Core
Java™
Programming

# Ch.01 History, Introduction and Language, Basics Classes And Object

## Syllabus : Ch.01

- History and Features of Java
- Java Editions
- JDK, JVM and JRE
- JDK Tools
- Compiling and Executing basic Java Program
- Java IDE (Net beans and Eclipse)
- Data Type (Integer, Float, Character, Boolean)

## Syllabus : Ch.01

- Java Tokens
  - (Keyword, Literal, Identifier, Whitespace, Separators, Comments, Operators)
- Operators
  - (Arithmetic, Relational, Boolean Logical, Bitwise Logical, Assignment, Unary, Shift, Special operators)
- Type Casting

## Syllabus : Ch.01

- Decision Statements
  - (if, switch)
- Looping Statements
  - (for, while, do..while)
- Jumping Statements
  - (break, continue, return)
- Array
  - (One Dim., Rectangular, Jagged)
- Command Line Argument Array

## Syllabus : Ch.01

- OOP Concepts
  - (Class, Object, Encapsulation, Inheritance, Polymorphism)
- Creating and using Class with members
- Constructor
- finalize() method
- Static and Non-Static Members
- Overloading
  - (Constructor & Method)
- - VarArgs

## History Of JAVA…

- Java was developed by James Gosling, Patrick Naughton and their team at the Sun Microsystems in 1991.
- At that time the language was named as 'Oak' but then it was renamed to Java in 1995.
- Java is an Object Oriented Programming Language.

## History Of JAVA…

- The term OOP can be understood by the OOP principles.
- The programming language that follows or supports the OOP principles can be known as Object Oriented Programming Language.

## Object Oriented Programming :

- In OOP your complex program is divided in module like data (member variables) and methods (functions).
- This data and methods are wrapped into a single unit which is known as class.
- Now as needed we can make objects of the class.

## OOP Principles :

- There are mainly three principles of OOP.
  - Encapsulation
  - Inheritance
  - Polymorphism

## OOP Principles : Encapsulation

- Encapsulation is a mechanism by which we can bind data (member variables) and methods together.
- It is a process of wrapping up of data and methods into a single unit.
- So by encapsulation we can achieve the concept of data hiding.

## OOP Principles : Inheritance

- Inheritance is very useful principle of Object Oriented Programming.
- It simply means to inherit something from one class to another.
- If object of class A gets the properties, data or methods of class B, then B class is called being inherited in the class A. That means the process of obtaining properties of another class to the object of one class is known as Inheritance.

## OOP Principles : Polymorphism

- Polymorphism is a Greek term, which means the ability to take more than one forms by a single entity.
- In OOP, polymorphism plays an important role.
- Polymorphism can be achieved by many ways like method overloading, constructor overloading and method overriding etc.

## OOP Principles : Polymorphism

- ***For Example,*** A single man can behave differently in different situations. Same as a single method/function performs differently depending its argument.

## Java Buzzword or Features of Java

- Java Buzzwords are features of Java. This makes the Java most striking programming language among other languages.
- Features of Java.
  - Simple
  - Secure
  - Portable
  - Object-Oriented

## Java Buzzword or Features of Java

- Features of Java.
  - Robust (Strong)
  - Multithreaded
  - Architecture Neutral (Performance Independent)
  - Interpreted
  - High Performance
  - Distributed
  - Dynamic

## Java Buzzword or Features of Java

- Simple
  - Learning Java is said to be simple if you have some basic knowledge of C / C++.
  - Java adopts many syntax of C language and if one understand the concept of Object Oriented Programming, then java will be much easier.
  - Many syntax like comments, declaring variables, etc are same as C and C++.

## Java Buzzword or Features of Java

- Secure
  - Security is the most important need for any programming language.
  - While downloading a file from Internet there is a big risk of viral infection.
  - In case of Java there is no risk of it. Because Java provides a ***Firewall*** between your computer and the application from which the computer downloads files.
  - Java does not allow access to other parts of your program.

## Java Buzzword or Features of Java

- Portable
  - A Java program is fully portable.
  - To make the program portable, Java compiler generates executable code known as Byte code. Byte code then interpreted via Java Virtual Machine (JVM).
  - This Byte code is in machine language code which every machine can read. This is main benefit of language portability.

## Java Buzzword or Features of Java

- Object Oriented
  - Java is a pure Object Oriented Programming language.
  - In OOP language the main focus is on the data (member variables) and methods.
  - Everything to be processed is put in the methods using data or member variables.

## Java Buzzword or Features of Java

- Robust (Strong)
  - Java is said to be robust (strong) language.
  - There are two main reasons that makes it very robust :
    - Memory Management and
    - Exception Handling

## Java Buzzword or Features of Java

- Robust (Strong)
  - Memory Management
    - Memory Management is much easier in Java as it automatically deallocates (free) memory through garbage collection.

# Java Buzzword or Features of Java

- Robust (Strong)
  - Exception Handling
    - Exception Handling in Java helps programmer to write efficient programs.
    - Divide by zero error, File not found exceptions etc. are errors that often arise during programming. Java has its own Exception Handler or you can manually handle exception also.

## Java Buzzword or Features of Java

- Multithreaded
  - A thread is a sub (child) process.
  - Using multithreading you can execute more than one thread simultaneously.
  - Java has a Thread class and its methods to accomplish multithreading. This feature makes no bargaining in Java’s Platform Independent feature.

## Java Buzzword or Features of Java

- Architecture Neutral (Platform Independent)
  - In Java the source code is first compiled by java compiler and then the compiler generates Bytecode.
  - The Bytecode is in machine readable form so that every machine can understand it. Then the Bytecode interpreted using Java Virtual Machine.
  - Therefore the java program written on any platform can be run on any other platform.
  - This features makes java Architecture Neutral (તટસ્થ) or Platform Independent.

## Java Buzzword or Features of Java

- Interpreted
  - The execution of Java program is a two step process.
  - First the source code is Compiled using Java compiler and generated Bytecode.
  - The Bytecode is machine independent which is interpreted using Java Virtual Machine. And give output.

## Java Buzzword or Features of Java

- High Performance
  - This feature is the result of above two features.
  - As Java generates Bytecode which is machine independent and are interpreted by JVM (Java Virtual Machine) there is a great performance arise.

## Java Buzzword or Features of Java

- Distributed
  - Java is a very good in Networked programming like client / server architecture.
  - A method on one PC can be invoked by the object on another PC. This is done by Remote Method Invocation (RMI).
  - By this Java can be used in distributed environment.

# Java Buzzword or Features of Java

- Dynamic
  - Java is far more dynamic language than C or C++.
  - In java Libraries can add new methods and variables as needed.
  - Here, we can get information of variables easily.
  - This is useful when you want to add or update code while downloading programs from Internet.

## JDK and its components :

- The Java Development Kit (JDK) is the installation package of java.
- It has a collection of components that are used in development of Java program. The JDK components are given below..

| JDK Tool | Meaning | Use |
|---|---|---|
| **javac** | Java Compiler | Compiles the source code **(.java file)** and translates it into Bytecode. |

## JDK and its components :

| JDK Tool | Meaning | Use |
|---|---|---|
| **java** | Java Interpreter | Interprets Bytecode and generate output. |
| **javadoc** | Java documentation | Used to create documentation for Java source code in HTML format. |
| **javah** | Java Header files | It produces header files for the program which uses native methods. |

## JDK and its components :

| JDK Tool | Meaning | Use |
|---|---|---|
| **javap** | Java dissembler | It converts Bytecode file **(.class file)** to program description |
| **jdb** | Java Debugger | Used to debug your program. |
| **Applet viewer** | Applet Viewer | Used to execute Java Applets |

## Application V/s Applet

- Application :
  - Java Application is a normal program which is run on your machine under the control of operating system.
- Applet :
  - Java applet is a program which is run from the web browser or it can be run by appletviewer.

## How to run a Java Program?

- Step-1
  - Download JDK using ***www.google.com***
- Step-2
  - Install JDK
- Step-3
  - ***Goto cmd***, and check path using **PATH** command. If PATH has java then ok.
  - ***Else*** Set Path=C:\Program Files\Java\jdk-15.0.2\bin

## How to run a Java Program?

- Step-4
  - Use any text editor like, **Notepad** and code you JAVA program.
  - Save that program file with, **<ClassName>.JAVA** extension.
- Step-5
  - Compile that program on Command prompt >**javac Test.JAVA**

## How to run a Java Program?

- Step-6
  - Run that program on command prompt.

  C:\>Java <**Java ByteCode File**>

  **C:\>** Java **Test**

## System.out.print()

- Syntax :

  System.out.print()

- Purpose :
  - To print given string or variable we can use **System.out.print().**
- **Example :**

  **System.out.print(**“MONARCH Lathi”**);**

## System.out.print()

- System :
  - System is a class that provides access to the system.
- out :
  - It is the output stream that is used to send data to the console (output screen).
- print()
  - Prints the string passed as argument.

## Def.1 WAPto print a Welcome Message.

```
// System.out.print > to print on same line
class test
{
  public static void main(String args[])
  {
    System.out.print("WelCome To JAVA");
  }
}
```

## Process to RUN Java program…

- Type the program using editor like notepad.
- Save the file by name **test.java** (We have to save the file by classname.java)
- Compile the file using **javac** commad.
- It there is no error then after successful compilation a **test.java** will create a new file test.class file.
- Run the file using >**Java Test**

## Understanding **main( )** method :

public static void **main(**String args[]**)**

- **Public**
  - It is access specifier which is public.
  - So that main() method can accessed from any class.
- **Static**
  - Static keyword specifies that main() method is called without having to create instance (object) of its class.

## Understanding main( ) method :

- **Void :**
  - It is the return type of main() method. Which specifies that main() method doesn’t return any value.
- **main()**
  - main() method is the entry point of your program. From main() the execution of program starts.

