±1.79769313486231570E+308 (15 significant decimal digits)	double d1 = 123.4	0.0d

- **char:**
  - char data type is a single 16-bit Unicode character.
  - Minimum value is '\u0000' (or 0).
  - Maximum value is '\uffff' (or 65,535 inclusive).
  - Char data type is used to store any character.
  - Example: char letterA ='A'

- **boolean:**
  - boolean data type represents one bit of information.
  - There are only two possible values: true and false.
  - This data type is used for simple flags that track true/false conditions.
  - Default value is false.
  - Example: boolean one = true

2. Derived Types (Reference Types):

- **Derived data types are those whose variables allow us to store multiple values of same type. But they never allow storing multiple values of different types.**
- A reference variable can be used to refer to any object of the declared type or any

compatible type.

- These are the data type whose variable can hold more than one value of similar type.
- **The value of a reference type variable, in contrast to that of a primitive type, is a reference to (an address of) the value or set of values represented by the variable.**
- Example

```
int a[] = {10,20,30};                  // valid
int b[] = {100, 'A', "ABC"};        // invalid
Animal animal = new Animal("giraffe");  //Object
```

1.6: JAVA - VARIABLES

- **A Variable is a named piece of memory that is used for storing data in java Program.**
- A variable is an identifier used for storing a data value.
- A Variable may take different values at different times during the execution if the program, unlike the constants.
- The variable's type determines what values it can hold and what operations can beperformed on it.
- **Syntax to declare variables:**

  datatype identifier [=value][,identifier [=value] ...];

- Example of Variable names:

  int average=0.0, height, total height;

- **Rules followed for variable names (consist of alphabets, digits, underscore anddollar characters)**
  1. A variable name must begin with a letter and must be a sequence of letter or digits.
  2. They must not begin with digits.
  3. Uppercase and lowercase variables are not the same.
     a. **Example:** Total and total are two variables which are distinct.
  4. It should not be a keyword.
  5. Whitespace is not allowed.
  6. Variable names can be of any length.

- **Initializing Variables:**
  - ✓ After the declaration of a variable, it must be initialized by means of assignment statement.
  - ✓ It is not possible to use the values of uninitialized variables.

- ✓ Two ways to initialize a variable:

1. Initialize after declaration:

Syntax: **variablename=value;**

```
int months;
months=1;
```

2. Declare and initialize on the same line:

Syntax: **Datatype variablename=value;**

```
int months=12;
```

- **Dynamic Initialization of a Variable:**

Java allows variables to be initialized dynamically using any valid expression at the timethe variable is declared.

Example: Program that computes the remainder of the division operation:

```
class FindRemainer
{
public static void main(String arg[]) {int num=5,den=2;
int rem=num%den; System.out.println(-Remainder is –+rem);
}
}
```

Output:

Remainder is 1

In the above program there are three variables **num, den** and **rem. num** and **den** ate initialized by constants whereas **rem** is initialized dynamically by the modulo division operation on **num** and **den.**

JAVA - VARIABLE TYPES

There are three kinds of variables in Java:

1. Local variables
2. Instance variables
3. Class/static variables

Local Variables	Instance Variable	Class / Static Variables
Local variables aredeclared in methods, constructors, or blocks.	Instance variables are declared in a class, but outside a method, constructor or any block.	Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block. There would only be one copy of each class variable per class, regardless of how many objects are created from it.
Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.	Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.	Static variables are created when the program starts and destroyed when the program stops.
Access modifiers cannot be used for local variables.	Access modifiers can be used forinstance variables.	Access modifiers can be used for class variables.
Local variables are visible only within the declared method, constructor or block.	The instance variables are visible for all methods, constructors and block in the class.	Visibility is similar to instance variables.
There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.	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.	Default values are same as instance variables.
Local variables can only be access inside the declared block.	Instance variables can be accessed directly by calling the variable name inside the class. However within static methods and different class should be called using the fully qualified name as follows: **ObjectReference.VariableName.**	Static variables can be accessed by calling with the class name. **ClassName.VariableName.**

Example program illustrating the use of all the above variables:

```
class area
{
int length=20;
int breadth=30;
```

```
static int classvar=2500;
void calc()
{
        int areas=length*breadth;
        System.out.println("The area is "+areas+" sq.cms");
}

public static void main(String args[])
{
   area a=new area();
   a.calc();
   System.out.println("Static Variable Value : "+classvar);
}
}
```

Output:

The area is 600 sq.cms
Static Variable Value : 2500

Program Explanation:

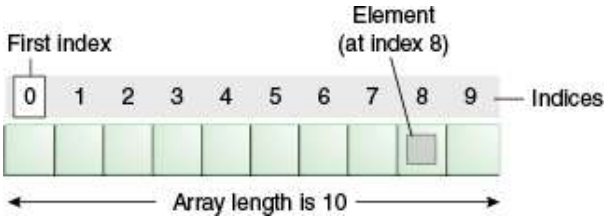
Class name: area
Method names: calc() and main()
Local variables: areas (accessed only in the particular method)
Instance variables: length and breadth (accessed only through the object's method)
Static variable: accessed anywhere in the program, without object reference

1.7: ARRAYS

Definition:

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

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



Advantage of Array:

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

Disadvantage of Array:

- **Size Limit:** We can store only fixed size of elements in the array. It doesn't growits size at runtime.

Types of Array:

There are two types of array.

1. One-Dimensional Arrays
2. Multidimensional Arrays

1. One-Dimensional Array:

Definition: One-dimensional array is an array in which the elements are stored in onevariable name by using only one subscript.

- **Creating an array:**

Three steps to create an array:

1. Declaration of the array
2. Instantiation of the array
3. Initialization of arrays

1. Declaration of the array:

Declaration of array means the specification of array variable, data_type and array_name.

Syntax to Declare an Array in java:

```
dataType[]  arrayRefVar;   (or)
dataType  []arrayRefVar;   (or)
dataType arrayRefVar[];
```

Example:

int[] floppy; (or) int []floppy (or) int floppy[];

2. **Instantiation of the array:**

Definition:

Allocating memory spaces for the declared array in memory (RAM) is called as **Instantiation of an array.**

Syntax:

arrayRefVar=new datatype[size];

Example: floppy=new int[10];

3. **Initialization of arrays:Definition:**

Storing the values in the array element is called as **Initialization of arrays.**

Syntax to initialize values to array element:

arrayRefVar[index value]=constant or value;

Example:

floppy[0]=20;

SHORTHAND TO CREATE AN ARRAY OBJECT:

Java has shorthand to create an array object and supply initial values at the same timewhen it is created.

```
dataType[] arrayRefVar={list of values};
                    (or)
dataType []arrayRefVar={list of values};
                    (or)
dataType arrayRefVar[]={list of values};
                    (or)
dataType arrayRefVar[]=arrayVariable;
```

Example 1:

```
int regno[]={101,102,103,104,105,106};
int reg[]=regno;
```

Example 2: double[] myList = new double[10];

ARRAY LENGTH:

The variable **length** can identify the length of array in Java. To find the number of elements of an array, use **array.length.**

Example1:

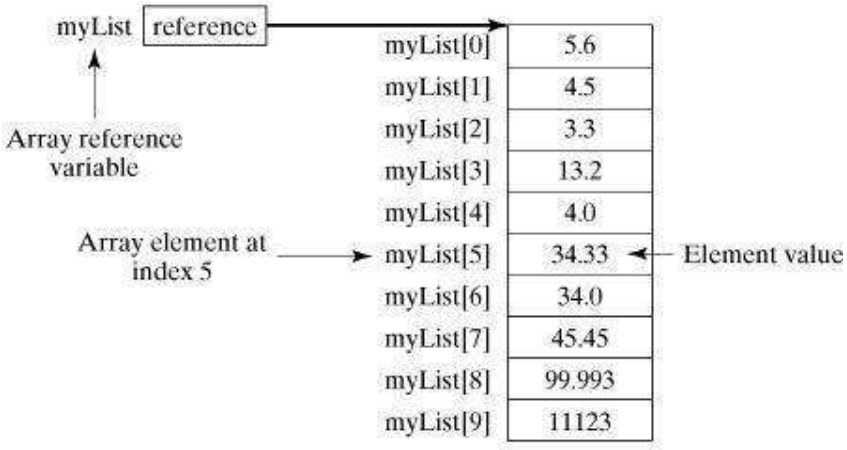
int regno[10]; len1=regno.length;

Example 2:

```
for(int i=0;i<reno.length;i++)
    System.out.println(regno[i]);
```

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

Example: (One-Dimensional Array)




```
class Array
{
public static void main(String[] args)
{
        int month_days[];
        month_days=new int[12];
        month_days[0]=31;
        month_days[1]=28;
        month_days[2]=31;
        month_days[3]=30;
        month_days[4]=31;
        month_days[5]=30;
        month_days[6]=31;
        month_days[7]=31;
        month_days[8]=30;
        month_days[9]=31;
        month_days[10]=30;
        month_days[11]=31;

        System.out.println("April has "+month_days[3]+ " days.");
}
}
```

Output:

April has 30 days.

Example 2: Finding **sum of the array elements** and **maximum** from the array:

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

        // Print all the array elements
        for (double element: myList)
        {
                System.out.println(element);
        }

        // Summing all elementsdouble total = 0;
        for (int i = 0; i < myList.length; i++)
        {
                total += myList[i];
        }
        System.out.println("Total is " + total);

        // Finding the largest elementdouble

        max = myList[0];
        for (int i = 1; i < myList.length; i++)
        {
                if (myList[i] > max)
                        max = myList[i];
        }
        System.out.println("Max is " + max);
    }
}
```

Output:

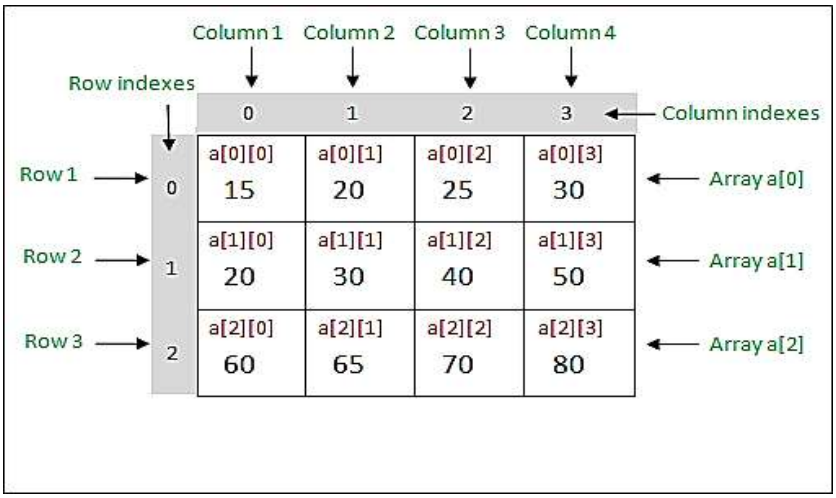
1.9
2.9
3.4
3.5

Total is 11.7
Max is 3.5

2. Multidimensional Arrays:

Definition:

Multidimensional arrays are ***arrays of arrays.*** It is an array which uses more than one index to access array elements. In multidimensional arrays, data is stored in row and column based index (also known as matrix form).



Uses of Multidimensional Arrays:

- ✓ Used for table
- ✓ Used for more complex arrangements

Syntax to Declare Multidimensional Array in java:

1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in java:

int[][] arr=new int[3][3]; //3 row and 3 column - internally this matrix is implemented as *arrays of arrays* of **int.**

Example to initialize Multidimensional Array in java:

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
```

```
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

Examples to declare, instantiate, initialize and print the 2Dimensional array:

```
class twoDarray
{
public static void main(String args[])
{
int array1[][]=new int[4][5];// declares an 2D array.
int array2[][]={{1,2,3},{2,4,5},{4,4,5}}; //declaring and initializing 2D arrayint i,j,k=0;

  // Storing and printing the values of Array1

System.out.println("-------Array 1------- " );
for(i=0;i<4;i++)
{
     for(j=0;j<5;j++)
     {
          array1[i][j]=k;k++;
          System.out.print(array1[i][j]+ " ");
     }
     System.out.println();
}

