K. N. Govt. Arts College for Women(A), Thanjavur-7

Department of Computer Science

Subject: Programming with Java

Subject Code: 18K4MACS3

Subject Incharge:

Unit I: N.Baby Kala, Guest Lecturer in Computer Science.

Unit II: K.Sharmila, Guest Lecturer in Computer Science.

Unit III: N.Anuradha, Guest Lecturer in Computer Science.

Course Content

Unit I: Fundamental of OOP: Object Oriented Programming –Basic Concepts- Benefits and Application- Java Evolution-History- Features- Overview of Java Language: Program Structure – Tokens and Statements - Implementing Program – Command Line Arguments.

Unit II: Constants, Variables and Data types: Constants, Variables and Data types – Declaration and Scope of Variables – Symbolic Constant – Type Casting – Operators and Expression: Arithmetic Expression – evaluations of Expression – Type Conversion – Operator Precedence of Associativity.

Unit III: Decision Making and Branching: Simple if, If..Else, nesting of if..else, else if ladder – switch – Ternary operator - Decision Making and Looping : while..do, do..while , for , Jumps in Loops – Labeled – Sample Programs.

Text Book : "Programming with Java- A Primer"- E. Balagurusamy – Mc Graw Hill Education (India) Pvt. Ltd., Fifth edition – Reprint 2015.

FUNDAMENTALS OF OBJECT ORIENTED - PROGRAMMING

INTRODUCTION:

- Object-oriented programming is an approach to development and an organization that attempts to eliminate some of the flaws of conventional programming.
- It involves new ways of organizing and developing programs and does not a concern using a particular language.
- C++ is a procedural language with object oriented extension.
- Java, a pure object oriented language.
- Languages supporting OOP features include Smalltalk, Objective C, C++, Ada, Pascal, and Java.

Definition of OOPS Concepts in Java

"Object-oriented programming is an approach that modularizes programs by creating a partitioned memory area for both functions and data that can be used as templates for creating copies of such modules on demand."

OOPS Paradigm

- The primary objective of the object-oriented approach is to eliminate some of the pitfalls that exist in the procedural approach.
- OOPS allows decomposing a problem into several entities called Objects and then build data and functions from these entities. The combination of the data makes up an object.

Object = Method + Data

- The data of an object is accessed by the methods associated with that object. However, the methods of an object can access methods of other objects.

Features of OOPS

Java is a robust and scalable object-oriented programming language that is based on the concept of objects and classes.

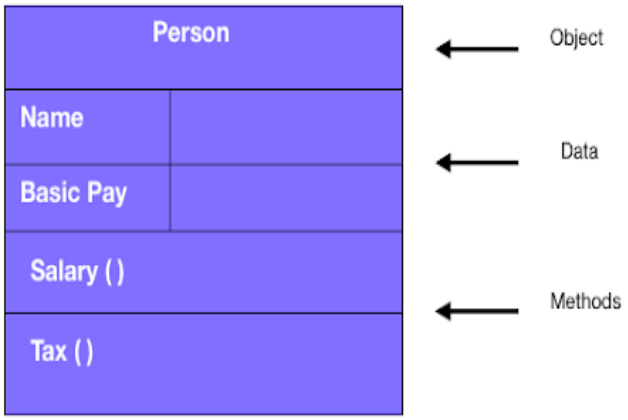
Some features of object-oriented programming in java are:

- Emphasis is on data than procedures
- Programs are divided into objects
- Data Structures are designed to characterize objects.
- Methods operating on the data of an object are tied together in the data structure.
- Data is hidden, and external functions cannot access it.
- Objects communicate with each other through methods
- New methods and data can be easily added whenever necessary
- Follows the bottom-up approach in program design

General OOPS concepts in Java:

Objects and Classes

- **Objects** are runtime entities in an object-oriented system.
- An object can represent a person, a bank account, a place, a table of data.
- Any programming problem is analysed based on objects and how they communicate amongst themselves. The objects interact with each other by sending messages to one another when a program is executed.
- For Example, 'customer' and 'account' are two objects that may send a message to the account object requesting for the balance. Each object contains code and data to manipulate the data. Objects can even interact without knowing the details of each other's code or data.
- The entire set of code and data of an object can be made user-defined data type using the concept of the class. A **class** is a 'data-type' and an object as a 'variable' of that type. Any number of objects can be created after a class is created.
- *The collection of objects of similar types is termed as a class*. For Example, apple, orange, and mango are the objects of the class Fruit. Classes behave like built-in data types of a programming language but are user-defined data types.



Representation of an Object

Data Abstraction and Encapsulation

The wrapping up of the data and methods into the single unit is known as **Encapsulation**. The data is accessible only to those methods, which are wrapped in the class, and not to the outside world.