## Understanding main( ) method :

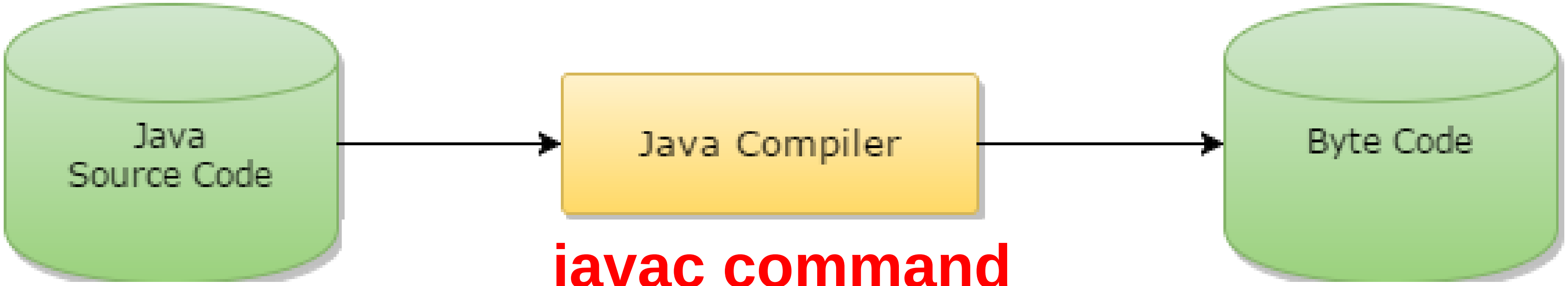
- **String args[ ]**
  - main() method can take array of String type values as argument, which can be supplied as command-line argument.

## Def.2 WAPto print Your Name and Address…

```
// System.out.println > to print on NEW line
class test
{ public static void main(String args[])
  {  System.out.println("Gohil M. A.");
     System.out.println("MONARCH Sankul");
     System.out.println("Lathi, Di. Amreli");
  }
}
```

## Java Virtual Machine (JVM)

- Java Virtual Machine (JVM) is the interpreter for the Bytecode generated from the source code. The execution of java program is two step process.
- **Step-1**

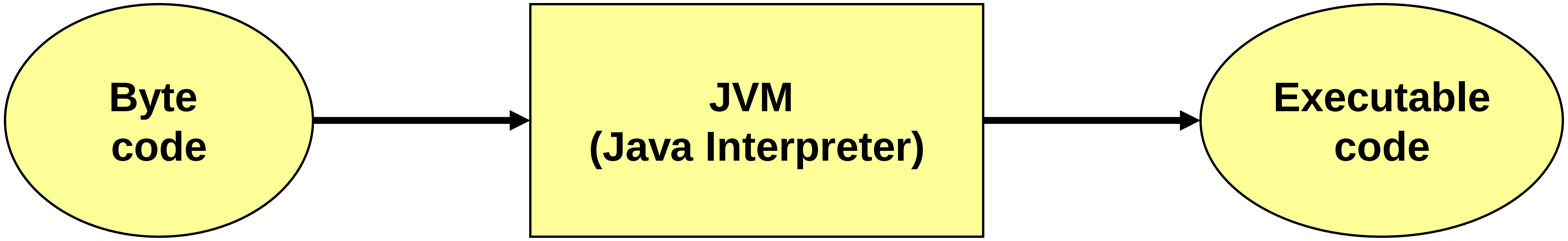


**javac command**

**[ .java file ]** **[ .class file ]**

## Java Virtual Machine (JVM)

- **Step-2**



**[ .class file ]**

**JVM**
**(Java Interpreter)**

**[ output ]**

## Java Virtual Machine (JVM)

- The source file (.java file) is Compiled by javac compiler and Bytecode is generated if no errors are present. Bytecode is .class file.
- This Bytecode is then interpreted by JVM and the output is generated.

## Basic Structure of Java Program :

- There may be more than one class in a java program. But you have to run that class which has the main method.
- Basic structure of a Java Program is as under :

  (1) Documentation Section

  (2) Package Statement

  (3) Import Statement

  (4) Class definition

  (5) Main method

## Basic Structure of Java Program :

- Documentation Section :
  - In this section we can write the documentation for our program like,
    - Program definition,
    - Author information,
    - The logic of your program etc.
  - This is optional statement.
  - It is written in comment lines **(//)** or **(/*……..*/)**

## Basic Structure of Java Program :

- Package statement :
  - If you want to save your program in a package (directory) then the package statement is included.
  - This is also an optional statement.

## Basic Structure of Java Program :

- Import statement :
  - If you want to import any library or user defined package, the import statement is used.
  - This is an optional section.
  - Now from this imported package you can use its classes and methods.
  - **Example :**

```
import java.util.Date;
```

## Basic Structure of Java Program :

- Class definition
  - You can define classes in this section.
  - Use **class** keyword to declare a class.
  - There can be member variables and methods in this class.
  - This is required and it is compulsory section.

## Basic Structure of Java Program :

- Main method
  - In your program you can have more than one class, but there should be one class that contains the main method.
  - This class is run to generate output because the main method is the entry point of program execution.
  - This is compulsory section.

## Java IDE (Integrated Development Environment) (NetBeans and Eclipse) :

- NetBeans and Eclipse are two most popular IDEs (Integrated Development Environment) in which you can develop, debug and deploy your program easily.
- These editors provide so many interesting features to develop programs in many languages including Java, PHP, C, C++ etc.

## Java IDE (Integrated Development Environment) (NetBeans and Eclipse) :

- NetBeans :
  - NetBeans is an open source editor with lots of features to develop any kind of complex project.
  - It lets you develop Java desktop, web and mobile applications, while also providing great tools for PHP and C/C++ developers very quickly and easily.

## Java IDE (Integrated Development Environment) (NetBeans and Eclipse) :

- NetBeans :
  - It is totally free and open source with a large community of users and developers around world.

## Features of NetBeans IDE :

- Fast and Smart Coding :
  - It’s not simply a text editor but much more than that. It helps you to code very easily and quickly by providing automatic line indents, it highlights similar keywords, and pair of brackets.
  - Also gives coding tips by opening pop-up menu to suggest the related code.

## Features of NetBeans IDE :

- Easy and Efficient Project Management :
  - It helps you to easily manage your project even if it is getting large with number of files with a large number of lines.
- Rapid User Interface Development :
  - It provides drag and drop facility to rapidly develop and designed your form or pages.

## Features of NetBeans IDE :

- Bug free Code :
  - The NetBeans provides static analysis tools such as FindBugs tool for identifying and fixing common errors in Java Code.

## How to Install NetBeans ?

- We can freely download the latest version of NetBeans form this site.

  *netbeans.org/downloads/*

## Eclipse

- Eclipse :
  - The Eclipse is another very powerful editor for so many languages such as Java, PHP, C, C++ etc.
  - The Eclipse IDE for Java developers contains everything that you will need to built Java applications.
  - It provides greater Java editing with validation, incremental compilation, cross-referencing, code assistant, and XML Editor and so many others.

## Features of Eclipse Editor :

- Great debugging support with hyperlink stepping.
- A new Quick Access feature to enhance IDE navigation.
- Quick Fix/Assist support.
- Task-focused development

- Eclipse can be freely download from *http://www.eclipse.org/downdoads/*

## Comments in JAVA :

- There are three types of comments in java.
  - Single line comments
  - Multi line comments
  - Documentation Comment

## Comments in JAVA :

- Single line comments
  - Double slashes are used to indicate single line comment.
  - Example :

    //This is a single line comment.

## Comments in JAVA :

- Multiline comments :
  - The text written between **/*** and ***/** considered as Multiline comments.
  - Example :

```
/* This line is in comment.
This is also in comment. */
```

## Comments in JAVA :

- Documentation Comment :
  - Documentation Comment is used to generate as HTML documentation file for your program. The text between **/**** and ***/** are documentation comment.
  - Example :

```
/**
  * This is documentation comment
  * @MONARCH Lathi
*/
```

## Identifiers :

- An identifier is a name given to a variable, a method, or a class.
- To declare an identifier we must follow some rules :
  - Identifier may consist of letters (uppercase or lowercase), digits, underscore or a dollar sign.
  - It must not start with a number.
  - No white space is allowed.

## Identifiers :

- No keyword can be used as identifier.
- Some valid examples of identifiers:
  - valididentifier, sub1, mark1, first_value, $price
- Some invalid examples of identifiers :
  - In valid, 1mark, first-value.

## Literals :

- In Java any constant value is known as literal.
- The literal can be any one of the following
  - Integer literal
  - Floating-point literal
  - Character literal

## Literals :

- Integer literal

  Example :

  ```
  int x=300;
  ```

  - Here **x** is a variable and **300** is value which is **integer literal**.

## Literals :

- Floating point literal :

  Example :

  ```
  double d=39.86;
  ```

  - Here **d** is a variable and **39.86** is floating point literal.

## Literals :

- Character literal :

  Example :

  ```
  char c='A';
  ```

  - Here **c** is a character type of variable and **'A'** is a character literal.
- Some character literals should be written using escape sequence character, which is back slash ('\').
- If we want to print some **special character** like quotes (" ") , we have to use back slash

## Literals :

- Character literal :

| Escape Sequence | Meaning |
|---|---|
| \n | To give new line |
| \b | Backspace |
| \t | To give space as much as tab |
| \’ | To print single quote |
| \” | To print double quote |
| \\ | To print back slash |

## Def. WAP to print following output

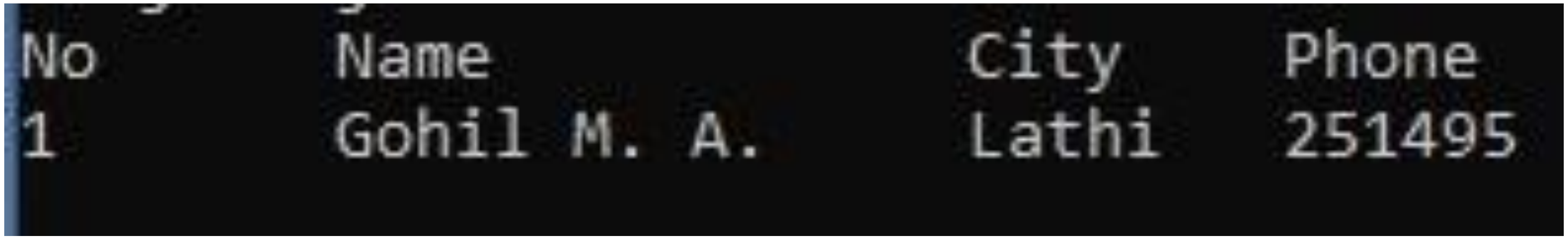
Welcome to my JAVA Program…

This is my “Special PROGRAM”.

Code :

```
class test
{ public static void main(String args[])
{ System.out.println("Welcome my JAVA Program");
  System.out.println("This is\"Special PROGRAM\"");
}
}
```

# Def. WAP to print PHONEBOOK



```
No      Name            City    Phone
1       Gohil M. A.     Lathi   251495
```

Code :

```
class test
{ public static void main(String args[])
{ System.out.println("No\tName\t\tCity\tPhone");
  System.out.println("1\tGohil M. A.\tLathi\t251495");
}
}
```

## White-spaces

- The white-space can be a space, a tab or a new line.
- The extra white space are ignored by Java compiler.
- Therefore if you write your program a single line it makes no difference.