// printing 2D array2
System.out.println("-------Array 2------- ");
for(int i=0;i<3;i++)
     {
     for(int j=0;j<3;j++)
     {
     System.out.print(array2[i][j]+
     }
     System.out.println();
     }
     }
}
```

Output:

----------Array1----------
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19

----------Array2----------
1 2 3
2 4 5
4 4 5

In the above program, the statement **int array1[][]=new int[4][5];** is interpreted automatically as follows:
array1[0]=new int[5];array1[1]=new int[5];array1[2]=new int[5];array1[3]=new int[5];

It means that, when we allocate memory for a multidimensional array, we need to only specify the memory for the first (leftmost) dimension. We can allocate the remaining dimensions separately with different sizes.

Example: Manually allocate differing size second dimensions:

```
class twoDarray
{
public static void main(String args[])
{
int array1[][]=new int[4][];              // declares an 2D array.

array1[0]=new int[1];
array1[1]=new int[2];
array1[2]=new int[3];
array1[3]=new int[4];
int i,j,k=0;

  // Storing and printing the values of Array
for(i=0;i<4;i++)
{
     for(j=0;j<i+1;j++)
     {
              array1[i][j]=k;k++;
              System.out.print(array1[i][j]+ " ");
```

```
    }
    System.out.println();
}
}
}
```

Output:

```
0
1 2
3 4 5
6 7 8 9
```

1.8: OPERATORS

Operators are used to manipulate primitive data types.
Java operators can be classified as unary,binary, or ternary—meaning taking one, two, or three arguments, respectively.

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations
i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Java Unary Operator Example: ++ and –

```
1. class OperatorExample
2. {
3. public static void main(String args[])
4. {
5. int x=10;
6. System.out.println(x++);          //10 (11)
7. System.out.println(++x);          //12
8. System.out.println(x--);          //12 (11)
9. System.out.println(--x);          //108.
10.}
11.}
```

Output:

```
10
12
12
10
```

Java Unary Operator Example 2: ++ and –

```
1. class OperatorExample
2. {
3. public static void main(String args[])
4. {
5. int a=10;
6. int b=10;
7. System.out.println(a++ + ++a);          //10+12=22
8. System.out.println(b++ + b++);          //10+11=21 7.
9.  }
10. }
```

Output:

```
22
21
```

Java Unary Operator Example: ~ and !

```
1. class OperatorExample{
2. public static void main(String args[]){
3.       int a=10;
4.       int b=-10;
5. boolean c=true;
6. boolean d=false;
7. System.out.println(~a);        //-11 (minus of total positive value which starts from 0)
8. System.out.println(~b);              //9 (positive of total minus, positive starts from 0)
9. System.out.println(!c);        //false (opposite of boolean value)
10.       System.out.println(!d);              //true
11.       }
12.       }
```

Output:

```
-11
9
False
true
```

A binary or ternary operator appears between its arguments.Java operators fall into

eight different categories:

1. **Assignment**
2. **Arithmetic**
3. **Relational**
4. **Logical**
5. **Bitwise**
6. **Compound assignment**
7. **Conditional**
8. **Type.**

Assignment Operators	**=**
Arithmetic Operators	**- + * / % ++ --**
Relational Operators	**> < >= <= == !=**
Logical Operators	**&& || & | ! ^**
Bit wise Operator	**& | ^ >> >>>**
Compound Assignment Operators	**+= -= *= /= %= <<= >>= >>>=**
Conditional Operator	**?:**

1. Java Assignment Operator

The java assignment operator statement has the following syntax:

<*variable*> = <*expression*>

If the value already exists in the variable it is overwritten by the assignment operator (=).

Java Assignment Operator Example

```
1. class OperatorExample{
2. public static void main(String args[])
3. {
4. int a=10;
5. int b=20;
6. a+=4;                    //a=a+4 (a=10+4)
7. b-=4;                    //b=b-4 (b=20-4)
8. System.out.println(a);
9. System.out.println(b);
10. }
11. }
```

Output:

```
14
16
```

2. Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division.They act as basic mathematical operations.

Assume integer variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

Java Arithmetic Operator Example: Expression

```
1. class OperatorExample
2. {
3. public static void main(String args[])
4. {
5.     System.out.println(10*10/5+3-1*4/2);4.
6. }
7. }
```

Output:

```
21
```

3. Relational Operators

Relational operators in Java are used to compare 2 or more objects. Java provides sixrelational operators: Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, ifvalues are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value ofright operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal tothe value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomestrue.	(A <= B) is true.

Example:

```
public RelationalOperatorsDemo( )
{
      int x = 10, y = 5;
      System.out.println("x > y : "+(x > y));
      System.out.println("x < y : "+(x < y));
      System.out.println("x >= y : "+(x >= y));
      System.out.println("x <= y : "+(x <= y));
      System.out.println("x == y : "+(x == y));
      System.out.println("x != y : "+(x != y));

  public static void main(String args[])
  {
        new RelationalOperatorsDemo();
  }
  }
```

Output:

```
$java RelationalOperatorsDemo
```

x > y : true
x < y : false
x >= y : true
x <= y : false
x == y : false
x != y : true

4. Logical Operators

Logical operators return a true or false value based on the state of the Variables. Given that x and y represent boolean expressions, the boolean logical operators are defined in the Table below.

x	y	!x	x & y x && y	x | y x || y	x ^ y
true	true	false	true	true	False
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Example:

```
public class LogicalOperatorsDemo
{
   public LogicalOperatorsDemo()
   {
      boolean x = true;
      boolean y = false;
      System.out.println("x & y : " + (x & y));
      System.out.println("x && y : " + (x && y));
      System.out.println("x | y : " + (x | y));
      System.out.println("x || y: " + (x || y));
      System.out.println("x ^ y : " + (x ^ y));
      System.out.println("!x : " + (!x));
   }
public static void main(String args[])
{
new LogicalOperatorsDemo();
}
}
```

Output:

```
$java LogicalOperatorsDemo
```

x & y : false
x && y : false
x | y : true
x || y: true
x ^ y : true
!x : false

5. Bitwise Operators

Java provides Bit wise operators to manipulate the contents of variables at the bit level. The result of applying bitwise operators between two corresponding bits in the operandsis shown in the Table below.

A	B	~A	A & B	A | B	A ^ B
1	**1**	0	1	1	0
1	**0**	0	0	1	1
0	**1**	1	0	1	1
0	**0**	1	0	0	0

```
public class Test
{
public static void main(String args[])
{
    int a = 60;  /* 60 = 0011 1100 */
    int b = 13;    /* 13 = 0000 1101 */int c = 0;
    c = a & b;      /* 12 = 0000 1100 */
    System.out.println("a & b = " + c );
    c = a | b;     /* 61 = 0011 1101 */
    System.out.println("a | b = " + c );
    c = a ^ b;     /* 49 = 0011 0001 */
    System.out.println("a ^ b = " + c );
    c = ~a;        /*-61 = 1100 0011 */
    System.out.println("~a = " + c );
    c = a << 2;   /* 240 = 1111 0000 */
    System.out.println("a << 2 = " + c );
    c = a >> 2;   /* 215 = 1111 */
    System.out.println("a >> 2  = " + c );
    c = a >>> 2;   /* 215 = 0000 1111 */
```

```
    System.out.println("a >>> 2 = " + c );
}
}
```

Output:

```
$java Test

  a & b = 12
  a | b = 61
  a ^ b = 49
  ~a = -61
  a << 2 = 240
  a >> 2  = 15
  a >>> 2 = 15
```

6. Compound Assignment operators

The compound operators perform shortcuts in common programming operations. Javahas eleven compound assignment operators.

Syntax: argument1 **operator** = argument2.

Java Assignment Operator Example

```
1. class OperatorExample
2. {
3. public static void main(String[] args)
4. {
5. int a=10;
6. a+=3;     //10+3
7. System.out.println(a);
8. a-=4;     //13-4
9. System.out.println(a);
10. a*=2;    //9*2
11. System.out.println(a);
12. a/=2;    //18/2
13. System.out.println(a);12.
14. }
15. }
```

Output:

```
          13
```

9
18
9

7. Conditional Operators

The Conditional operator is the only ternary (operator takes three arguments) operator in Java. The operator evaluates the first argument and, if true, evaluates the second argument.

If the first argument evaluates to false, then the third argument is evaluated. The conditional operator is the expression equivalent of the if-else statement.

The conditional expression can be nested and the conditional operator associates from right to left: **(a?b?c?d:e:f:g) evaluates as (a?(b?(c?d:e):f):g)**

Example:

```
public class TernaryOperatorsDemo {

        public TernaryOperatorsDemo() {
                int x = 10, y = 12, z = 0;
                z = x > y ? x : y;
                System.out.println("z : " + z);
        }
        public static void main(String args[]) {
                new TernaryOperatorsDemo();
        }
}
```

Output:

```
$java TernaryOperatorsDemo
z : 12
```

8. instanceof Operator:

This operator is used only for object reference variables. The operator checks whether the object is of a particular type(class type or interface type). instanceof operator is written as:

(Object reference variable) instanceof (class/interface type)

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is the

Example:

```
public class Test
{
   public static void main(String args[])
  {
      String name = "James";
       // following will return true since name is type of String
       boolean result = name instanceof String;
       System.out.println( result );
   }
}
```

This would produce the following result:

```
True
```

OPERATOR PRECEDENCE:

The order in which operators are applied is known as precedence. Operators with a higher precedence are applied before operators with a lower precedence.

The operator precedence order of Java is shown below. Operators at the top of the table are applied before operators lower down in the table.

If two operators have the same precedence, they are applied in the order they appear in a statement. That is, from left to right. You can use parentheses to override the default precedence.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right

Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Example:

In an operation such as,

result = 4 + 5 * 3

First (5 * 3) is evaluated and the result is added to 4 giving the Final Result value as 19. Note that _*' takes higher precedence than _+' according to chart shown above. This kindof precedence of one operator over another applies to all the operators.

1.9: CONTROL-FLOW STATEMENTS

Java Control statements control the order of execution in a javaprogram, based on data values and conditional logic.

There are three main categories of control flow statements;

- **Selection statements:** if, if-else and switch.
- **Loop statements:** while, do-while and for.
- **Transfer statements:** break, continue, return, try-catch-finally and assert.

We use control statements when we want to change the default sequential order of execution

1. Selection statements (Decision Making Statement)

There are two types of decision making statements in Java. They are:

- if statements
- if-else statements
- nested if statements
- if-else if-else statements
- switch statements

if Statement:

- An if statement consists of a Boolean expression followed by one or more statements.
- Block of statement is executed when the condition is true otherwise no statement will beexecuted.

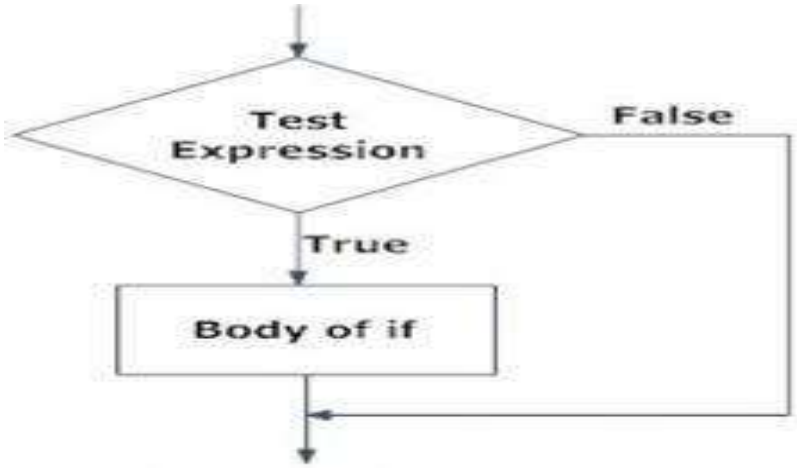
Syntax:

```
if(<conditional expression>)
{
  < Statement Action>
}
```

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed.

If not the first set of code after the end of the if statement (after the closingcurly brace) will be executed.

Flowchart:



Example:

```
public class IfStatementDemo {

        public static void main(String[] args)
        {
        int a = 10, b = 20;
        if (a > b)
              System.out.println("a > b");
         if (a < b)
              System.out.println("b > a");
        }
    }
```

Output:

```
$java IfStatementDemo
b > a
```

if-else Statement:

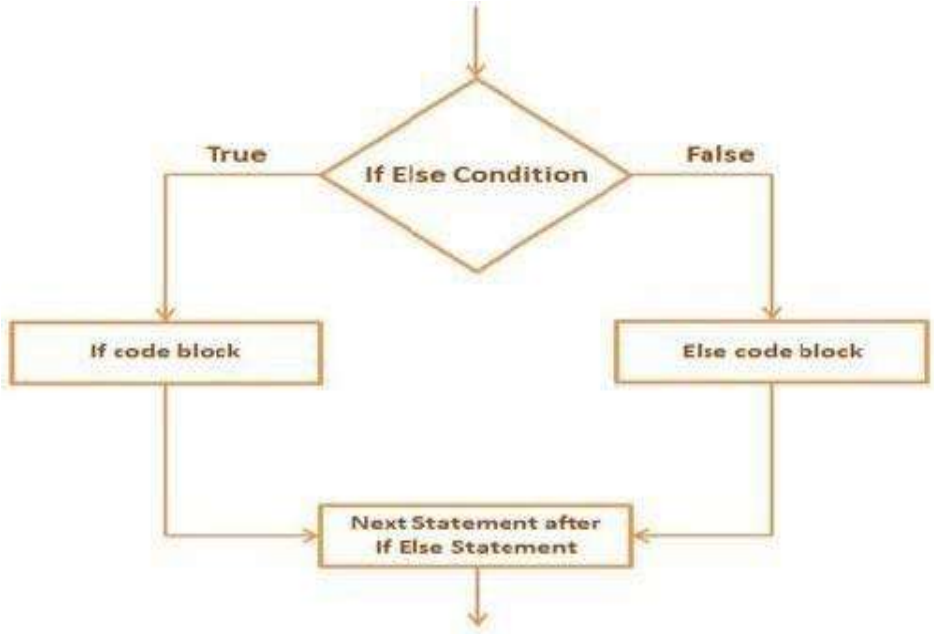
The if/else statement is an extension of the if statement. If the statements in the

ifstatement fails, the statements in the else block are executed.

Syntax:

The if-else statement has the following syntax:

```
if(<conditional expression>)
{
  < Statement Action1>
}
else
{
  < Statement Action2>
}
```



Example:

```
public class IfElseStatementDemo {

        public static void main(String[] args)
        {
                int a = 10, b = 20;
                if (a > b) {
                        System.out.println("a > b");
                }
                else {
                        System.out.println("b > a");
                }
        }
}
```

Output:

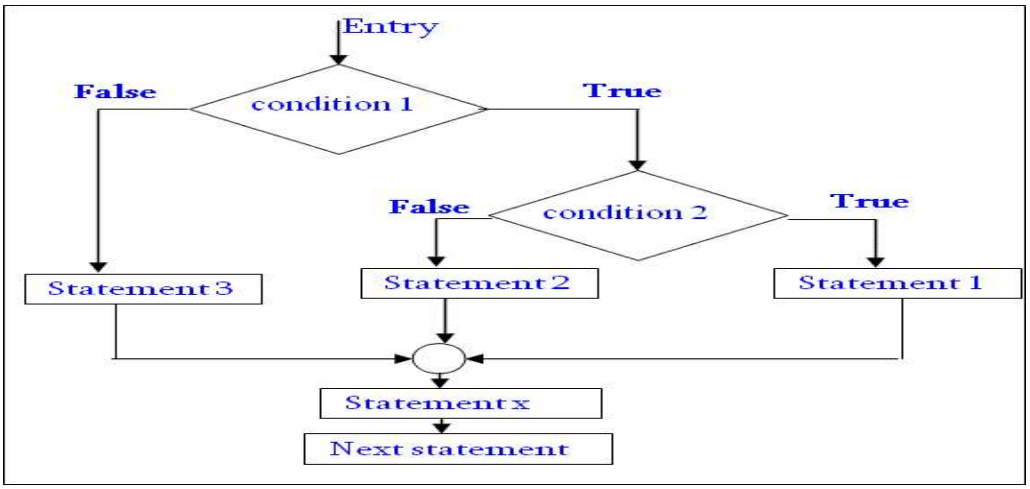
```
$java IfElseStatementDemo
      b > a
```

Nested if Statement:

Nested if-else statements, is that using one if or else if statement inside another if or else ifstatement(s).

Syntax:

```
if(condition1)
{
    if(condition2)
    {
    //Executes this block if  condition is True
    }
    else
    {
     //Executes this block if condition is false
    }
}
else
{
 //Executes this block if condition is false
}
```



Example-nested-if statement:

```
class NestedIfDemo
{
```

```
public static void main(String args[])
{
int i = 10;
if (i ==10)
{
if (i < 15)
{
System.out.println("i is smaller than 15");
}
else
{
System.out.println("i is greater than 15");
}
}
else
{
System.out.println("i is greater than 15");
}
}
}
```

Output:

i is smaller than 15

if...else if...else Statement:

An if statement can be followed by an optional *else if...else* statement, which is veryuseful to test various conditions using single if...else if statement.

Syntax:

```
if(Boolean_expression 1){
  //Executes when the Boolean expression 1 is true
}else if(Boolean_expression 2){
  //Executes when the Boolean expression 2 is true
}else if(Boolean_expression 3){
  //Executes when the Boolean expression 3 is true
}else {
  //Executes when the none of the above condition is true.
}
```

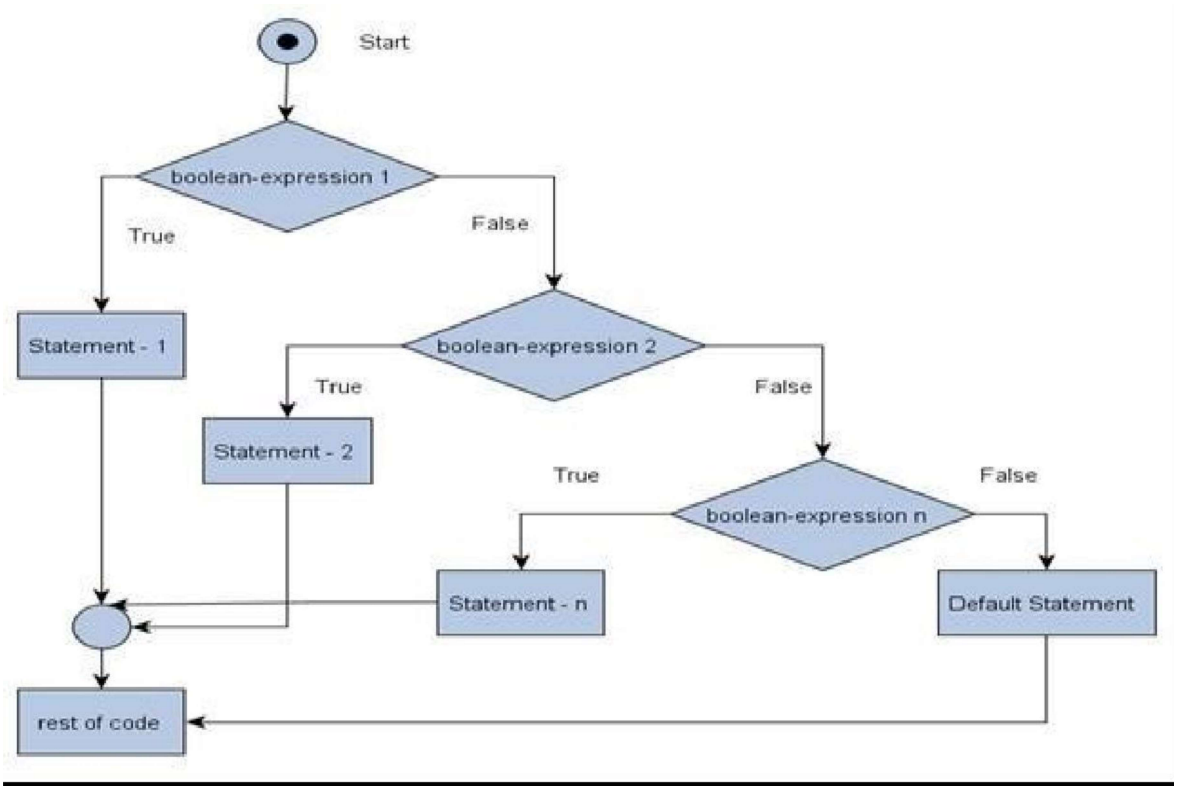
Example:

```
public class Test {

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

```
    int x = 30;

    if( x == 10 ){

      System.out.print("Value of X is 10");
    }else if( x == 20 ){
      System.out.print("Value of X is 20");
    }else if( x == 30 ){
      System.out.print("Value of X is 30");
    }else{
      System.out.print("This is else statement");
    }
  }
}
```

Output:

Value of X is 30



switch Statement:

- **The switch case statement, also called a case statement is a multi-way branch with several choices. A switch is easier to implement than a series of if/else statements.**
- A *switch* statement allows a variable to be tested for equality against a list of

values. Eachvalue is called a case, and the variable being switched on is checked for each case.

- The switch statement begins with a keyword, followed by an expression that equates to a no long integral value. Following the controlling expression is a code block that contains zero or more labeled cases. Each label must equate to an integer constant and each must be unique.
- When the switch statement executes, it compares the value of the controlling expression to the values of each case label.
- The program will select the value of the case label that equals the value of the controllingexpression and branch down that path to the end of the code block.
- If none of the case label values match, then none of the codes within the switch statementcode block will be executed.
- Java includes a default label to use in cases where there are no matches. We can have a nested switch within a case block of an outer switch.

Syntax:

switch (<expression>)
{
case label1:
<statement1>
case label2:
<statement2>
…
case labeln:
<statementn>
default:
<statement>
}

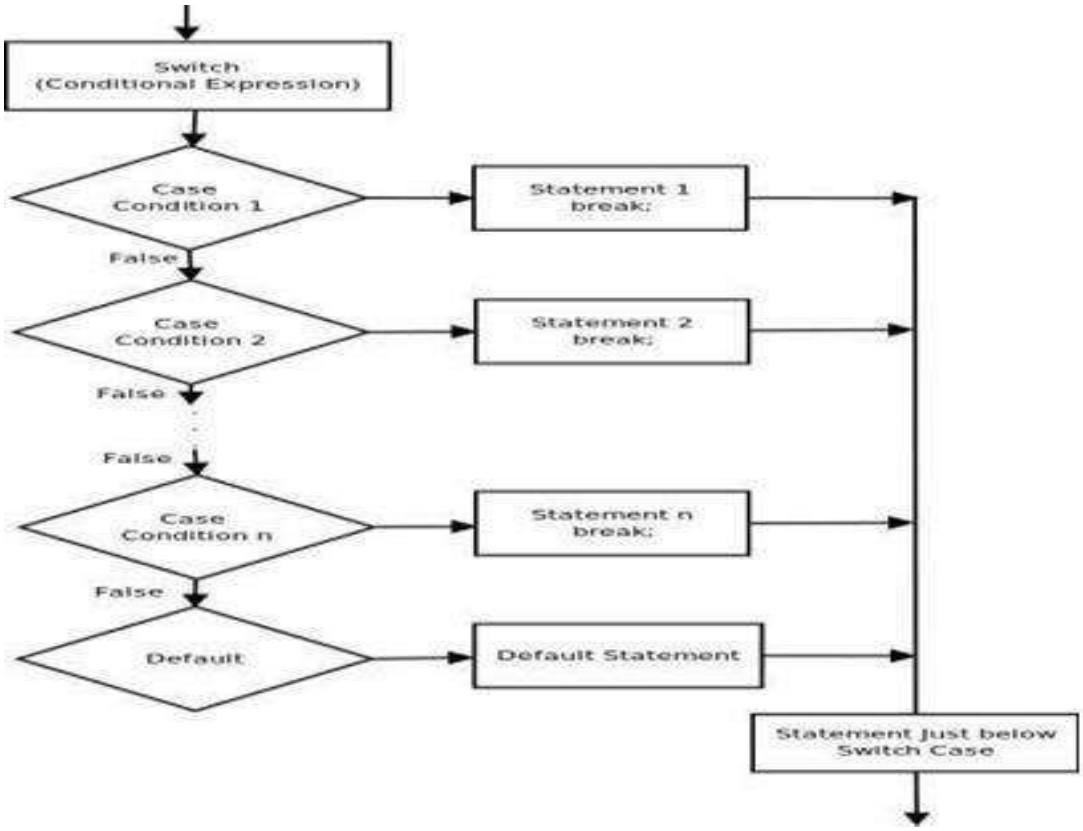
Example:

```
public class SwitchCaseStatementDemo {

    public static void main(String[] args) {int a =
            10, b = 20, c = 30;
            int status = -1;
            if (a > b && a > c) {
                    status = 1;
            } else if (b > c) {
                    status = 2;
            } else {
```

```
                    status = 3;
                }
                switch (status) {case 1:
                    System.out.println("a is the greatest");break;
                case 2:
                    System.out.println("b is the greatest");break;
                case 3:
                    System.out.println("c is the greatest");break;
                default:
                    System.out.println("Cannot be determined");
                }
        }
}
```