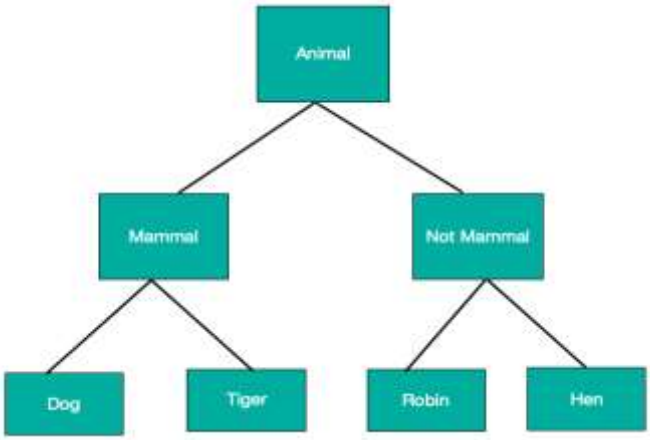
This insulation of data from the direct access of the program is called **Data hiding**.

Encapsulation- Objects as "black-boxes"

Abstraction is the act of reducing programming complexity by representing essential features without including the background explanations or details. Classes wrap or encapsulate all the essential properties of the objects that are to be created.

Inheritance

- Inheritance is the process by which objects of *one class acquire some properties of objects of another class*. Inheritance supports the concept of hierarchical classification.
- In OOP, the idea of inheritance provides the concept of **reusability**. It means that we can add additional features to parent class without modification; this is possible by deriving a new class from the parent class.
- The new class consists of the combined features from both the classes. In Java, the derived class is also known as the **subclass**.
- For Example, a bird Robin is part of the class, not a mammal, which is again a part of the class Animal. The principle behind this division is that each subclass shares common characteristics from the class from its parent class.



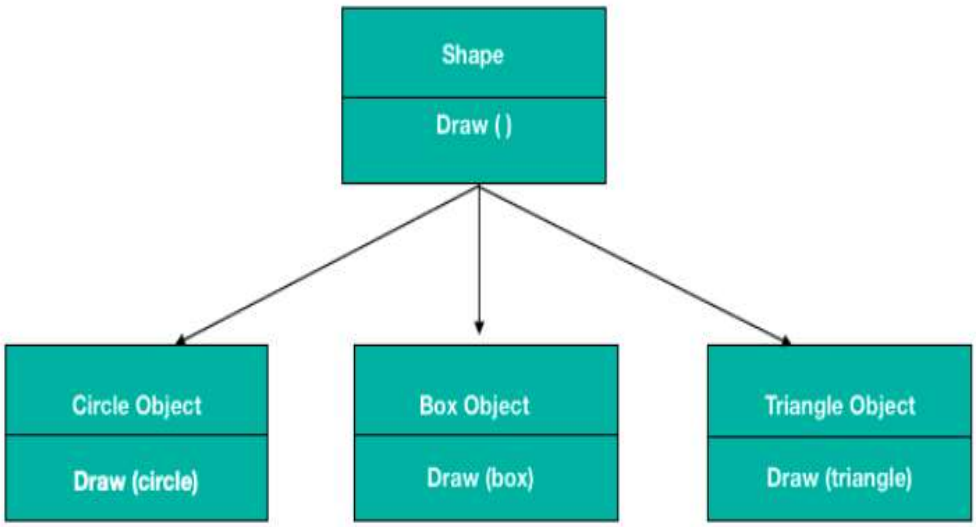
Properties of Inheritance

- ***Types of Inheritance***

- Single
- Multiple
- Multilevel
- Hybrid

Polymorphism

- Polymorphism means the ability to take many forms. For Example, an operation exhibits different behavior in different situations.
- Example: A single function name can be used to handle the different numbers and different types of arguments.



- In polymorphism, objects having different internal structures can share the same external interface.
- Inheritance extensively uses the concept of polymorphism.
- Polymorphism can be achieved in two ways:
  - *Method Overloading*
  - *Method Overriding*

Dynamic Binding

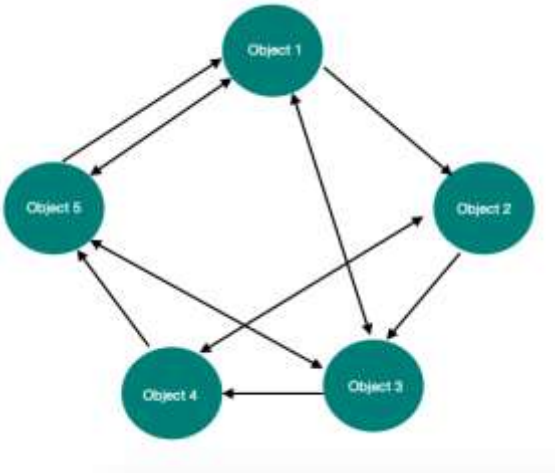
- Binding is the process of linking a procedure call to the code to be executed in response to the call.
- It means that the code associated with the given procedure call is not known until the time of the call at runtime.
- It is associated with inheritance and polymorphism.

Message Communication

- Objects communicate with each other in OOPs. The process of programming in case of OOP consists of the following:

- Creating classes defining objects and their behavior.
- Creating objects
- Establishing communication between objects.

The network of Objects Communicating with Each Other



- Specifying the object name, the method name, and the information to be sent is involved in message passing.