## Separators :

- The separators are used to separate the statements for each other.
- For example,
  - The semicolon (;) is used to terminate a statement. The list of various separator is shown below.

| Separator | Name | Use |
|---|---|---|
| ; | Semicolon | Used to terminate a statement |

## Separators :

| Separator | Name | Use |
|---|---|---|
| , | Comma | Used to declare more than one variables of same type. Also used in for() loops to specify more than one conditions. |
| . | Dot or period | Used to access a variable or method of a class through object. Also used to separate package from sub package. |
| : | Colon | Used to specify a label. |
| ( ) | Paren thesis | Used in method declaration and method call. Also used in if conditions and loops. |

## Separators :

| Separator | Name | Use |
|---|---|---|
| { } | Braces | Used to enclosed statements of methods, class or any block of code. Also used to initialize an array. |
| [ ] | Brackets | Used to declare array and array elements. |

## Keywords :

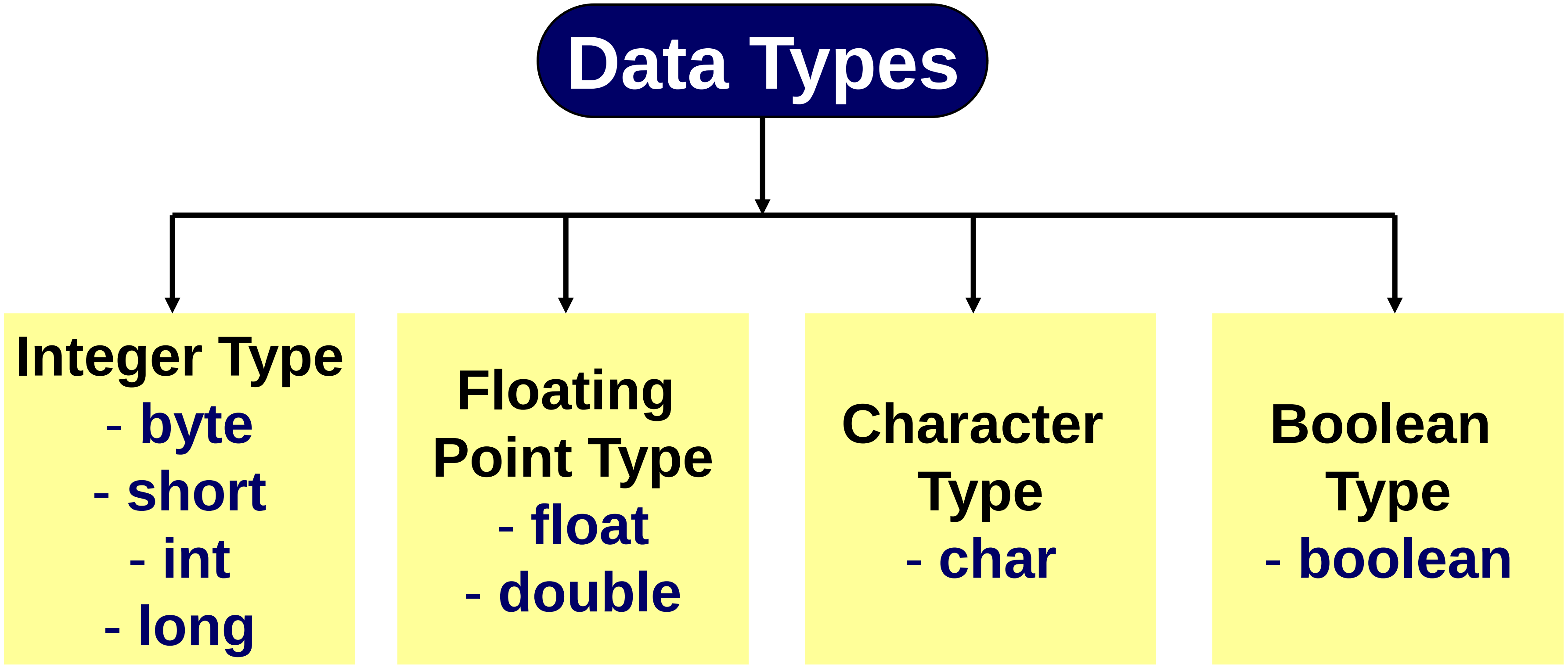
- Keyword is a pre-defined word. These words are reserved by programming language.
- There are 49 keywords in Java which are reserved and each one has its unique meaning.

## Keywords :

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

## Data Types :

- In Java, every variable has a data type. They can be classified as follows…



## Integer Data Type

- The byte, short, int and long are integer data types.
- They all are signed types. So they can have positive as well as negative values.

## Integer Data Type

| Type | Size (in bits) | Range |
|---|---|---|
| byte | 8 | -128 **to** 127 |
| short | 16 | -32,768 **to** 32767 |
| Int | 32 | -2,14,74,83,648 **to** 2,14,74,83,647 |
| long | 64 | -9,223,372,036,854,775,808 **to** 9,223,372,036,854,775,807 |

## Integer Data Type

- Byte:
  - Byte is one byte data type.
  - Its size is 8 bits and it can store values between $-2^7$ to $2^7-1$. (-128 to 127)
  - To declare a byte type variable, we have to use ***byte*** as a keyword.
- Example :

```
byte a;
```

## Integer Data Type

- Short:
  - Short is a 16 bit data type. It can store values between $-2^{15}$ to $2^{15}-1$.
  - To declare a variable of a short type ***short*** keyword is used.
- Example :

short a=1000;

## Integer Data Type

- Int :
  - Size of int type is 32 bits. Its range is -$2^{31}$ to $2^{31}$-1.
  - ***int*** keyword is used to declare an integer type variable.
- Example :

int i, j;

## Integer Data Type

- Long
  - Long type is of 64 bit and it can store values between $-2^{63}$ to $2^{63}-1$.
  - To declare a long type variable long keyword is used.
- Example :

long val;

## Floating point types :

- The float and double are floating point type data types.
- They are used to store floating point values. The size and range of these types are as follows.

| Type | Size (in bits) | Range |
|---|---|---|
| **float** | 32 | 1.4e - 045 **to** 3.4e + 038 |
| **double** | 64 | 4.9e - 324 **to** 1.8e + 308 |

## Floating point types :

- Float
  - Float data type is used when no more precision is required. Its size is 32-bits.
  - It is generally used to store rupees, dollars, etc. with some less digit precision.
- Example :

```
float f;
```

## Floating point types :

- Double
  - Double is a 64 bit data type. It stores data with double precision (keyword).
  - Therefore if we want to deal with the mathematical functions like sine, cosine, or want to find square root of a number then double data type is used.
- Example :

double d;

## Character Types

| Type | Size (in bits) | Range |
|---|---|---|
| char | 16 | 0 **to** 65,535 |

- Char type is used to store a single character.
- The char type in Java differs from C and C++.
- In C and C++ char type is 8 bits in size whereas here char is 16-bit wide which uses Unicode 16-bit format to store data which is used worldwide.

## Character Types

- Example :

  char c='A';

  char ch=65;

- Both the variables **c** and **ch** have the same value.

## Boolean type

| Type | Size (in bits) | Range |
|---|---|---|
| **boolean** | 8 | True **or** False |

- Boolean data type is 8 bit data type and used to store logical values. (true/false)
- Every relational operator returns a boolean value.
- Example :

```
boolean flag=false;
boolean isOK=true;
```

## DataType and its default values :

- In Java all variables are of a specific data type and must be initialized before using them.
- So if we do not initialize them, they have default values.
- But remember that these default values are just to initialize with default values, Java does not automatically initialize them.

## DataType and its default values :

| Data Type | Default Initial Value |
|---|---|
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |

| Data Type | Default Initial Value |
|---|---|
| float | 0.0f |
| double | 0.0d |
| char | ‘\0’ |
| boolean | false |

## Variable :

- A variable is an identifier for storage in a Java program. Each variable has a data type and it has a scope.
- **To declare a variable:**
- Syntax :

  Data_type Identifier [ = value];
- Example :

  int a=0; or int a;

  int a,b,c; or int a=11,b=220,c;

## Variable : scope and Lifetime of Variable

- When you declare a variable, it has a specific scope up to where it is visible and can be used.
- So when you declare a variable in a block, this block is the scope of this variable.
- So if you declare a variable in a method then it can be used within that method, if you declare it in a class, its scope is the entire class.

## Def. ___ WAP to demonstrate use of scope and lifetime of variable…

```
class sp
{ public static void main(String args[])
   {   int a=10;
       if(a>5)
       { int b=20;
         System.out.println("A = "+a+" and B= "+b);
         a++; }
       System.out.println("A is : "+a);
       //System.out.println("B = "+b);
       scope();}
```

## Def. ___ WAP to demonstrate use of scope and lifetime of variable…

Continue…

```
   static void scope()
   {
        int c=30;
        System.out.println("C = "+c);
        //System.out.println("A = "+a);
   }
}
```

## Type conversation and Type casting :

- In many expressions we generally assign some values or variables to other variables. These variables should be compatible with each other.
- In Java, before assigning value to a variable, if their types are not same, type conversation is necessary.

## Type conversation and Type casting :

- There are two types of type conversation
  - Implicit (Automatic) type conversion
  - Explicit type conversion also

## Type conversation and Type casting :

- Implicit (Automatic) type conversion
  - The implicit type conversion takes place automatically when the types are compatible and the destination type are larger than the source type.
  - **For Example,**
    - If we assign an int type value to a long type variable, it will be converted automatically.

## Type conversation and Type casting :

- Implicit (Automatic) type conversion
  - All numeric data types are compatible with each other but not with **char** or **boolean** type.
  - Because **char** and **boolean** are not compatible with each other.

## Type conversation and Type casting :

- Explicit Type Conversion :
  - When in an expression the destination type is not larger than the source type, the conversion will not be done automatically, but we have to do it explicitly by type casting.