Output:

c is the greatest



2. Looping Statements (Iteration Statements)

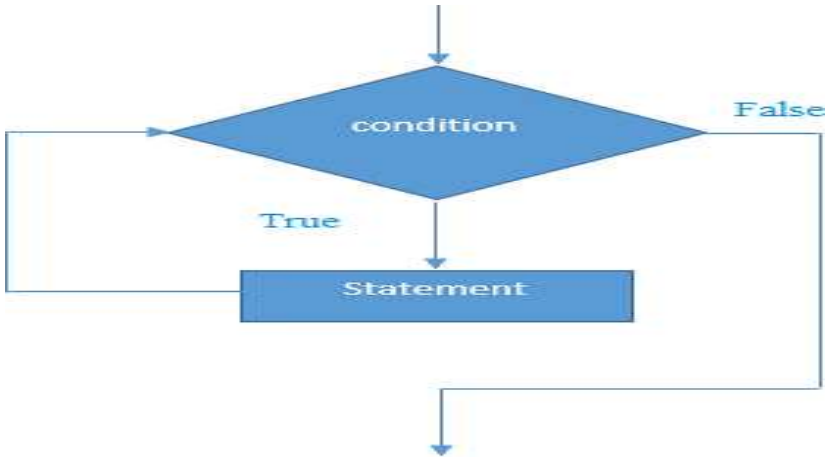
While Statement

- The while statement is a looping control statement that executes a blockof code while a condition is true. It is entry controlled loop.
- You can either have a single statement or a block of code within the while loop.The loop will never be executed if the testing expression evaluates to false.
- The loop condition must be a boolean expression.

Syntax:

The syntax of the while loop is

```
while (<loop condition>)
{
<statements>
}
```



Example:

```
public class WhileLoopDemo {
        public static void main(String[] args) {int
                count = 1;
                System.out.println("Printing Numbers from 1 to 10");
                while (count <= 10) {
                        System.out.println(count++);
                }
        }
}
```

Output

```
Printing Numbers from 1 to 10
        1
        2
        3
```

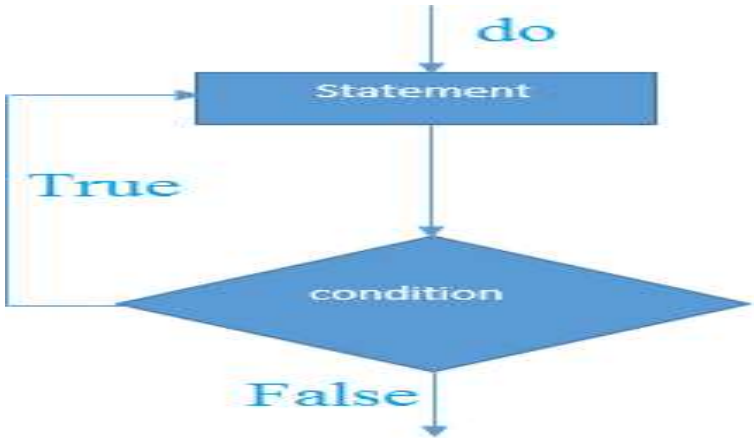
4
5
6
7
8
9
10

do-while Loop Statement

- do while loop checks the condition after executing the statements atleast once.
- Therefore it is called as Exit Controlled Loop.
- The do-while loop is similar to the while loop, except that the test is performed at the endof the loop instead of at the beginning.
- This ensures that the loop will be executed at least once. A do-while loop begins with the keyword do, followed by the statements that make up the body of the loop.

Syntax:

```
do
{
<loop body>
}while (<loop condition>);
```



Example:

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

```
            int count = 1;
            System.out.println("Printing Numbers from 1 to 10");
            do {
                    System.out.println(count++);
            } while (count <= 10);
    }
}
```

Output:

```
Printing Numbers from 1 to 10
1
2
3
4
5
6
7
8
9
10
```

For Loops

The for loop is a looping construct which can execute a set of instructions a specified number of times. It's a counter controlled loop. A for statement consumes the initialization, condition and increment/decrement in one line. It is the entry controlled loop.

Syntax:

```
for (<initialization>; <loop condition>; <increment expression>)
{
        <loop body>
}
```

- ✓ The first part of a for statement is a starting initialization, which executes once before the loop begins. The <initialization> section can also be a comma-separated list of expression statements.
- ✓ The second part of a for statement is a test expression. As long as the expression is true, the loop will continue. If this expression is evaluated as false the first time, the loop will never be executed.

- ✓ The third part of the for statement is the body of the loop. These are the instructionsthat are repeated each time the program executes the loop.
- ✓ The final part of the for statement is an increment expression that automatically executes after each repetition of the loop body. Typically, this statement changes the value of the counter, which is then tested to see if the loop should continue.

Exmple:

```
public class ForLoopDemo {
        public static void main(String[] args)
        {
        System.out.println("Printing Numbers from 1 to
                10");
        for (int count = 1; count <= 10; count++)
        {
                        System.out.println(count);
       }
 }
}
```

Output:

```
Printing Numbers from 1 to 10
1
2
3
4
5
6
7
8
9
10
```

Enhanced for loop or for- each loop:

As of Java 5, the enhanced for loop was introduced. This is mainly used for Arrays.

- ✓ The for-each loop is used to traverse array or collection in java.
- ✓ It is easier to use than simple for loop because we don't need to increment value and usesubscript notation.
- ✓ It works on elements basis not index.
- ✓ It returns element one by one in the defined variable.

Syntax:

```
for(declaration : expression)
{
   //Statements
}
```

- **Declaration:** The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.
- **Expression:** This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Example:

```
public class Test {


public static void main(String args[])
{
   int [] numbers = {10, 20, 30, 40, 50};

   for(int x : numbers )
   {
      System.out.print( x );
      System.out.print(",");
   }
   System.out.print("\n\n");
   String [] names ={"B", "C", "C++", "JAVA"};
   for( String name : names )
   {
      System.out.print( name );
      System.out.print(",");
   }
   }
   }
```

Output:

```
10,20,30,40,50,
B,C,C++,JAVA
```

3. Transfer Statements / Loop Control Statements/Jump Statements)

1. break statement
2. continue statement

1. Using break Statement:

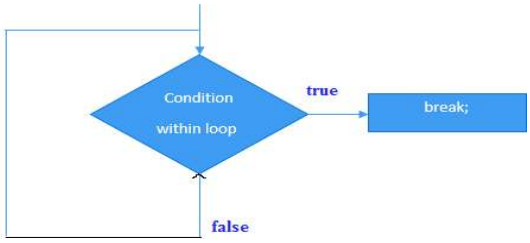
- ✓ The *break* keyword is used to stop the entire loop. The break keyword must beused inside any loop or a switch statement.
- ✓ The break keyword will stop the execution of the innermost loop and start executingthe next line of code after the block.

Syntax:

The syntax of a break is a single statement inside any loop:

```
break;
```

Flowchart:



Example:

```
public class Test {

public static void main(String args[]) { int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ) { if( x == 30 ) { break;
}
System.out.print( x ); System.out.print("\n");
}
}
}
```

Output:

10
20

2. Using continue Statement:

- The *continue* keyword can be used in any of the loop control structures. It causes theloop to immediately jump to the next iteration of the loop.
- **The Java *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.**

Syntax:

The syntax of a continue is a single statement inside any loop:

continue;

Example:

```
public class Test {

  public static void main(String args[]) {int
    [] numbers = {10, 20, 30, 40, 50};

    for(int x : numbers )
    {
      if( x == 30 )
      {
        continue;
      }
      System.out.print( x );
      System.out.print("\n");
    }
  }
}
```

Output:

10
20
40
50

1.10: DEFINING CLASSES and OBJECTS

A class is a collection of similar objects and it contains data and methods that operate on that data. In other words **-Class is a blueprint or template for a set of objects that share a common structure and a common behavior**.

DEFINING A CLASS:

The keyword **class** is used to define a class.

Rules to be followed:

1. Classes must be enclosed in parentheses.
2. The class name, superclass name, instance variables and method names may be any validJava identifiers.
3. The instance variable declaration and the statements of the methods must end with ;(semicolon).
4. The keyword **extends** means derived from i.e. the class to the left of the **extends**
   (subclass) is derived from the class to the right of the **extends** (superclass).

Syntax to declare a class:

```
[public|abstract|final] class class_name [extends superclass_name implements interface_name]
{
        data_type instance_variable1;
        data_type instance_variable2;
        .
        .
        data_type instance_variableN;

        return_type method_name1(parameter list)
        {
        Body of the method
        }
        .
        .
        return_type method_nameN(parameter list)
        {
        Body of the method
        }
}
```

- ✓ The data, or variables, defined within a **class** are called *instance variables*.
- ✓ The code to do operations is contained within *methods*.
- ✓ Collectively, the methods and variables defined within a class are called *members* of

theclass.

- ✓ Variables defined within a class are called instance variables because each instance of theclass (that is, each object of the class) contains its own copy of these variables.
- ✓ Thus, the data for one object is separate and unique from the data for another.

✓ **Example:**

```
class box {
  double width;
  double height;
  double depth;
  void volume()
 {
            System.out.println(–\n Volume is : –);
            Systme.out.println(width*height*depth);
  }
}
```

Program Explanation:

Class : keyword that initiates a class definition
Box : class name
Double : primitive data type
Height, depth, width: Instance variables
Void : return type of the method
Volume() : method name that has no parameters

DEFINING OBJECTS

An **Object** is an instance of a class. It is a blending of methods and data.

Object = Data + Methods

- It is a structured set of data with a set of operations for manipulating that data.
- The methods are the only gateway to access the data. In other words, the methods and dataare grouped together and placed in a container called Object.

Characteristics of an object:

An object has three characteristics:

1) **State:** represents data (value) of an object.
2) **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
3) **Identity:** Object identity is an unique ID 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.

CREATING OBJECTS:

Obtaining objects of a class is a two-step process:

1. Declare a variable of the class type – this variable does not define an object. Instead,it is simply a variable that can refer to an object.
2. Use **new** operator to create the physical copy of the object and assign the referenceto the declared variable.

NOTE: The **new** operator dynamically allocates memory for an object and returns a referenceto it. This reference is the address in memory of the object allocated by **new.**

Advantage of using new operator: A program can create as many as objects it needs duringthe execution of the program.

Syntax:

```
class_name object_name = new class_name();
     (or)
class_name object_name;
object_name = new class_name();
```

Example:

box b1=new box();(or)
box b2; b2=new box();

ACCESSING CLASS MEMBERS:

- ✓ Accessing the class members means accessing instance variable and instance methods in a class.
- ✓ To access these members, a dot (.) operator is used along with the objects.