## Type conversation and Type casting :

- **For Example,**
  - If we want to assign a long type value to an int type value, this will not done automatically, but you have to use a cast.
- The general form of type casting is:

  Var = (target-type) value;

Example:

```
long x=9999999;
int i=(int)x;
```

## Type conversation and Type casting :

- The following list shows which type can be converted automatically to which types.

| From | To |
|---|---|
| byte | short, int, long, float, double |
| short | int, long, float, double |
| char | int, long, float, double |
| int | long, float, double |
| long | float, double |
| float | double |

## Def.___ WAP to perform implicit and Explicit type casting.

```
class sp
{ public static void main(String args[])
   {   int i=1000;
       System.out.println("Value of i=" +i);
       long l;
       l=i; // this is implicit type conversion
       System.out.println("Value of L=" +l);
```

## Def.___ WAP to perform implicit and Explicit type casting.

```
      // Simple example
      long lg=999999;
      System.out.println("Value of Lg=“+lg);
      int ii;
      ii=(int)lg; //Explicit type casting
      System.out.println("Value of ii=" +ii);
   }
}
```

## Def. ____ WAP for type casting to convert **byte type** to **short type**

```
class sp
{ public static void main(String args[])
  {  byte a=10;
     int x=(short)a;
     System.out.println("Value of
                      Byte to short =" +x);
  }
}
```

## Def. ____ WAP for type casting to convert **short type** to **double**

```
class sp
{ public static void main(String args[])
  {   short a=10;
      double x=(double)a;
      System.out.println("Value of
                          Short to Double =" +x);
  }
}
```

## Array :

- An array is a group of variables of same data type which have same name.
- Therefore it is used when more than one variables of same data type are to be used in a program.

## Array :

- There are different types of array can be classified in JAVA.

  (1) One dimensional Array

  (2) Two dimensional Array

  (3) Multi dimensional Array

  (4) Jagged Array

## One dimensional Array :

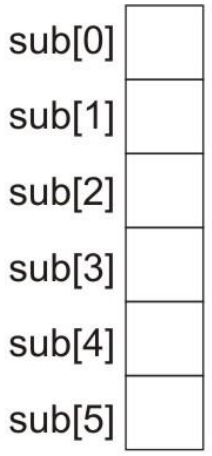
- A one dimensional array is a collection of variables with one row and multiple columns or one column and multiple rows.
- Declaration of one dimensional array:

  Syntax:

  ```
  Data_Type array_name[ ];
  ```

  Example :

  ```
  int sub[ ];
  ```



- Here the **sub** is said to be array of integers. The array is just created but it is not ready to use. To use it allocate memory space to it using new keyword.

## One dimensional Array :

- Here, the **int** means to said that its an array of integers.
- The array is just created but it is not ready to use. To use it allocate memory space to it **using new keyword**.

  Syntax:

  Array_name = **new** Data_Type[ ];

  - Example :

  marks = new int [10];

## One dimensional Array :

- Now marks is allocated size of **10** int values and it can be used now.
- Alternatively we can combine these two syntax like:

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

- Now to initialize array element you can refer to it by its index which starts from **0.**

## **Def___ WAP to assign 5 values to an array.** (Define individual values)

```
class sp
{     public static void main(String args[])
      {     int a[];
            a=new int[5];
            a[0]=10;
            a[1]=20;
            a[2]=30;
            a[3]=40;
            a[4]=50;
```

## Def___ WAP to assign 5 values to an array.

```
  System.out.println("Value of a[0]="+a[0]);
  System.out.println("Value of a[1]="+a[1]);
  System.out.println("Value of a[2]="+a[2]);
  System.out.println("Value of a[3]="+a[3]);
  System.out.println("Value of a[4]="+a[4]);
  }
}
```

## Def___ WAP to assign 5 values to an array. (Define all values at a time)

```
class sp
{ public static void main(String args[])
  { int a[ ]={10,20,30,40,50};
   System.out.println("Value of a[0]="+a[0]);
   System.out.println("Value of a[1]="+a[1]);
   System.out.println("Value of a[2]="+a[2]);
   System.out.println("Value of a[3]="+a[3]);
   System.out.println("Value of a[4]="+a[4]);
   }
}
```

## Def. ___ WAP to store marks for 4 subjects and print its total, percent

```
int marks[];
marks=new int[4];
marks[0]=45;
marks[1]=45;
marks[2]=45;
marks[3]=45;
tot=marks[0]+marks[1]+marks[2]+marks[3];
float per=tot/4;
System.out.println("Total  ="+tot);
System.out.println("Percent="+per);
```

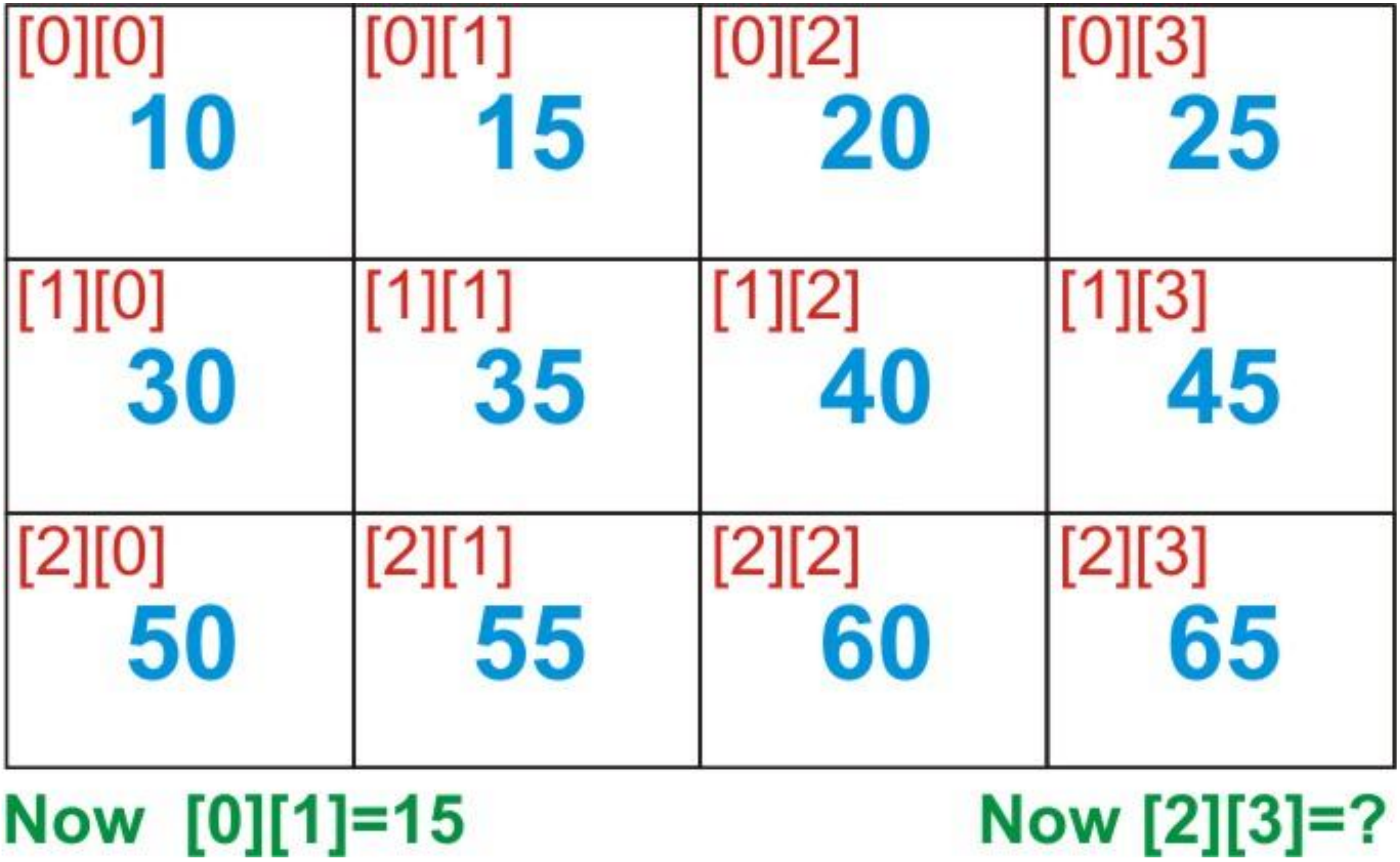
## Def. ___ WAP to store Name of Weekdays and print it using loop.

```
String days[]={"Mon","Tue","Wed",

                   "Thu","Fri","Sat","Sun"};

for(int i=0;i<7;i++)

{ System.out.println("Day = "+days[i]);

}
```

## Two or Multi Dimensional Array

- The multi dimensional arrays are array of array. It represents a variable which has values in tabular form i.e. in rows and columns.
- A list of items can be store in table format by row and column is known as Two-Dimensional Array.

# Two or Multi Dimensional Array



| | | | |
|---|---|---|---|
| [0][0] 10 | [0][1] 15 | [0][2] 20 | [0][3] 25 |
| [1][0] 30 | [1][1] 35 | [1][2] 40 | [1][3] 45 |
| [2][0] 50 | [2][1] 55 | [2][2] 60 | [2][3] 65 |

Now [0][1]=15 Now [2][3]=?

## Two Dimensional Array

- How to declare two dimensional array?

Syntax :

```
<datatype> NameOfArray[][]
                = new <datatype>[int][int];
```

Purpose :

- To declare two dimensional array we can use above syntax.

Example :

int **twoD**[][] = new int[4][5];

**Def.___ WAP to store result of 5 students with 4 subjects using TWO Dimensional array. Then print total, percent using loop.**

```
class TwoDArray
{ public static void main(String args[])
  {   int x, y;
      int sub[][]={
            {50,50,46,43}, {60,40,47,44},
            {70,40,48,45}, {80,45,49,46},
            {90,55,49,47} };
  for(x=0; x<5; x++)
  {System.out.print("\nStudnet "+(x+1)+ ": ");
      for(y=0; y<4; y++)
      {     System.out.print(sub[x][y]);
            System.out.print(",");
      }
  }
}
}
```

## Jagged Array :

- Jagged array is a special type of multi dimensional array in **which each row has different number of columns.**
- Therefore it is also known as variable length array.
- To declare a jagged array you have to specify only row size and leave the column size blank.

## Jagged Array :

Syntax:

```
Data_type array_name[ ][ ]=new
                        data_type[Rowsize][ ];
```

Purpose :

- To declare a jagged array we can use above syntax.

Example :

```
double d[ ][ ] = new double[5][ ];
int j[ ][ ]=new int[5][ ];
```

- Here **j** is a jagged array having **5** rows.

## Jagged Array :

- Now we can declare the size of each columns for each row as shown below.

```
j[0]=new int[3];
j[1]=new int[2];
j[2]=new int[4];
j[3]=new int[3];
```

## Jagged Array :

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | J[0][0] | J[0][1] | J[0][2] | |
| 1 | J[1][0] | J[1][1] | | |
| 2 | J[2][0] | J[2][1] | J[2][2] | J[2][3] |
| 3 | J[3][0] | J[3][1] | J[3][2] | |

## Def. ___ Demonstrate use of jagged array.

```
class jagged
{ public static void main(String args[])
  {  int a[][]={
             {1,2,3,4},
             {1,2},
             {1,2,3},
             {1,2,3,4,5}
             };
  System.out.println("\nJagged Array...\n");
  for(int i=0;i<a.length;i++)
  {  for(int j=0;j<a[i].length;j++)
     {     System.out.print(a[i][j]+" ");
     }
     System.out.println("\n");
  }
}}
```

## Operators :

- Meaning : Operator is a symbol that performs some operations on its operands and gives the results.
- Java has a rich set of operators. Operators in java can be divided in main four categories.

  (1) Arithmetic Operators

  (2) Relational Operators

  (3) Logical Operators

  (4) Bitwise Operators

## (1) Arithmetic Operators :

- Arithmetic Operators are those which are used in mathematical calculators such as addition, multiplications etc.
- Arithmetic operators can be divided in further sub categories.

(1) Basic Arithmetic Operators

(2) Increment/Decrement Operators

(3) Short-Hand Operators

## (1) Arithmetic Operators :

- In all these arithmetic operators we can use only numeric data type values such as int or float, but we cannot use boolean type values in these operators.
- Whereas we can use char type values because in fact char is a subtype of int data type.

## (1) Arithmetic Operators :

- Basic Arithmetic Operators :

| Operator | Description |
|---|---|
| + | Performs addition of given two operands. Also used as unary plus operator. |
| - | Performs subtraction of given two operands. Also used unary minus operator. |

## (1) Arithmetic Operators :

- Basic Arithmetic Operators :

| Operator | Description |
|---|---|
| * | Performs multiplication of given two operands. |
| / | Performs division of given two operands. |
| % | Returns the reminder of division (Modules) |

## (1) Arithmetic Operators :

- All basic arithmetic operators are binary operators and the **+** and **–** operators are both binary and unary operators.
- Unary (-) operator simply changes the sign of the operands.

## Def. ___ Demonstrate Use of arithmetic operators...

```
class test
{ public static void main(String args[])
  {   float a=10, b=20, c;
      c=a+b;
      System.out.println("\nA+B = "+c);
      c=a-b;
      System.out.println("\nA-B = "+c);
```

```
    c=a*b;
    System.out.println("\nA x B = "+c);
    c=a/b;
    System.out.println("\nA / B = "+c);
  }
}
```

## (1) Arithmetic Operators :

- The Modules Operator (%):
  - The modulus operator in Java is more powerful than other languages.
  - Because it can be applied to integer type values as well as floating point values.
  - It simply returns the reminder of division operation.

## **Def. ___ check that given number is odd or even.** (Using modules operator)

```
class test
{ public static void main(String args[])
  {   float a=10;
      if(a%2==0)
      {System.out.print("\nGiven Number is ODD");
      }
  }
}
```

## Def. ___ Display reminder values of integer & float . (Using modules operator)

```
class mod
{ public static void main(String args[])
   {   int num1=11,imod;
       double num2=11.5,fmod;
       imod=num1%10;
       fmod=num2%10;
       System.out.print(“Integer Reminder is:"+imod);
       System.out.print(“Floating Reminder is:"+fmod);
   }
}
```

## Increment (++) / Decrement(--) Operators:

- Both these operators are unary operators.
- The increment operator increases the value of operand by one and the decrement operator decreases the value of operand by one.
- For Example:

  a=a+1;  is equivalent to a++;  and
  b=b-1;  is equivalent to b--;
- Here, these operators can be used in one of two forms: Postfix and Prefix.

## Increment (++) /Decrement(--) Operators:

- Postfix operator is one which the operator is put after (**post**) the operand and in Prefix operator the operator is put before (**pre**) the operand.
- ***For Example:***

  x++ is same as ++x and y-- is same as --y

  int a=10, b=20;

  int c=++a;

  Here, value of c will 11 and a will be 11 too.

  But, if we use postfix operator, like

  int d=b++;

  Then the value of d will be 20 and b will be 21.

## Increment (++) /Decrement(--) Operators:

- Because in case of postfix operator the increment or decrement the happens after (post) the assignment, and in prefix the increment or decrement happens before assignment takes place. i.e.

  a=10;

  c=++a; $\longrightarrow$ ++a; $\longrightarrow$ a=a+1;

  Here, c=11 and a=13

## Def : Write a program to print value of variables using increment / decrement operator.

```
class InDe
{ public static void main(String args[])
   {  int a,b,c,d;
      a=b=c=d=10;  d=--b;  c=a++;
      System.out.print(“a :”+a);
      System.out.print(“b :”+b);
      System.out.print(“c :”+c);
      System.out.print(“d :”+d);
  }}
```

## Short hand Operators :

- Short hand operators are the combination of an arithmetic operator and assignment operator.
- These operators are used because they are shorter to write and faster (efficient) to use.

## Short hand Operators :

- Syntax :

  variableName=VariableName ***ope*** expression;

- Here, variable name is any valid variable, ***ope*** is any arithmetic operator and *expression* is any expression.

- **For Example :**

  x=x+3; is same as x+=3;

- Its advantage can be seen when complex operation is involved.

## Short hand Operators:

```
x=x+(y*z*2.5);
```

- This statement can be rewrite using short hand operator as follows:

```
x+=y*z*2.5;
```

## Def.___ Demonstrate use of Short Hand Operator.

```
class shorthandOpe
{ public static void main(String args[])
  {  int a,b,c;
     a=b=c=10;  a+=5;  b/=2;      c%=3;
     System.out.println("a :"+a);
     System.out.println("b :"+b);
     System.out.println("c :"+c);
  }}
```

## (2) Relational Operators:

- The relational operators in Java are used to check the relation between two operands.
- They return a Boolean value i.e. **true** or **false**.
- The relational operators are listed below:

| Operator | Description |
|---|---|
| = = | Is equal to |
| ! = | Is not equal to |
| < | Is less than |
| > | Is greater than |
| < = | Is less than or equal to |
| > = | Is greater than or equal to |

## Def.___ Demonstrate use of Relational Operator.

```
class rOpe
{ public static void main(String args[])
  {  int a,b,c,d;
     a=10;b=20;c=10;d=5;
     System.out.println("a="+a);
     System.out.println("b="+b);
     System.out.println("c="+c);
     System.out.println("d="+d);
```

## Def. Continue...

```
        System.out.print("a==b is :"+(a==b));
        System.out.print("a==c is :"+(a==c));
        System.out.print("a>b is :"+(a>b));
        System.out.print("a<=b is :"+(a<=b));
        System.out.print("a>d is :"+(a>d));
        }
}
```

## (3) Logical Operators:

- Logical operators in Java are used to combine two or more Boolean conditions together.
- Here the operands must be of Boolean type variables or conditions and the results returned is also a Boolean value.

## (3) Logical Operators:

| Operator | Description |
|---|---|
| & | Logical AND |
| && | Short-Circuit AND |
| \| | Logical OR |
| \|\| | Short-Circuit OR |
| ^ | Exclusive OR |
| ! | Logical NOT |

## (3) Logical Operators:

| X | Y | X&Y | X\|Y | X^Y | !X | !Y |
|---|---|---|---|---|---|---|
| True | True | True | True | False | False | False |
| True | False | False | True | True | False | True |
| False | True | False | True | True | True | False |
| False | False | False | False | False | True | True |

## (3) Logical Operators:

- The logical AND (&) operators returns true if both the operands are true else returns false.
- Whereas the logical OR (|) operator returns false if both the operands are false else returns true. Means it returns true if any one operand (condition) is true.

## (3) Logical Operators:

- The Ex-OR (^) operator returns true if from both operands exactly one operand is true (i.e. not zero or both operands are true).

## Def.____ Demonstrate use of LOGICAL operators.

```
class LOpe
{ public static void main(String args[])
  {   boolean a=(10<5);          // false
      boolean b=('1'=='1');      // true
      boolean c=(5==5);          //true
      System.out.print("a="+a);
      System.out.print("b="+b);
      System.out.print("c="+c);
```

## Def.____ Demonstrate use of LOGICAL operators.

```
        System.out.print("a&b="+(a&b));
        System.out.print("a|b="+(a|b));
        System.out.print("a^b="+(a^b));
        System.out.print("b^c="+(b^c));
        System.out.print("!a="+(!a));
    }
}
```

## Difference between simple and short-circuit operators:

- The **&&** and **||** operators are known as short-circuit operators which are not available in other programming languages.
- There is a minor difference between short-circuit and simple operators.

## Difference between simple and short-circuit operators:

- From the operation logical AND and logical OR operators we can find that the AND operator evaluates to true if both the operands are true and otherwise false. Means if anyone operand is false then the result is false.

## Difference between simple and short-circuit operators:

- And the OR operator evaluates to true if anyone operand is true no matter what the other is.
- The short-circuit operators uses this concept to perform AND / OR operation.

## Difference between simple and short-circuit operators:

- Short-circuit AND (&&) operator checks the first condition and if it is false then it doesn’t check the next condition, as this will evaluates to false.
- Same as in short-circuit OR (||) operator, on encounter of a true expression it will never check the next condition and will evaluate to true.

## Difference between simple and short-circuit operators:

- For example :

```
if( b!=0 && a | b > 5)
{
        //statements….
}
```

- Here if ***b*** is **0** then the first condition becomes false so the next expression will never be performed.

## Difference between simple and short-circuit operators:

- So there is no change of divide by exception. But if this statement have written using & operator then both conditions would have checked.

## (4) Bitwise Operators:

- Bitwise operators can be used with integer type values.
- As the name suggest they perform operations on operands bit by bit.
- Java has following bitwise operators:

| Operator | Description |
|---|---|
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Bitwise exclusive OR (XOR) |
| ~ | Bitwise unary NOT |
| << | Bitwise shift left |
| >> | Bitwise shift right |

## (4) Bitwise Operators:

- These operators operate on bits. So all the operands are converted into binary number system first and then the operation is perform on bits.
- Bitwise AND, OR, XOR and NOT uses the same logic as the logical operators.
- Consider **1** for true and **0** for false.

| **X** | **Y** | **X&Y** | **X\|Y** | **X^Y** | **!X** | **!Y** |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

## Def.___ Demonstrate use of Bitwise Operator.

```
class bOpe
{ public static void main(String args[])
  {  int a=12;
     int b=10;
     System.out.println("a\t="+a);
     System.out.println("b\t="+b);
     System.out.println("a&b\t="+(a&b));
     System.out.println("a|b\t="+(a|b));
```

## Def.___ Demonstrate use of Bitwise Operator.

```
  System.out.println("a^b\t="+(a^b));
  System.out.println("~a\t="+(~a));
  System.out.println("a<<2\t="+(a<<2));
  System.out.println("a>>2\t="+(a>>2));
  }
}
```

## Conditional Operator

- C programming offers the conditional operator used in C language is ( ? ) question mark and ( : ) column.
- It is used for checking the condition and gives output according to the condition.
- If condition is true then it will execute the first value or it will execute the second value given in the syntax.