Syntax for accessing the instance members and methods:

```
object_name.variable_name;
object_name.method_name(parameter_list);
```

Example:

```
class box
{
      double width;
      double height;
```

```
        double depth;
void volume()
{
        System.out.print("\n Box Volume is : ");
        System.out.println(width*height*depth+" cu.cms");
}
}
public class BoxVolume
{
public static void main(String[] args)
{
     box b1=new box(); // creating object of type box
     b1.width=10.00;     // Accessing instance variables through object
     b1.height=10.00;
     b1.depth=10.00;
     b1.volume();          // Accessing method through object
}
}
```

Output:

Box Volume is: 1000.0 cu.cms

1.11: METHODS

DEFINITION :

A Java method is a collection of statements that are grouped together to perform an operation.

Syntax: Method:

```
modifier Return –type method_name(parameter_list) throws exception_list
   {
// method body
    }
```

The syntax shown above includes:

- **modifier:** It defines the access type of the method and it is optional to use.
- **returnType:** Method may return a value.
- **Method_name:** This is the method name. The method signature consists of the methodname 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.

Example:

This method takes two parameters num1 and num2 and returns the maximum between the two:

```
        /** the snippet returns the minimum between two numbers */
        public static int minFunction(int n1, int n2)
        {
int min;
if (n1 > n2)
    min = n2;
else
    min = n1;
return min;
        }
```

- **METHOD CALLING (Example for Method that takes parameters and returning value):**
  - ✓ For using a method, it should be called.

- ✓ A method may take any no. of arguments.
- ✓ A *parameter* is a variable defined by a method that receives a value when the method is called. For example, in **square( )**, **i** is a parameter.
- ✓ An *argument* is a value that is passed to a method when it is invoked. For example, **square(100)** passes 100 as an argument. Inside **square( )**, the parameter **i** receives that value.
- ✓ There are two ways in which a method is called.
- calling a method that returns a value or
- calling a method returning nothing (no return value).
- ✓ The process of method calling is simple. When a program invokes a method, the programcontrol gets transferred to the called method.
- ✓ This called method then returns control to the caller in two conditions, when:

1. return statement is executed.
2. reaches the method ending closing brace.

- ✓ **Example:**

Following is the example to demonstrate how to define a method and how to call it:

```
public class ExampleMinNumber
{
public static void main(String[] args)
{
int a = 11;
int b = 6;
int c = minFunction(a, b);
System.out.println("Minimum Value = " + c);
}

/** returns the minimum of two numbers */
public static int minFunction(int n1, int n2)
{
    int min;
        if (n1 > n2)
            min = n2;
    else
        min = n1;
    return min;
}
}
```

This would produce the following result:

```
Minimum value = 6
```

1.12: CONTRUCTORS

Definition:

Constructor is a **special type of method** that is used to initialize the object. Constructor is **invoked at the time of object creation.** Once defined, the constructor is automatically called immediately after the object is created, before the **new** operator completes.

- **Rules for creating constructor:**

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type
3. Constructors can be declared public or private (for a Singleton)
4. Constructors can have no-arguments, some arguments and var-args;
5. A constructor is always called with the **new** operator
6. The default constructor is a no-arguments one;
7. If you don't write ANY constructor, the compiler will generate the default one;
8. Constructors CAN'T be **static**, **final** or **abstract**;
9. When overloading constructors (defining methods with the same name but with different arguments lists) you must define them with different arguments lists (as number or as type)

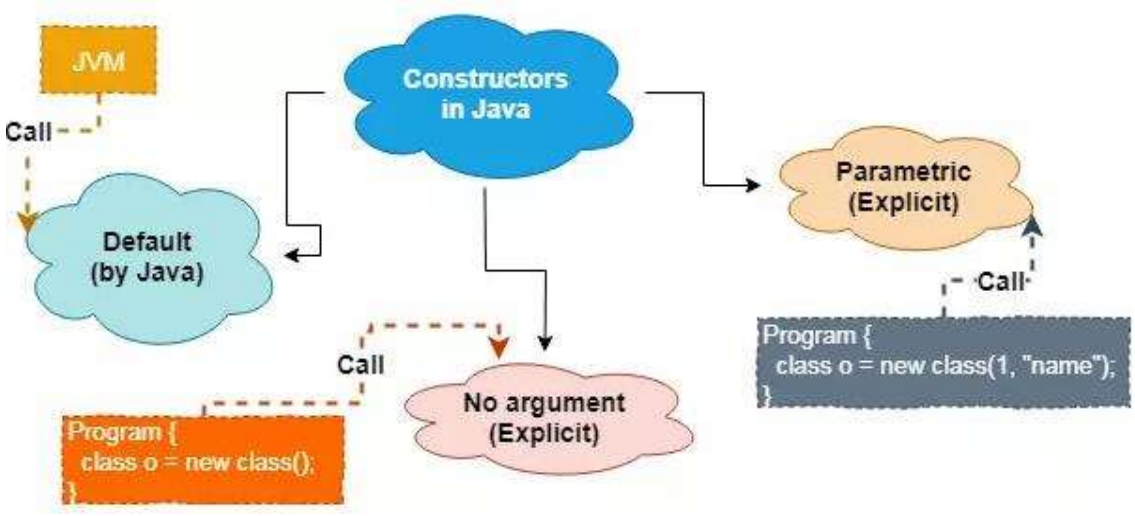
- **What happens when a constructor is called?**

1. All data fields are initialized to their default value (0, false or null).
2. All field initializers and initialization blocks are executed, in the order in which theyoccur in the class declaration.
3. If the first line of the constructor calls a second constructor, then the body of thesecond constructor is executed.
4. The body of the constructor is executed.

- **Types of constructors**

There are two types of constructors:

1. Default constructor
2. no-arg constructor
3. Parameterized constructor



1. Default Constructor

- **Default constructor refers to a constructor that is automatically created by compilerin the absence of explicit constructors.**

Rule: If there is no constructor in a class, compiler automaticaaly creates a default constructor.

Purpose of Default Constructor: It is used to provide the default values to the object members like 0, null etc. depending on the data type.

Example:

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

Output:

```
0 null
1   null
```

2) No-Argument Constructor

- **Constructor without parameters is called no-argument constructor.**

Purpose of No-Arg Constructor: It is used to provide values to be common for all objects of the class.

Syntax of default constructor:

```
Classname()
{
    // Constructor body
}
```

Example:

```
class Box
{
  double width;
  double height;
  double depth;

  // This is the constructor for Box
  Box()
  {
        System.out.println("Constructing Box…");
        width=10;
        height=10;
        depth=10;
  }
// Compute and return volume
 double volume()
 {
          return width*height*depth;
 }
}
class BoxDemo
{
        public static void main(String arg[])
        {
                // declare, allocate and initialize Box objects
```

```
        Box mybox1=new Box();
        Box mybox2=new Box();
        double vol;

        // Get volume of first box
        vol=mybox1.volume();
        System.out.println("Volume is " +vol);

        // Get volume of second box
        vol=mybox2.volume();
        System.out.println("Volume is "+vol);
    }
}
```

Output:

```
Constructing Box
Constructing Box
Volume is 1000.0
Volume is 1000.0
```

As you can see, both **mybox1** and **mybox2** were initialized by the **Box( )** constructor when they were created. Since the constructor gives all boxes the same dimensions, 10 by 10 by 10, both **mybox1** and **mybox2** will have the same volume.

3 Parameterized Constructor

A constructor that takes parameters is known as parameterized constructor.

Purpose of parameterized constructor

Parameterized constructor is used to provide different values to the distinct objects.

Example:

```
class Box
{
  double width;
  double height;
  double depth;

  // This is the constructor for Box
```

```
  Box(double w, double h, double d)
  {
        width=w;height=h;depth=d;
  }

// Compute and return volume
  double volume()
  {
 return width*height*depth;
  }
}
class BoxDemo
{
public static void main(String arg[])
{
        // declare, allocate and initialize Box objects
        Box mybox1=new Box(10,20,15);
        Box mybox2=new Box(3,6,9);
        double vol;
        // Get volume of first box

        vol=mybox1.volume();
        System.out.println("Volume is " +vol);

        // Get volume of second box
        vol=mybox2.volume();
        System.out.println("Volume is " +vol);
}
}
```

Output:

Volume is 3000.0
Volume is 162.0

As you can see, each object is initialized as specified in the parameters to its constructor. Forexample, in the following line,

Box mybox1 = new Box(10, 20, 15);

the values 10, 20, and 15 are passed to the **Box( )** constructor when **new** creates the object. Thus,

mybox1's copy of **width**, **height**, and **depth** will contain the values 10, 20, and 15, respectively.

Difference between constructor and method:

There are many differences between constructors and methods. They are given below

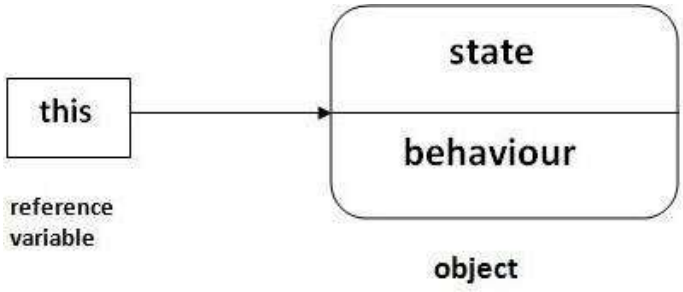
Constructor	Method
Constructor is used to initialize the state of anobject.	Method is used to expose behaviour of anobject.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in anycase.
Constructor name must be same as the classname.	Method name may or may not be same asclass name.

"this" KEYWORD:

Definition:

In java, **this** is a reference variable that refers to the current object.

- **Usage of this keyword**
  1. this keyword can be used to refer current class instance variable.
  2. this() can be used to invoke current class constructor.
  3. this keyword can be used to invoke current class method (implicitly)
  4. this can be passed as an argument in the method call.
  5. this can be passed as argument in the constructor call.
  6. this keyword can also be used to return the current class instance.



Instance Variable Hiding:

It is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes.

We can also have local variables, which overlap with the names of the class' instance

variables.

However, **when a local variable has the same name as an instance variable, the local variable *hides* the instance variable.**

We can use **"this"** keyword to to resolve any namespace collisions that might occur betweeninstance variables and local variables.

Example:

```
class Student
{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee)
{
        this.rollno=rollno;
        this.name=name;
        this.fee=fee;
}
void display()
{
        System.out.println(rollno+" "+name+" "+fee);
}
}

class TestThis2
{
public static void main(String args[])
{
        Student s1=new Student(111,"ankit",5000f);
        Student s2=new Student(112,"sumit",6000f);
        s1.display();
        s2.display();
}
}
```

Output:

```
111 ankit 5000
112 sumit 6000
```

CONSTRUCTOR OVERLOADING:

Definition:

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading:

```
class Box
{
        double width;
        double height;
        double depth;

        // constructor used when all the dimensions are specified
        Box(double w, double h, double d)
        {
                width=w;
                height=h;
                depth=d;
        }

        // constructor used when no dimensions are specified
        Box()
        {
                width=-1;
                height=-1;
                depth=-1;
        }

        // constructor used when cube is created
        Box(double len)
        {
                width = height = depth = len;
        }

        // Compute and return volume
    double volume()
        {
                return width*height*depth;
        }
  }
```

```
class ConsOverloadDemo
{
public static void main(String arg[])
{
    // declare, allocate and initialize Box objects
    Box mybox1=new Box(10,20,15);
    Box mybox2=new Box();
    Box mybox3=new Box(7);
    double vol;

    // Get volume of first box
    vol=mybox1.volume();
    System.out.println("Volume of Box1 is  "+vol);

    // Get volume of second box
    vol=mybox2.volume();
    System.out.println("Volume of Box2 is  "+vol);

    // Get volume of cube
    vol=mybox2.volume();
    System.out.println("Volume of Cube is  "+vol);
}
}
```

Output:

Volume of Box1 is 3000.0
Volume of Box2 is -1.0
Volume of the cube is 343.0

As we can see, the proper overloaded constructor is called based upon the parameters specifiedwhen **new** is executed.