## Conditional Operator

- Syntax :

  <condition>?<True Value>:<False Value>

- Purpose:
  - If condition will be true then it will return the true value else it will return the false value.
- Example :

  c=(a>b)?a:b;

## Def.___ WAP to store two value in variables and check for big value.

```
class ConOp
{ public static void main(String args[])
  {int a=12;
   int b=100;
   int c=(a>b)?a:b;
   System.out.println("Big Value is "+c);
  }
}
```

## Control Structure :

- There are two type of Control structure
  - Conditional
  - Looping
- In conditional statement there are optional environment available.
- In looping statements we have to repeat same code one or more time.

## Select Control Structure :

- If Statement
- Switch Statement

## Select Control Structure :

- If Statement in Details
  - We can write conditional statement **IF** in different ways...
    - if (Simple if Without Else Statement)
    - if....else
    - Nested if (if within if)
    - if....else if ladder

## if (Simple if without else)

Syntax :

```
if(condition)
{    statements;
}
```

- Purpose:
  - **If statement is the decision control statement.** This statement will check the given condition. If condition is **TRUE** then it **executes given statements**.

## Def.___ WAP to store your age, Now check that you can VOTE or Not. (Using IF)

```
class vote
{ public static void main(String args[])
  {  int age=18;
     if(age>18)
        System.out.print("You Can Vote");
     if(age<=18)
        System.out.print("You Can’t Vote");
  }
}
```

## if....else

Syntax : **if(condition)**

```
{    statement Group1;     }
 else
{    statement Group2 ;    }
```

- **Purpose:**
  - **If..else statement is a decision control statement.**
  - If condition is **TRUE** then **it execute given statements group1**. If condition if **FALSE** then **it executes statement group2**.

**Def.___** WAP to store your age, Now check that you can VOTE or Not **(Using if..else)**

```
class vote
{ public static void main(String args[])
  {  int age=18;
     if(age>18)
        System.out.print("You Can Vote");
     else
        System.out.print("You Can’t Vote");
  }
}
```

## if….else [ Nested ]

- Syntax :

```
if(Condition)
{     if(Condition)
      {          }
      else
      {          }
}
```

Purpose:

- If..else nested is also decision control statement.

## if….else [ Nested ]

- If outer condition will be true then it will go inside and check inner condition. This method is known as NESTED IF…ELSE
- It will check conditions more then one time.

## **Def.___** Check that you can *VOTE or NOT* & You Can *Marry or NOT* ***(Using Nested If)***

```
int age=22;
if(age>18)
{  System.out.print("You Can VOTE");
   if(age>21)
     System.out.print("You Can MARRY");
   else
     System.out.print("Sorry! You Can't MARRY");
}
```

## The if-else-if Ladder

```
Syntax : if(condition1)
                    statement block1;
              else if(condition2)
                    statement block2;
             else if(condition3)
                    statement block3;
             else
                    statement block n;
```

## The if-else-if Ladder

Purpose:

- Here, condition1 is evaluated first, if it is true then the statement block 1 will be executed. If condition1 is false then condition2 will be evaluated, if condition2 is true then the statement block 2 will be executed. This will happen until any one condition will be true. If all the condition are false then the **else** part will be executed.

## Def. WAP to store Student Percent and print its CLASS [ Distinction, First, Second, Pass, *** ] (using if... else... if... ladder)

```
int per=68;
if(per>=70)
    System.out.print("Class = Dist");
else if(per>=60)
    System.out.print("Class = First");
```

## Def. Continue...

```
else if(per>=48)
    System.out.print("Class = Second");
else if(per>=40)
    System.out.print("Class = Pass");
else
    System.out.print("Class = * * *");
```

## How To Use Command line Arguments?

- We can use command line arguments in our program.
- Default its STRING arguments…
- To run Command line arguments we have to use following syntax.

  Syntax :

  ```
  java <java.classFile> <arg1> <arg2>..
  ```

  Example :

  java **welcome** Manoharsinh

## Def ___ WAP to pass your name as argument on command line and print a welcome message with name.

```
class welcome

{ public static void main(String args[])

   { System.out.print("Welcome "+args[0]);

   }

}
```

## How To Use Command line Arguments?

- We can also use multiple command line arguments in our program.

  Example :

  ```
  java sum 25 30
  ```

  Output :

  ```
  Sum = 55
  ```

## Def ___ WAP to pass tow values as arguments on command line and print its sum.

```
int v1, v2, tot;
v1=Integer.parseInt(args[0]);
v2=Integer.parseInt(args[1]);
tot=v1+v2;
System.out.println("Value 1 = "+v1);
System.out.println("Value 2 = "+v2);
System.out.println("Total V1+V2= "+tot);
```

## switch... case

Syntax :

```
switch(expression)
{
  case value1:
        statement block1;
        break;
  case value2:
        statement block2;
        break;
```

## switch... case

```
        case value3:
                statement block3;
                break;
        case valueN:
                statement blockN;
                break;
        default:
                default statement block;
      }
```

## switch... case

Purpose:

- In switch statement, the expression gives one value which is compared with the values written with case statements.
- If one case is evaluated **true** then the statement block related to that case will be executed.

## Def. Write a program to Enter Two Values and print its ***Sum*** or ***Subtraction*** or ***Multiplication*** or ***Division.*** *(using three arguments)*

```
class args
{ public static void main(String args[])
  {  int v1, v2;
     int choice;
     v1=Integer.parseInt(args[0]);
     v2=Integer.parseInt(args[1]);
     choice=Integer.parseInt(args[2]);
```

```
System.out.print("\nWhat To Do ?");
System.out.print("\n1 = Sum");
System.out.print("\n2 = Subtraction");
System.out.print("\n3 = Multiplication");
System.out.print("\n4 = Division");
System.out.print("\nEnter Your Choice :");
switch(choice)
{  case 1:
      System.out.print("Sum = "+(v1+v2));
      break;
```

```
  case 2:
      System.out.print("Subtraction = "+(v1-v2));
      break;
  case 3:
      System.out.print("Multiplication = "+(v1*v2));
      break;
  case 4:
      System.out.print("Division = "+(v1/v2));
      break;
  default:
      System.out.print("\n Selection is WRONG");
  }}
}
```

## Looping Statements

- If we want to execute one statement for one or more time, in programming language there are some facility available for that.
  - In looping we have 2 type of looping structure available.

    1. Entry control loop
    2. Exit control loop

## Looping Statements

- 1. Entry Control Loop
  - There are two types of entry control loops available
    - for loop
    - while loop

## For... (Entry Control Loop)

- Syntax :

```
for(initialization;condition;increment or decrement)
{       statement 1;
        statement 2;
}
```

- Purpose:
  - For loop is a counter loop. We can define number of loops with **specified number** with **condition**.

## For… (Entry Control Loop)

**Syntax explanation…**

- Initialization
  - Its a portion where we can define a value for counter variable.
- Condition
  - We can specify the condition (logic) for counter loop.
- Increment or Decrement in variable
  - We can increase or decrease in variable using this syntax.

## Def.. Write a program to print your name 10 times.

```
class args
{ public static void main(String args[])
  {  int x;
     for(x=1;x<=10;x++)
          System.out.println("MONARCH");
  }
}
```

## Def.. Write a program to print your 1 to 10 times.

```
class args
{ public static void main(String args[])
  {  int x;
     for(x=1;x<=10;x++)
          System.out.println(x);
  }
}
```

## Def.. Write a program to print your 10 to 1 times.

```
class args
{ public static void main(String args[])
  {  int x;
     for(x=10;x>=1;x--)
          System.out.println(x);
  }
}
```

## while... (Entry Control Loop)

- Syntax :

```
while(condition)
{     statement 1;
      statement 2;
}
```

- Purpose:
  - While loop is a conditional loop. We can set condition as required. It loops till the condition will be true.

## Def.. Print 1 to 10 using while loop

```
class args
{ public static void main(String args[])
  {  int x=1;
     while(x<=10)
     {     System.out.println(x);
           x++;
     }
  }
}
```

## Def.. Print 10 to 1 using while loop

```
class args
{ public static void main(String args[])
  {  int x=10;
     while(x>=1)
     {    System.out.println(x);
          x--;
     }
  }
}
```

## **do…while** (Exit Control Loop)

- Syntax :

```
do
{     statement 1;
      statement 2;
}while(condition);
```

- Purpose:
  - Do…While loop is a conditional loop. We can set condition as required. ***It runs minimum one time.*** Then it loops till the condition will be true.

## Def.. Print 1 to 10 using do while loop

```
class args
{ public static void main(String args[])
  {  int x=1;
     do
     {    System.out.println(x);
          x++;
     }while(x<=10);
  }
}
```

## Def.. Print 10 to 1 using do while loop

```
class args
{ public static void main(String args[])
  {  int x=10;
     do
     {    System.out.println(x);
          x--;
     }while(x>=1);
  }
}
```

## **Def..** WAP to print sum of 1 to 10 serial number. **(Using do...while)**

```
int x=1, sum=0;
do
{  sum=sum+x;
   x++;
}while(x<=10);
System.out.print("Sum Of Values "+sum);
```

## Break statement…

Syntax :

```
break;
```

Purpose :

- The break statement is used to terminate loops or to exit from switch case.
- The break statement will break or terminate the inner-most loop.
- We can use **break** statement within **while**, **do-while**, **for** or **switch case** statement.

## Def. Design a for loop to print 1 to 10. But break it after print 1 to 5 using break statement.

```
{  int a;
   for(a=1;a<=10;a++)
   {     System.out.print("\n"+a);
         if(a==5)
               break;
   }
}
```

## CONTINUE STATEMENT…

- Syntax :

  continue;

- Purpose :
  - Continue statement is used to skip or to bypass some step or interaction of looping structure.
  - It does not terminate the loop but just skip the particular sequence of the loop structure.

## Def. Design a while loop to print 1 to 10. But print only odd numbers 1,3,5,7,9 using continue statement.

```
int a=0;
while(a<10)
{    a++;
     if(a%2==0)
          continue;
     System.out.print("\n"+a);
}
```

## Return Statement :

- The return statement cause the control to be transferred back from a method to the caller of the method.

Syntax :

```
return;
```

## Def. Design a **while loop** to demonstrate use of “return” statement.

```
int i=1;
while(i<10)
{     if(i==5)
            return;
      System.out.println("i is "+i);
      i++;
}
```

## OOP Concept…

- ***Class***
- ***Object***

## Class :

- Java is Object Oriented Programming language. Classes and objects are the most fundamentals parts of Java.
- Class is the way by which Java achieves encapsulation.
- To define a class ***class*** keyword is used.

## Class :

- How to define a Class?
- Syntax :

```
class <nameofClass>{
  datatype var1;
  datatype var2;
  datatype varN;
}
```

## Def___ WAP to define a class.

```
class student{
   int rno;
   int sub1, sub2, sub3, sub4;
   int tot;
   float per;
}
```

**Note :** Class must define above the main class.

## Creating Objects :

- When we want to use any class, we have to create its object. By object we can access member of class.
- To create an object :

  Syntax :

  ```
  className <ObjectName>=new ClassName();
  ```

  Purpose :

  - To create an object of class, we can use above syntax.

## Creating Objects :

Example :

```
student s1=new student();
```

- Here, the class **student** has instance variables. So after creating its object **s1** we can access these variable.
- The object of class is declared by the above statement.

Def.___ WAP for student result demonstrate to create a class and its object.

```
class student{
  int rno, s1, s2, s3, s4, tot;
  float per;
}
class newClass
{ public static void main(String args[])
  {   student s=new student();
```

```
s.rno=25;

s.s1=55;

s.s2=65;

s.s3=50;

s.s4=45;

s.tot=s.s1+s.s2+s.s3+s.s4;

s.per=(float)s.tot/4;
```

```
      System.out.print("Roll Num:"+s.rno);
      System.out.print("Subject 1 : "+s.s1);
      System.out.print("Subject 2 : "+s.s2);
      System.out.print("Subject 3 : "+s.s3);
      System.out.print("Subject 4 : "+s.s4);
      System.out.print("TotalMark :"+s.tot);
      System.out.println("Percent :"+s.per);
   }
}
```

## New Keyword :

- The **new** keyword is used to allocate memory equal to the size of instance variables of class.
- Before using **new** keyword the object can not be used. So you can say that the object gets physical existence after the memory allocation by **new** keyword.
- The **new** keyword allocates memory dynamically-at run-time to objects.

## New Keyword :

- We can create as many objects as we want.
- We can also define **reference** to object.

student **s**=new student();

student **ref**=**s**;

  - Here, **s** is assigned memory by new keyword but **ref** is not assigned with memory. So **ref** is a reference variable of student **s**.

Def.___ WAP for student result demonstrate to create a class and its object. Using reference.

```
class student{
   int rno, s1, s2, s3, s4, tot;
   float per;
}
class newClass
{ public static void main(String args[])
   {  student s=new student();
```

```
student ref=s;
ref.rno=25;
ref.s1=55;
ref.s2=65;
ref.s3=50;
ref.s4=45;
ref.tot=ref.s1+ref.s2+ref.s3+ref.s4;
ref.per=(float)ref.tot/4;
```

**Here, ref** is a reference variable **ref** & **s** both pointing to the same object **s**...

```
      System.out.print("Roll Num:"+s.rno);
      System.out.print("Subject 1 :"+s.s1);
      System.out.print("Subject 2 :"+s.s2);
      System.out.print("Subject 3 :"+s.s3);
      System.out.print("Subject 4 :"+s.s4);
      System.out.print("TotalMark :"+s.tot);
      System.out.print("Percent :"+s.per);
   }
}
```

## Adding Methods to class :

- As per requirement we can instance variable to any class. Same as that we can add methods to a class.
- The main string of a class is in its methods. The general form of a method is is given in the syntax.

## Adding Methods to class :

Syntax :

```
returnType methodName(parameter-list){
    //Statements
    //Optional return statement
}
```

Purpose :

- To define a method we can use above syntax.

## Def.___ Demonstrate use of method to calculate interest. (void type)

```
class interest{
  float amount, rate, year;
  void si(){
      float interest;
      interest=amount*rate*year/100;
      System.out.print("Interest:"+interest);
      }
}
```

```
class intCalc
{ public static void main(String args[])
  {   interest i1=new interest();
      i1.amount=5000;
      i1.rate=12;
      i1.year=2;
      System.out.print("Amount:"+i1.amount);
      System.out.print("Rate :"+i1.rate);
      System.out.println("Year :"+i1.year);
      i1.si();
  }
}
```

## Def.___ Demonstrate use of method to calculate interest. (Method with return type)

```
class interest{
  float amount, rate, year;
  float si(){
          float interest;
          interest=amount*rate*year/100;
          return interest;
      }
}
```

```
class intCalc
{ public static void main(String args[])
  {   interest i1=new interest();
      i1.amount=5000;
      i1.rate=12;
      i1.year=2;
      System.out.print("Amount:"+i1.amount);
      System.out.print("Rate    :"+i1.rate);
      System.out.print("Year    :"+i1.year);
      System.out.print("Interest:"+i1.si());
  }}
```

## Def.___ Demonstrate use of method to calculate interest. (Method with argument with return type)

```
class interest{
   float amount, rate, year;
   float si(float amount, float rate, float year){
        float interest;
        interest=amount*rate*year/100;
        System.out.print("Amount : "+amount);
        System.out.print("Rate       : "+rate);
        System.out.print("Year       : "+year);
        return interest;  } }
```

## Continue...

```
class intCalc

{ public static void main(String args[])

   {interest i1=new interest();

    System.out.print("Interest:" +i1.si(5000,12,2));

   }

}
```

## Constructors :

- The constructor is a special method which has the same name as its class.
- The constructor initializes the object of its class automatically at the time of creation.
- An important point about constructors is that *they have no return type*. Not even void. This is because they actually return the object of its class.

## Constructors :

- When we create an object, we call the constructor actually return the object of its class. So the statements written in the constructor are executed at that time.

## Def.___ Demonstrate use of method to calculate interest. (Using Constructor)

```
class interest{
  float p;
  float r;
  float n;
  interest(){                    //Its constructor
      System.out.print("Executing Constructor");
      p=5000;
      r=12;
      n=5;
      }
```

## Def. Continue...

```
   float simpleinterest(){
        return (p*r*n/100);
        }
}
class args{
   public static void main(String args[]){
        interest i1=new interest(); //Constructor
        float si=i1.simpleinterest();
        System.out.println("Interest is "+si);
   }
}
```

## Def.___ Demonstrate use of method to calculate interest. (Using arguments with Constructor)

```
class interest{
   float p;          float r;     float n;
   interest(float pa, float ri, float ny){ //Its constructor
        System.out.print("Executing Constructor...");
        p=pa;     r=ri;       n=ny;
        }
   float simpleinterest(){  return (p*r*n/100);
        }
}
```

## Def. Continue...

```
class args{      //Class With Main...
  public static void main(String args[]){
      interest i1=new interest(5000, 12, 2);
      float si=i1.simpleinterest();
      System.out.println("Amount\t: "+i1.p);
      System.out.println("Rate\t: "+i1.r);
      System.out.println("Year\t: "+i1.n);
      System.out.println("Interest is "+si);
  }
}
```

## Use of *this* keyword :

- ‘this’ keyword refers to the object of the current class. So when you want to access to the instance member of the current class in a method of current class, ‘this’ keyword is used.
- Sometimes we have same names of instance variables and other local variables in this case, ‘this’ keyword is used to distinguish them.

## Def.___ Demonstrate use of method to calculate interest. (Using this keyword)

```
class interest{
   float p;         float r;     float n;
   interest(float p, float r, float n){ //Its constructor
        System.out.print("Executing Constructor...");
        this.p=p;
        this.r=r;
        this.n=n; }
   float simpleinterest(){  return (p*r*n/100);
        }
}
```

## Def. Continue...

```
class args{        //Class With Main...
  public static void main(String args[]){
      interest i1=new interest(5000, 12, 2);
      float si=i1.simpleinterest();
      System.out.println("Amount\t: "+i1.p);
      System.out.println("Rate\t: "+i1.r);
      System.out.println("Year\t: "+i1.n);
      System.out.println("Interest is "+si);
  }
}
```

## Method Overloading : (Polymorphism)

- As we know that polymorphism is an important feature of OOP.
- Method overloading is a way by which Java achieves polymorphism. In a class you can define more than one method.
- When in a class there is more than one method with same name but different number and/or types of parameters, the method is said to be overloaded and this process is known as method overloading.

## Method Overloading : (Polymorphism)

- The number and type of parameters of a method is known as type signature. Remember that return type is not considered in type signature.
- So we can say that if there is more than one method in a class with same name but different type signature, the method is said to be overload.

## Method Overloading : (Polymorphism)

- Now, you might be wondering that how a call to the method will be resolved. When a method is called, Java matches the exact definition for the method.
- So when the number and type of parameters matches, Java invoke that method. If no exact type of parameter is matched, Java tires to convert the type.

## Method Overloading : (Polymorphism)

- If it can be done implicitly, Java does it and calls that method. If you have two or more methods with same type signature but with different return types, it is not method overloading.
- In this case, we will get an error saying “method is already defined in class”.

## Def.___ Use a method to total 2,3,4 parameteres. (Using overloaded)

```
class sum{
  void sum(){
      System.out.print("Sorry No Argument");
  }
  void sum(int a, int b){
      int x=a+b;
      System.out.print("Total A + B \t:"+x);
  }
  void sum(int a, int b, int c){
      int x=a+b+c;
      System.out.print("Total A+B+C \t:"+x);
  }
}
```

```
class args{
  public static void main(String args[]){
     sum s=new sum();
     s.sum();                    //Method Overloaded
     s.sum(10,20);
     s.sum(10,20,30);
  }
}
```

## Constructor Overloading :

- Like normal methods, we can also overload constructors in Java.
- When we create more than one constructor with different parameters, it is called constructor overloading.
- Sometimes it is needed to create objects with less or no parameters.

## Def.___ Use a method to total 2,3,4 parameteres. (Using constructor overloaded)

```
class sum{
  sum(){
      System.out.print("Sorry No Argument");
  }
  sum(int a, int b){
      int x=a+b;
      System.out.println("Total A+B\t:"+x);
  }
```

## Def. Continue...

```
  sum(int a, int b, int c){
      int x=a+b+c;
      System.out.print("Total A+B+C="+x);
  }
}
```

## Def. Continue…

```
class args{
  public static void main(String args[]){
      sum s=new sum();
      sum s1=new sum(10,20);
      sum s2=new sum(10,20,30);
  }
}
```

## Access Protection :

- In a class we can specify which member of our class can be accessed by other variables.
- To do this, Java provides three access specifiers :
  - Public
  - Private and
  - Protected

## Access Protection :

- If you do not specify any of these to a member, Java assigns it default access. The meanings of them are as under :
- **Public :**
  - When a member is defined by public specifier, it can be accessed from anywhere. By its class members and also by members of other class and other package. **'Public'** keyword is used to declare a member as public.