CONSTRUCTOR CHAINING:

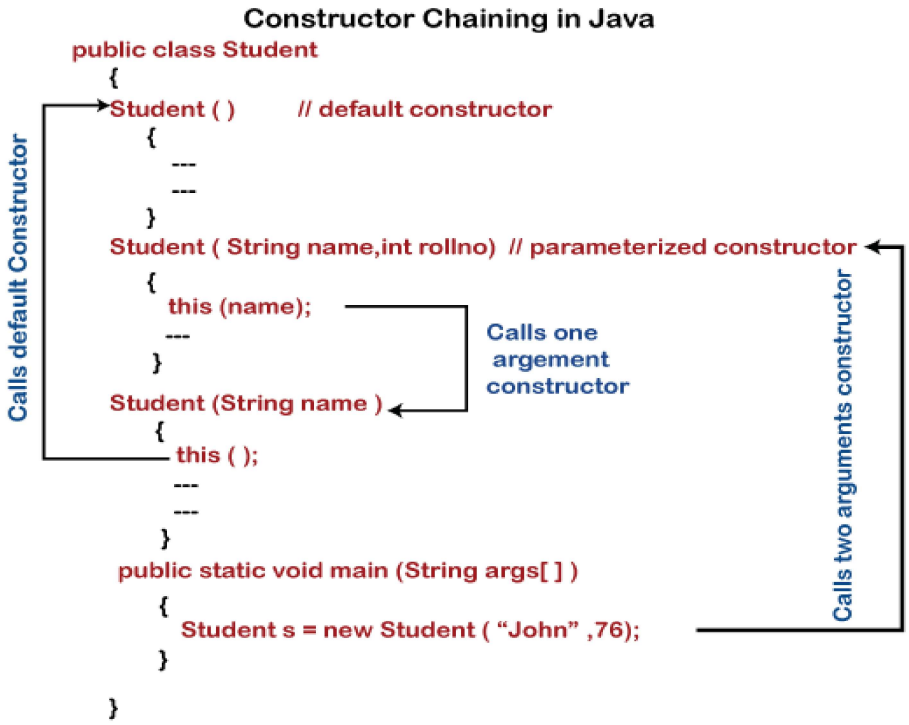
Constructor chaining is the process of calling one constructor of a class from another constructor of the same class or another class using the current object of the class.

- It occurs through inheritance.

Ways to achieve Constructor Chaining:

We can achieve constructor chaining in two ways:

- **Within the same class:** If we want to call the constructor from the same class, then we use **this** keyword.
- **From the base class:** If we want to call the constructor that belongs to different classes (parent and child classes), we use the **super** keyword to call the constructor from the base class.



Rules of Constructor Chaining:

- ✓ An expression that uses **this** keyword must be the first line of the constructor.
- ✓ Order does not matter in constructor chaining.
- ✓ There must exist at least one constructor that does not use this

Advantage:

- ✓ Avoids duplicate code while having multiple constructors.
- ✓ Makes code more readable

Example

```
class Shape
{
  int radius,length,breadth;

  Shape(int radius)
  {
    this.radius=radius;
  }
```

```
    Shape(int r,int l,int b)
    {
       this(r);
       length=l;
       breadth=b;
    }

    void areaCircle()
    {
       System.out.println("Area of Circle is "+(3.14*radius*radius));
    }
    void areaRectangle()
    {
       System.out.println("Area of Rectangle is "+(length*breadth));
    }
}
public class ConstructorChaining
{
    public static void main(String arg[])
    {
    Shape s1=new Shape(5,10,50);
    s1.areaCircle();
    s1.areaRectangle();
    }
}
```

Output:

Area of Circle is 78.5
Area of Rectangle is 500

1.13: ACCESS SPECIFIERS

Definition:

Access specifiers are used to specify the visibility and accessibility of a class constructors, member variables and methods.

Java classes, fields, constructors and methods can have one of four different accessmodifiers:

1. Public
2. Private
3. Protected
4. Default (package)

1. **Public (anything declared as public can be accessed from anywhere):**
   A variable or method declared/defined with the public modifier can be accessed anywhere in the program through its class objects, through its subclass objects andthrough the objects of classes of other packages also.

2. **Private (anything declared as private can't be seen outside of the class):**
   The instance variable or instance methods declared/initialized as private can be accessedonly by its class. Even its subclass is not able to access the private members.

3. **Protected (anything declared as protected can be accessed by classes in the samepackage and subclasses in the other packages):**
   The protected access specifier makes the instance variables and instance methods visibleto all the classes, subclasses of that package and subclasses of other packages.

4. **Default (can be accessed only by the classes in the same package):**
   The default access modifier is friendly. This is similar to public modifier except only theclasses belonging to a particular package knows the variables and methods.

	PRIVATE	DEFAULT	PROTECTED	PUBLIC
Same class	Yes	Yes	Yes	Yes
Same package Subclass	No	Yes	Yes	Yes
Same package Non-subclass	No	Yes	Yes	Yes
Different package Subclass	No	No	Yes	Yes
Different package Non-subclass	No	No	No	Yes

Example: Illustrating the visibility of access specifiers:

Z:\MyPack\FirstClass.java

```
package MyPack;

public class FirstClass
{
   public String i="I am public variable";
   protected String j="I am protected variable";
```

```
   private String k="I am private variable";
   String r="I dont have any modifier";
}
```

Z:\MyPack2\SecondClass.java

```
package MyPack2;
import MyPack.FirstClass;
class SecondClass extends FirstClass {
   void method()
   {
      System.out.println(i);    // No Error: Will print "I am public variable".
     System.out.println(j);    // No Error: Will print "I am protected variable".
     System.out.println(k);   // Error: k has private access in FirstClass
     System.out.println(r);   // Error: r is not public in FirstClass; cannot be accessed
                                //           from outside package
   }

   public static void main(String arg[])
   {
      SecondClass obj=new SecondClass();
      obj.method();
   }
}
```

Output:

```
I am public variable
I am protected variable

Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - k has private access in MyPack.FirstClass
```

1.14: "static" MEMBERS:

Static Members are data members (variables) or methods that belong to a static or non-static class rather than to the objects of the class. Hence it is not necessary to create object of that class to invoke static members.

- ✓ The static can be:
    1. variable (also known as class variable)

2. method (also known as class method)
3. block
4. nested class

❖ **Static Variable:**

- ✓ **When a member variable is declared with the static keyword, then it is called static variable and it can be accessed before any objects of its class are created, and without reference to any object.**
- ✓ **Syntax** to declare a static variable:

  [access_spefier] static data_type instance_variable;
- ✓ When a static variable is loaded in memory (static pool) it creates only a single copy of static variable and shared among all the objects of the class.
- ✓ A static variable can be accessed outside of its class directly by the class name anddoesn‘t need any object.

  Syntax : <class-name>.<variable-name>

Advantages of static variable

- ✓ It makes your program **memory efficient** (i.e., it saves memory).

❖ **Static Method:**

If a method is declared with the static keyword , then it is known as static method.

- ✓ A static method belongs to the class rather than the object of a class.
- ✓ A static method can be invoked without the need for creating an instance of a class.
- ✓ A static method can access static data member and can change the value of it.

o **Syntax: (defining static method)**

```
[access_specifier] static Return_type method_name(parameter_list)
{
  // method body
}
```

o Syntax to access static method:

```
<class-name>.<method-name>
```

✓ The most common example of a **static** member is **main( )**. **main( )** is declared as **static** because it must be called before any objects exist.

▪ Methods declared as **static** have several restrictions:

- ✓ They can only directly call other **static** methods.
- ✓ They can only directly access **static** data.
- ✓ They cannot refer to **this** or **super** in any way.

❖ **Static Block:**

Static block is used to initialize the static data member like constructors helps toinitialize instance members and it gets executed exactly once, when the class is first loaded.

- It is executed before main method at the time of class loading in JVM.

- **Syntax:**

```
class classname
{
   static
   {
      // block of statements
   }
}
```

The following example shows a class that has a **static** method, some **static** variables, and a **static**
initialization block:

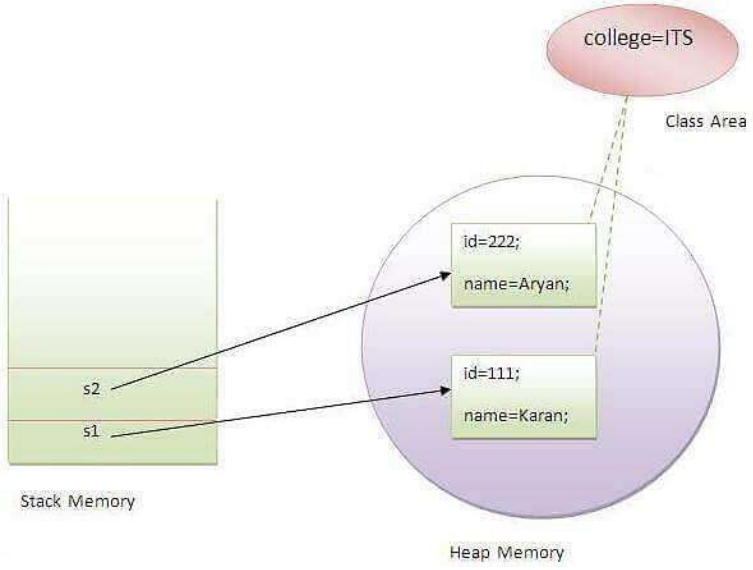
// Demonstrate static variables, methods, and blocks.

```
1.  class Student
2.  {
3.  int rollno;
4.  String name;
5.  static String college = "ITS";
6.  //static method to change the value of static variable
7.  static void change(){
8.  college = "BBDIT";
9.  }
10. //constructor to initialize the variable
11. Student(int r, String n){
12. rollno = r;
13. name = n;
14. }
15. //method to display values
16. void display()
17. {
18. System.out.println(rollno+" "+name+" "+college);
19. }
20. }
21. //Test class to create and display the values of object
22. public class TestStaticMembers
```

```
23.         {
24.         static
25.         {
26.         System.out.println(-*** STATIC MEMBERS – DEMO ***||);
27.         }
28.
29.         public static void main(String args[])
30.         {
31.         Student.change();  //calling change method
32.         //creating objects
33.         Student s1 = new Student(111,"Karan");
34.         Student s2 = new Student(222,"Aryan");
35.         Student s3 = new Student(333,"Sonoo");
36.         //calling display method
37.         s1.display();
38.         s2.display();
39.         s3.display();
40.         }
41.         }
```

Here is the output of this program:

```
*** STATIC MEMBERS – DEMO ***
111         Karan           BBDIT
222         Aryan           BBDIT
333         Sonoo           BBDIT
```



1.15: JavaDoc Comments

Definition:

Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code. Java documentation can be created as part of the source code.

Input: Java source files (.java)

- Individual source files
- Root directory of the source files

Output: HTML files documenting specification of java code

- One file for each class defined
- Package and overview files

HOW TO INSERT COMMENTS?

The javadoc utility extracts information for the following items:

- Packages
- Public classes and interfaces
- Public and protected methods
- Public and protected fields

Each comment is placed immediately above the feature it describes.

Format:

- ✓ A Javadoc comment precedes similar to a multi-line comment except that it beginswith a forward slash followed by two asterisks (/**) and ends with a */
- ✓ Each /** . . . */ documentation comment contains free-form text followed by tags.
- ✓ A tag starts with an @, such as @author or @param.
- ✓ The first sentence of the free-form text should be a summary statement.
- ✓ The javadoc utility automatically generates summary pages that extract thesesentences.
- ✓ In the free-form text, you can use HTML modifiers such as ... foremphasis, <code>...</code> for a monospaced —typewriter font, ... for strong emphasis, and even <img ...> to include an image.
- ✓ Example:

```
/**
This is a <b>doc</b> comment.
*/
```

TYPES OF COMMENTS:

1. Class Comments

The class comment must be placed *after* any import statements, directly before the classdefinition.

Example:

```
import java.io.*;
/** class comments should be written here */Public class sample
{
....
}
```

2. Method Comments

The method comments must be placed immediately before the method that it describes.