## Access Protection :

- **Private :**
  - A member modified as private can be accessed only by the members of its class.
  - The member of other class can not access private members. **‘private’** keyword is used to declare a member as private.

## Access Protection :

- **Protected :**
  - This specifier is used in inheritance only. The protected members can be accessed by the members of other package, but only to the subclass of this class.
  - Package and inheritance are discussed in later chapters. **'protected'** keyword is used to declare a member as protected.

## Access Protection :

- **Default :**
  - When you do not specify any access specifier, the default specifier is applied to that member.
  - The default members can be accessed from the members of any class, but in the same package.

## Access Protection :

- This is the reason why the **main()** method is always specified as public.
- Because the Java run this system calls this method to execute your program.

## Def.___ WAP to demonstrate use of access specifiers...

```
class simple{
  int i;                //Default Access
  public int j;         //Public Access
  private int k;        //Private access
  void setValue(int k){
  this.k=k;     //Private member can be
                accessed only from same class member
  }
```

## Def... Continue...

```
int getValue(){
                    return k;     }
}
class args{
  public static void main(String args[]){
      simple s=new simple();
      s.i=10;   //It accessed from any class, same package
      s.j=20;   //Can be accessed from any class, any package
  // s.k=30;   //Error : con't access k as it is private
```

## Def… Continue…

```
        s.setValue(30);    //setValue() bas default access
        System.out.print("i= "+s.i);
        System.out.print("j= "+s.j);
        System.out.print("k="+s.getValue());
                                   //getValue() has default access
    }
}
```

## The static keyword :

- Normally we access member variables and methods by the object of its class. But sometimes **we need to define a class member that can be used without object of its class.**
- The static members are initialized and static methods are executed before any objects of its class are created.

## The static keyword :

- The main() method is static, because it must be executed before any object of that class is created.
- We can also define a block as static. This static block will be executed when the class is loaded.

## The static keyword :

- The static method of other class can be called by the class name in which it is defined. We don’t have to create an object in this case.

## The static keyword :

- There are some rules for static members.
  - The static members can call only static methods and access only static variables.
  - They can not refer to **this** or **super.** (**super** keyword is related to inheritance)

## Def.___ demonstrate use of static.

```
class Print{
   static void print(){
        System.out.println("Hello");
   }
}
```

## Def. Continue...

```
class args{
  static int a=10; //static member...
  static int b=0; //static member...
  static void display(int c){ //static method...
      System.out.println("a= "+a);
      System.out.println("b= "+b);
      System.out.println("c= "+c);
      }
```

## Def. Continue...

```
   static{                              //static block
        System.out.print("Static block executed");
        b=a+10;
        }
public static void main(String args[]){
        display(30);
        Print.print();
   }
}
```

## The final keyword :

- The final keyword can be applied to variables, methods and classes.
- Applying final to method and class is an inheritance concept.
- Declaring a variable as final makes it constant. Therefore the content of final variable can not be modified by any statement in the program.
- Thus the final variables can be used as read only.  We can use them in right side of an expression but can not change them.

## Def.___ demonstrate use of final keyword.

```
class args{
   public static void main(String args[]){
   final double PI=3.14;
   final int SIZE=100;
   System.out.println("Final = Constant");
   System.out.println("PI\t= "+PI);
   System.out.println("SIZE\t= "+SIZE);
   }
}
```

## Nested classes and inner classes :

- We can define a class within another class. These classes are known as nested classes. If a class **B** is defined in class **A**, the class **B** is known as **inner class** and class **A** is known as **outer class**.
- Here, the class B is known to A, but not outside the class A.
- The outer class can access all the members of inner class, even the private member also.

## Def.___ demonstrate use of Nested class.

```
class OuterClass{
  int o=10;
  void test(){
      Inner in=new Inner();
      in.display();
  }
  class Inner{
      int i=100;
      void display(){
          System.out.println("Outer is :"+o);
      }
  }
  void showInner(){
      Inner in=new Inner();
      System.out.println("Inner is :"+in.i);
      }
}
```

```
class args{
  public static void main(String args[]){
      OuterClass outer=new OuterClass();
      outer.test();
      outer.showInner();
  }
}
```

## Garbage Collection :

- When we create an object using new operator, the memory is allocated dynamically to that object.
- So if we create more and more objects, memory is allocated to those objects.
- If any object goes out-of scope, then memory should be deallocated to that object.

## Garbage Collection :

- In some languages like C++ there is destructor to do this.
- In Java, we do not have to do this manually because it does this automatically to destroy objects which are out of scope.
- The process of deallocating memory for objects is known as garbage collection.

## Garbage Collection :

- When there is no reference to an object exists, this object should be destroyed. Thus, when the garbage collection takes place this process of freeing memory is done.
- The garbage collection is done periodically during the execution of the program.

## Garbage Collection :

- There is no any fixed amount of time when it will happen. And even there is no guarantee that garbage collection will occur because in a program there may be one or more objects which are not used.

## Finalize() method :

- Sometimes we will need to do some action when an object is destroyed by garbage collection.
- For example,
  - When your program is using some non-java resource, you have to free the memory for these resources. This process is known as finalization.

## Finalize() method :

- The Finalization is done by the finalize() method. The finalize() method has the form :

```
protected void finalize(){
   // Statements to be executed when finalization occurs
}
```

## Finalize() method :

- Finalize method does not return any value so the return type is void. Protected keyword is access-specifier so that any object of outside class can not access to the method.
- The finalize() method is called just before when Java runtime system performs garbage collection.

## Finalize() method :

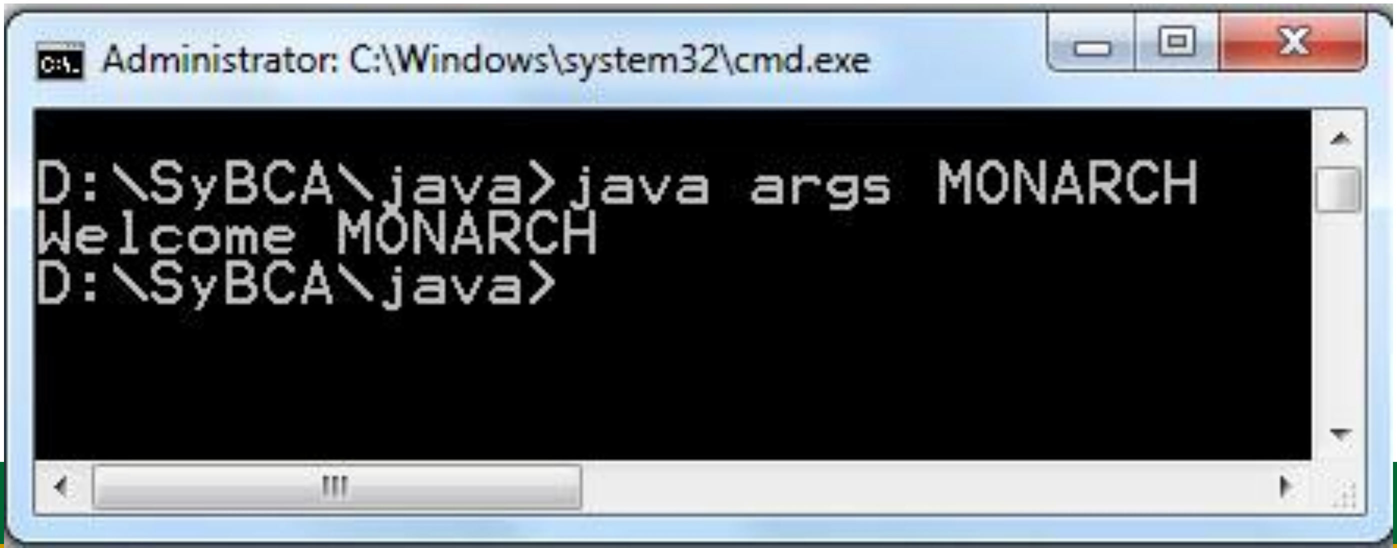
- In C++ we can define a destructor for a class to destroy an object when it goes out of scope. In Java there is no need for this.

## Command-line arguments :

- We can pass arguments to the main() method. This can be done at runtime.
- As we know that the main() method takes array as String as argument, the arguments given to the main() method will be assigned to this array.
- Thus giving command-line argument is quite easy in Java.

## Def.___ WAP to pass a name by command line argument and print a welcome message…

```
class args{
  public static void main(String **args[]**){
      System.out.print("Welcome "+**args[0]**);
  }
}
```


Administrator: C:\Windows\system32\cmd.exe

```
D:\SyBCA\java>java args MONARCH
Welcome MONARCH
D:\SyBCA\java>
```

## Native keyword :

- Java supports native methods. Native methods are those methods which are written in some other language like C or C++.
- Native keyword is used to declare a method as native.
- A simple statement to call a native method is :

```
public native returnType methodName();
```

## Volatile keyword :

- If a variable is modified by volatile keyword, it can be changed unexpectedly by other parts of program such as threads.
- For example
  - In multithreading, there may be more than one threads which share a same instance variable, then this variable can be modified as volatile.

## Transient keyword :

- When an instance variable is specified as transient, then Java runtime system will not write its content to the prersistent storage area, when the object will be saved.
- Example :

```
class transientEx{
   int a; //will be saved
   transient int b; //will not be saved
}
```

## VarArgs :

- In general, our method definition specifies the number of arguments we have to pass. But the VarArgs is a concept in which we can define method that takes variable number of arguments.
- It means that you can pass as many numbers of arguments to a function as you want with no restrictions to match method’s type signature.

## VarArgs :

- To declare VarArgs method,
- Syntax :

```
Access-specifier Return_Type
  Method_name(Data_type... argName)
  {
    //Method Boyd
  }
```

## VarArgs :

- In general, our method definition specifies the number of arguments we have to pass. But the VarArgs is a concept in which we can define method that takes variable number of arguments.
- It means that you can pass as many numbers of arguments to a function as you want with no restrictions to match method’s type signature.

## Def. Demonstrate VarArgs.

```
class args
{ public static void test(String...args)
{ System.out.print("Arguments:"+args.length+"-->");
      for(String s: args)
      {
            System.out.print(s+" ");
      }
      System.out.println();
  }
```

## Def. Demonstrate VarArgs.

```
  public static void main(String args[])
  {  test("Hi", "Hello");
     test("Welcome");
     test("Good", "Bye", "Thank", "You");
     test();
  }
}
```