Tags used:

Tag	Description	Syntax
@param	It describes the method parameter	**@param name description**
@return	This tag describes the return value from a method with the exception void methods and constructors.	**@return description**
@throws	This tag describes the method that throws an exception.	**@throws class description**

Example:

```
/** adding two numbers
@param a & b are two numbers to be added
@return the result of addition
**/
public double add(int a,int b)
{
int c=a+b;
return c;
}
```

3. ***Field Comments***

Field comments are used to document public fields—generally that means static constants.

For example:

```
/**
* Account number
*/
public static final int acc_no = 101;
```

4. ***General Comments***

Tag	Description	Syntax
The following tags can be used in class documentation comments		
@author	This tag makes an ―author entry. You can have multiple @author tags, one for each author.	**@author name**
@version	This tag makes a ―version‖ entry. The text can be any description of the current version.	**@version text**
The following tags can be used in all documentation comments		
@since	This tag makes a ―since‖ entry. The text can be any description of the version that introduced thisfeature. For example, @since version 1.7.1	**@since text**
@deprecated	This tag adds a comment that the class, method, or variable should no longer be used. The text should suggest a replacement. For example: @deprecated Use <code>setVisible(true)</code>instead	**@deprecated text**
Hyperlinks to other relevant parts of the javadoc documentation, or to external documents,with the @see and @link tags.		
@link	This tag place hyperlinks to other classes or methods anywhere in any of your documentationcomments.	**{@link package.class#feature label}**

@see	This tag adds a hyperlink in the —see also section. It can be used with both classes and methods. Here, reference can be one of the following: package.class#feature label label "text" Example: @see -Core java 2 @see Core Java	**@see reference**

COMMENT EXTRACTION

Here, *docDirectory* is the name of the directory where you want the HTML files to go.

Follow these steps:

1. Change to the directory that contains the source files you want to document.
2. To create the document API, you need to use the javadoc tool followed by java file name.There is no need to compile the javafile.

Here, docDirectory is the name of the directory where you want the HTML files to go.

Follow these steps:

1. Change to the directory that contains the source files you want to document.
2. Run the command

javadoc -d docDirectory nameOfPackage

for a single package. Or run

javadoc -d docDirectory nameOfPackage1 nameOfPackage2...

to document multiple packages.

If your files are in the default package, then instead run

javadoc -d docDirectory *.java

If you omit the -d docDirectory option, then the HTML files are extracted to the currentdirectory.

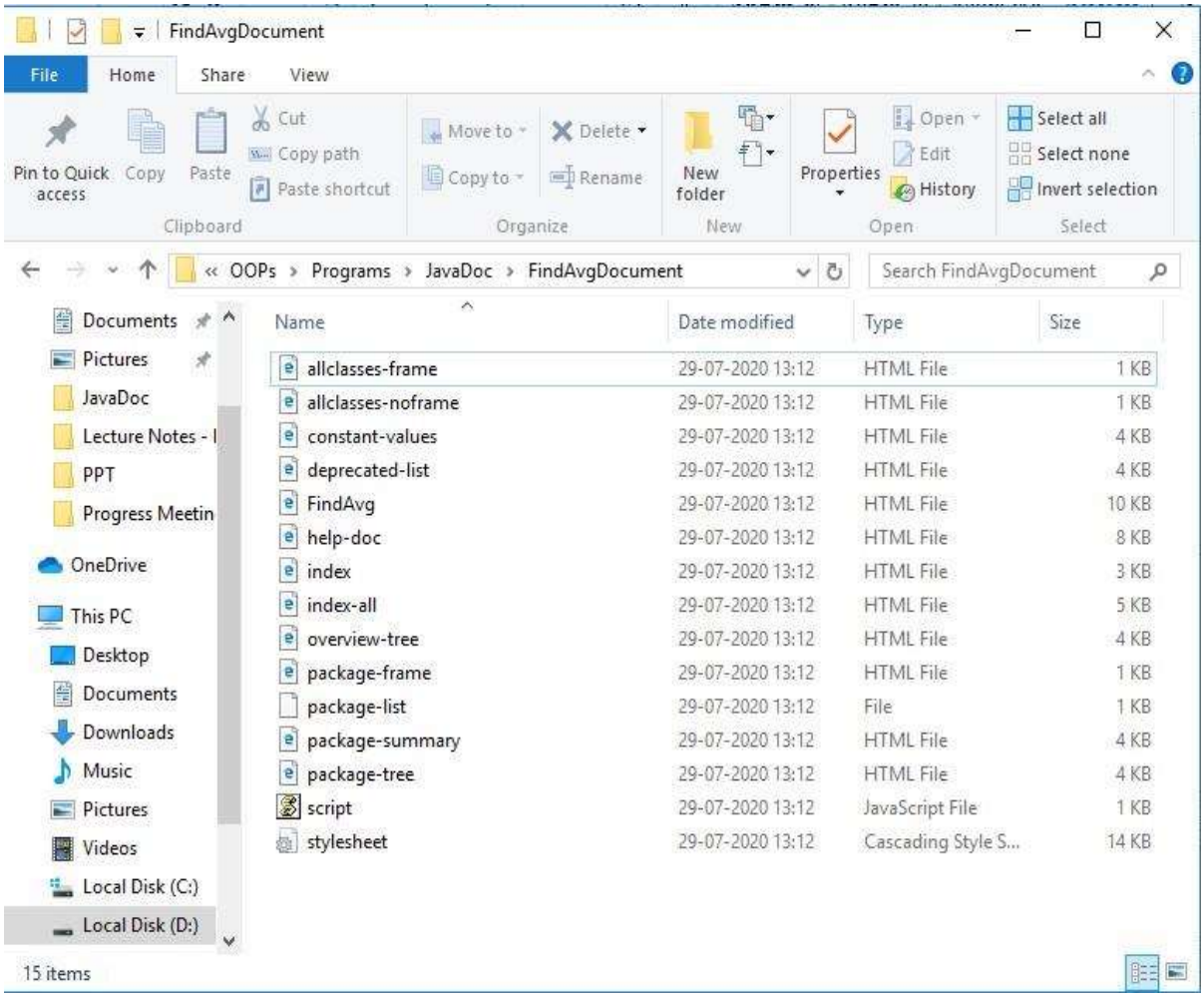
Example:

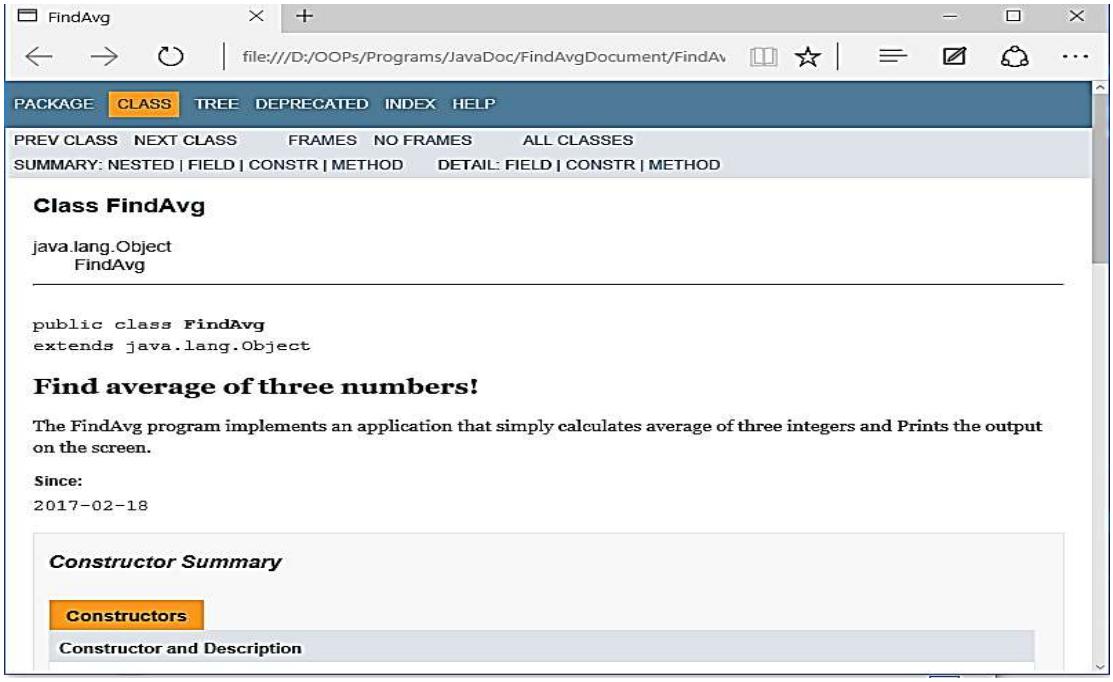
```
//Java program to illustrate frequently used
// Comment tags

/**
* <h1>Find average of three numbers!</h1>
* The FindAvg program implements an application that
* simply calculates average of three integers and Prints
* the output on the screen.
*
* @author  Pratik Agarwal
* @version 1.0
* @since  2017-02-18
*/
public class FindAvg
{
   /**
   * This method is used to find average of three integers.
   * @param numA This is the first parameter to findAvg method
   * @param numB  This is the second parameter to findAvg method
   * @param numC  This is the third parameter to findAvg method
   * @return int This returns average of numA, numB and numC.
   */
   public int findAvg(int numA, int numB, int numC)
   {
      return (numA + numB + numC)/3;
   }

   /**
   * This is the main method which makes use of findAvg method.
   * @param args Unused.
   * @return Nothing.
   */

   public static void main(String args[])
   {
      FindAvg obj = new FindAvg();
      int avg = obj.findAvg(10, 20, 30);

      System.out.println("Average of 10, 20 and 30 is :" + avg);
   }
}
```

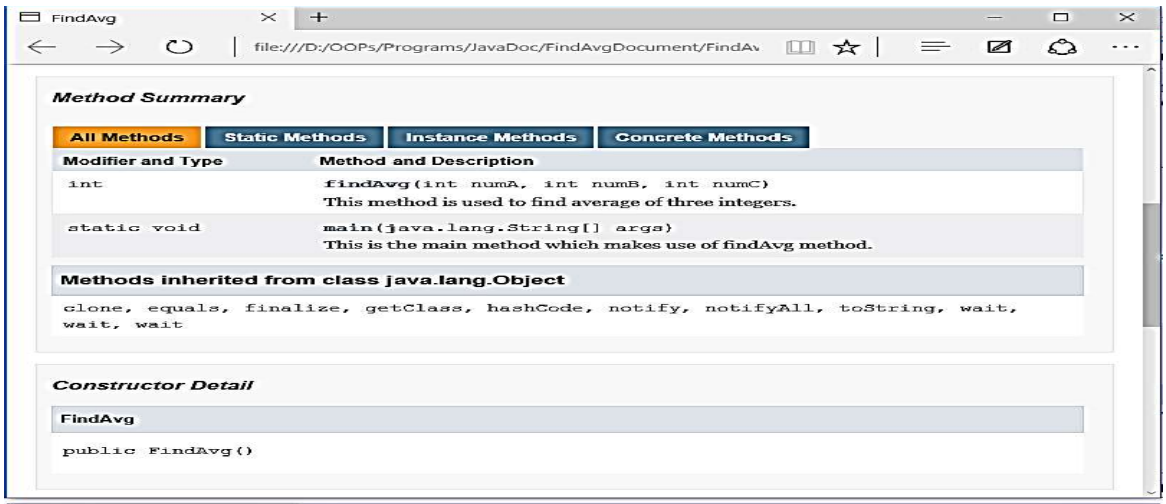
OUTPUT:

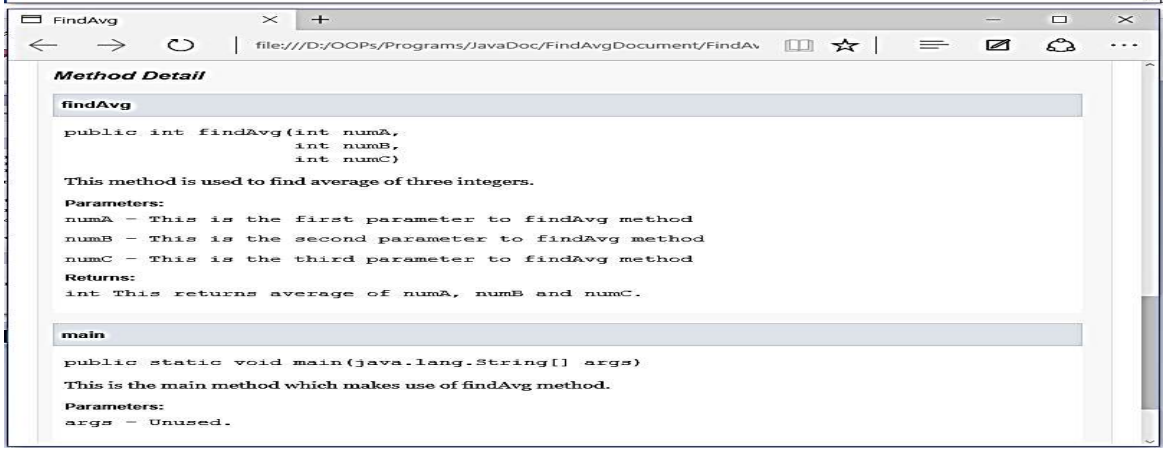
D:\OOPs\Programs\JavaDoc>javadoc -d FindAvgDocument FindAvg.java
Loading source file FindAvg.java...
Constructing Javadoc information...
Creating destination directory: "FindAvgDocument\"
Standard Doclet version 1.8.0_251

```
Building tree for all the packages and classes...
Generating FindAvgDocument\FindAvg.html...
FindAvg.java:32: error: invalid use of @return
    * @return Nothing.
      ^
Generating FindAvgDocument\package-frame.html...
Generating FindAvgDocument\package-summary.html...
Generating FindAvgDocument\package-tree.html...
Generating FindAvgDocument\constant-values.html...
Building index for all the packages and classes...
Generating FindAvgDocument\overview-tree.html...
Generating FindAvgDocument\index-all.html...
Generating FindAvgDocument\deprecated-list.html...
Building index for all classes...
Generating FindAvgDocument\allclasses-frame.html...
Generating FindAvgDocument\allclasses-noframe.html...
Generating FindAvgDocument\index.html...
Generating FindAvgDocument\help-doc.html...
1 error
```




FindAvg
file:///D:/OOPs/Programs/JavaDoc/FindAvgDocument/FindAv
PACKAGE CLASS TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
Class FindAvg
java.lang.Object
FindAvg
public class FindAvg
extends java.lang.Object
Find average of three numbers!
The FindAvg program implements an application that simply calculates average of three integers and Prints the output on the screen.
Since:
2017-02-18
Constructor Summary
Constructors
Constructor and Description


FindAvg
file:///D:/OOPs/Programs/JavaDoc/FindAvgDocument/FindAv
Method Summary
All Methods Static Methods Instance Methods Concrete Methods
Modifier and Type Method and Description
int findAvg(int numA, int numB, int numC)
This method is used to find average of three integers.
static void main(java.lang.String[] args)
This is the main method which makes use of findAvg method.
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Constructor Detail
FindAvg
public FindAvg()


FindAvg
file:///D:/OOPs/Programs/JavaDoc/FindAvgDocument/FindAv
Method Detail
findAvg
public int findAvg(int numA,
int numB,
int numC)
This method is used to find average of three integers.
Parameters:
numA - This is the first parameter to findAvg method
numB - This is the second parameter to findAvg method
numC - This is the third parameter to findAvg method
Returns:
int This returns average of numA, numB and numC.
main
public static void main(java.lang.String[] args)
This is the main method which makes use of findAvg method.
Parameters:
args - Unused.

1.16: Additional Topics
Comments, Literals, Keywords, Type Conversion, Garbage Collection, Command Line Arguments

1.16.1: JAVA – COMMENTS

- Java comments are either explanations of the source code or descriptions of classes,interfaces, methods, and fields.
- They are usually a couple of lines written above or beside Java code to clarify what itdoes.
- Comments in Java do not show up in the executable program.
- The Java language supports three kinds of comments:

1. Line comment:

- ✓ When you want to make a one line comment type "//" and follow the two forward slasheswith your comment.
- ✓ **Syntax:** // text
- ✓ **Example:** // this is a single line comment
- ✓ The compiler ignores everything from // to the end of the line.

2. Block Comment:

- ✓ To start a block comment type "/*". Everything between the forward slash and asterisk, even if it's on a different line, will be treated as comment until the characters "*/" end the comment.
- ✓ **Syntax:** /* *text* */
- ✓ **Example:** /* it is a comment */ (or)

```
/* thisis a block
comment
*/
```

- ✓ The compiler ignores everything from /* to */.

3. Documentation Comment:

- ✓ This type of comment helps is generating the documentation automatically.
- ✓ **Syntax:** /** *documentation* */

The JDK javadoc tool uses doc comments when preparing automatically generated documentation. For more information on javadoc, see the Java tool documentation.

- ✓ **Example:**

```
/*
 * Title: Conversion of Degrees
 * Aim: To convert Celsius to Fahrenheit and vice versa
 * Date: 31/08/2000
 * Author: Tim
*/
```

1.16.2: JAVA - CONSTANTS

- ✓ A constant is an identifier written in uppercase (convention and not a rule) that prevents
  its contents form being modified by the program during the execution.
- ✓ If an attempt is made to change the value, the compiler will give an error message.
- ✓ In Java, the keyword **final** is used to declare constants.
- ✓ The value of a final variable cannot change after it has been initialized.

final datatype variablename=value;

- ✓ **Syntax:**
- ✓ **Example:**final float PI=3.14f;

1.16.3: JAVA - IDENTIFIERS

- ✓ Identifiers are names given to the variables, classes, methods, objects, labels, package and interface in our program.
- ✓ The name we are giving must be meaningful and it may have random length.
- ✓ The following rule must be followed while giving a name:
  1. The first character must not begin with a number.
  2. The identifier is formed with alphabets, number, dollar sign ($) and underscore (_).
  3. It should not be a reserved word.
  4. Space is not allowed in between the identifier name.

- ✓ **Example:**

```
String name = "Homer Jay Simpson";
int weight = 300;
double height = 6;
```

1.16.4: JAVA – RESERVED WORDS (KEYWORDS)

- ✓ There are some words that you cannot use as object or variable names in a Java program. These words are known as reserved words; they are keywords that are already used by the syntax of the Java programming language.
- ✓ For example, if you try and create a new class and name it using a reserved word:

```
// you can't use finally as it's a reserved word!
        class finally {
        public static void main(String[] args)
        {
```

```
//class code..
}
}
```

- ✓ It will not compile, instead you will get the following error: <identifier> expected
- ✓ The table below lists all the words that are reserved:

abstract	**assert**	**boolean**	**break**	**byte**	**case**
catch	**char**	**class**	const*	continue	default
double	do	else	enum	extends	false
final	**finally**	**float**	**for**	goto*	**if**
implements	**import**	instanceof	**int**	**interface**	long
native	**new**	null	**package**	private	protected
public	return	short	**static**	strictfp	**super**
switch	synchronized	**this**	**throw**	**throws**	transient
true	**try**	**void**	volatile	while	

1.16.5: TYPE CONVERSIONS AND CASTING

Type Conversion is the task of converting one data type into another data type.

Two types of type conversion:

1. Implicit Type Conversion (or) Automatic Conversion
2. Explicit Type Conversion (or) Casitng

1. Implicit Type Conversion (or) Automatic Conversion;

If the two types are compatible, then Java will perform the conversion automatically. When one type of data is assigned to another type of variable, an ***Automatic type conversion (or) Widening Conversion*** will take place if the following two conditions are met:

- Two types are compatible
- The destination type is larger than the source type

.

Example:

```
byte a=100;
int b=a;        // b is larger than a

long d=b;       // d is large than b
float e=b;      // e is larger than b

float sum=10;
int s=sum;          // s is smaller than sum, So we need to go for explicit conversion.
```

1. Explicit Type Conversion (or) Casting:

If the two types are compatible, a forced conversion of one type into another type is performed This forced conversion is called as Explicit Type Conversion. **Casting (or) narrowing conversion** is an operation which performs an explicit conversion between incompatible types.

Example: converting int to byte.

Syntax to perform "Cast":

(target-type) value;

Here,

Target-type = specifies the desired type to convert the specified value.

Example:

```
class conversion {
public static void main(String arg[])
{
byte b;
int i=257;
double d=323.142;

System.out.println("\nConversion of int to byte: ");
b=(byte) i;
System.out.println("i and b : "+i+" , "+b);

System.out.println("\nConversion of double to int: ");
i=(int) d;
System.out.println("d and i : "+d+" , "+i);

System.out.println("\nConversion of double to byte: ");
b=(byte) d;
```

```
System.out.println("d and b : "+d+" , "+b);

        // Automatic Type promotions in expressions
byte r=40;
byte s=50;
byte t=100;
int p=r * s / t;        // r*s exceeds the range of byte, so automatic type promotion take place.
System.out.println("Value of P = "+p);
s=s*2;          //Error! cannot assign int to a byte.
s=(byte)(s*2);      // Possible.
}
}
```

Output:

```
Conversion of int to byte:
i and b : 257 , 1
Conversion of int to byte:
d and i : 323.142 , 323
Conversion of int to byte:
d and b : 323.142 , 67
Value of P = 20
```

Type Promotions rules:

1. All **byte, short** and **char** values are promoted to int.
2. If one operand is **long,** the whole expression is promoted to **long.**
3. If one operand is **float,** the whole expression is promoted to **float.**
4. If any of the operand is **double,** the result is **double.**

1.16.6: GARBAGE COLLECTION

- ✓ Since objects are dynamically allocated by using the **new** operator, you might be wondering how such objects are destroyed and their memory released for later reallocation.
- ✓ In some languages, such as C++, dynamically allocated objects must be manually released by use of a **delete** operator.

- ✓ Java takes a different approach;

Automatic Garbage Collection: The technique that accomplishes automatic deallocation ofmemory occupied by an unused object is called *garbage collection*.

It works like this:

- When no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. There is no explicit need to destroy objects as in C++.
- Garbage collection only occurs sporadically (if at all) during the execution of your program.

➢ **Finalization:**

- ✓ Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some non-Java resource such as a file handle or character font, then you might want to make sure these resources are freed before an object is destroyed.
- ✓ To handle such situations, Java provides a mechanism called *finalization*. By using finalization, you can define specific actions that will occur when an object is just about tobe reclaimed by the garbage collector.

➢ **Finalize() method:**

A **finalize()** method is a method that will be called by the garbage collector on an object when garbage collection determines that there are no more references to the object.

Inside the **finalize( )** method, we will specify those actions that must be performed before anobject is destroyed.

The finalize( ) method has this general form:

```
protected void finalize( )
{
// finalization code here
}
```

Here, the keyword protected is a specifier that prevents access to finalize( ) by code definedoutside its class.

Example:

```
public class TestGarbage1
{
public void finalize()
{
```

```
System.out.println("object is garbage collected");
}
public static void main(String args[])
{
TestGarbage1 s1=new TestGarbage1();
TestGarbage1 s2=new TestGarbage1();
s1=null;
s2=null;
System.gc();
}
}
```

Output:

object is garbage collectedobject is garbage collected

1.16.7: USING COMMAND LINE ARGUMENTS:

- ✓ Sometimes you will want to pass information into a program when you run it. This isaccomplished by passing *command-line arguments* to **main( )**.

- ✓ **A command-line argument is the information passed to the main() method that directly follows the program's name on the command line when it is executed.**

- ✓ To access the command-line arguments inside a Java program is quite easy—they are stored as strings in a **String** array passed to the **args** parameter of **main( )**.
- ✓ The first command-line argument is stored at **args[0]**, the second at **args[1]**, and so on.
- ✓ For example, the following program displays all of the command-line arguments that it iscalled with:

```
// Display all command-line arguments.

class CommandLine
{
public static void main(String args[])
{
for(int i=0; i<args.length; i++)
  System.out.println("args[" + i + "]: " + args[i]);
}
}
```

Try executing this program, as shown here:

```
>java CommandLine this is a test 100 -1
```

When you do, you will see the following output:

```
args[0]: this
args[1]: is
args[2]: a
args[3]: test
args[4]: 100
args[5]: -1
```