- Objects can be created or destroyed as they have a life cycle. It allows communication between the objects until the objects are alive.

Benefits of OOPs

- Inheritance eliminates redundant code and enables reusability.
- As Message passing allows communication with objects, this presents writing code from scratch every time. It is thus saving development time and higher productivity.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by a code in other parts of the program.
- It is possible to have multiple objects to coexist without any interference.
- It is easy to Partitions work in a project based on classes and objects.
- Systems up-gradation is easy from small to larger system.
- Software complexity can be easily managed.

Applications of OOPs

- Real-time systems
- Simulation and modeling
- Object-oriented databases
- Hypertext and Hypermedia
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office Automation systems
- CIM/CAD/CAD System.

HISTORY

- Java is the general purpose, true object oriented programming language and is highly suitable for modeling the real world and solving the real world problems.
- In the company Sun Microsystem, James Goslings(an employee there), started a project named „GREEN".
- So James Goslings started to develop a new language known as "OAK". In may 1995, Sun officially announced this language at Sunworld 95.Due to some reasons, the name was changed to "JAVA".
- The prime motive of Java was the need for a platform independent language.
- The second motive was to design a language which is "Internet Enable". To run the programs on internet, Green project team come up with the idea of developing "Web Applet". The team developed a web browser called "Hot Java" to locate and run applet programs on internet.
- The most striking feature of Java is "Platform neutralness". Java is the first language that is not tied to any particular hardware or O.S.

JAVA MILESTONES

YEAR	DEVELOPMENT
1990	Sun Microsystem started to develop special software for electronic devices.
1991	James Goslings developed a new language "OAK".
1992	New language Oak was demonstrated in hand held devices.
1993	World Wide Web(WWW)appeared on internet and transformed the text based internet to graphical environment. Sun developed Applets.
1994	Green project team developed a web browser "Hot Java" to run applets on internet.
1995	Oak was renamed to JAVA.
1996	Java established itself as the leader of internet programming. Sun released Java development kit 1.0.
1997	Sun released Java development kit 1.1(JDK 1.1).
1998	Sun released Java2 with version 1.2(SDK 1.2).
1999	Sun released Java2 standard edition (J2SE) and Enterprise Edition (J2EE).
2000	J2SE with SDK 1.3 was released.
2002	J2SE with SDK 1.4 was released.
2004	J2SE with JDK 5.0 was released.

Features of Java

Some following Features of Java are given below:

1. Simple and Small
2. Compiled and Interpreted
3. Platform-Independent and Portable
4. Object-Oriented
5. Distributed
6. Multithreaded
7. Dynamic and Extensible
8. High performance
9. Ease of Development
10. Scalability
11. Monitoring and Manageability
12. Desktop Client

1. Simple and Small : Java is a small and simple language, Many features of C and C++ that are either redundant or sources of unreliable code are not part of Java. Such as Java does not use pointers, preprocessor header files, **goto** statement and many others. It also eliminates operator overloading and multiple inheritances.

2. Compiled and Interpreted : Generally a computer language is either compiled or interpreted. Java combines both these approaches thus making java is a two-stage system. First, Java compiler translates source code into bytecode instruction. Bytecode instructions are not machine instructions. In the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the java program. So, We can say that Java is both Compiled and Interpreted Language.

3. Platform-Independent and Portable : Java ensures portability in two ways, first, Java compiler generates bytecode instructions that can be implemented on any machine. Second, the size of the primitive data types is Machine-Independent. The most significant contribution of java over other languages is its portability. Java programs can be easily moved from one computer system to another.

4. Object-Oriented : Java is a true object-oriented language. Almost everything in Java is an object. All program code and data reside within objects and classes. Java comes with an extensive set of classes, arranged in packages.

5. Distributed : Java is designed as a distributed language for creating applications on networks. It can share both data and programs. Java applications can open and access remote objects on the Internet as easily as they can do in a local system. This enables multiple programs at multiple remote locations to collaborate and work together on a single project.

6. Multithreaded : Multithreaded means that handling multiple tasks simultaneously. Java supports multithreaded programs. This means that we need not wait for the application to finish one task before beginning another.

7. Dynamic and Extensible : Java is a dynamic language, Java is also capable of dynamically linking in new class libraries, methods, and objects. Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program depending on the response.

8. High performance : Java performance is impressive for an interpreted language, it mainly due to the use of intermediate bytecode. According to Sun, Java speed is comparable to the native C/C++. Java architecture is also designed to reduce overheads during run-time.

9. Ease of Development : Java 2 Standard Edition (J2SE) 5.0 supports features – Generics, Enhanced for Loop, Autoboxing or Unboxing, Typesafe Enums, Static import and Annotation. These features reduce the work of a programmer by shifting the responsibility of creating the reusable code to the compiler. The resulting source code is free from bugs because the errors made by the compiler are less when compared to those made by programmers.

10. Scalability : J2SE 5.0 assures a significant increase in scalability by improving the startup time and reducing the amount of memory used in Java 2 runtime environment.

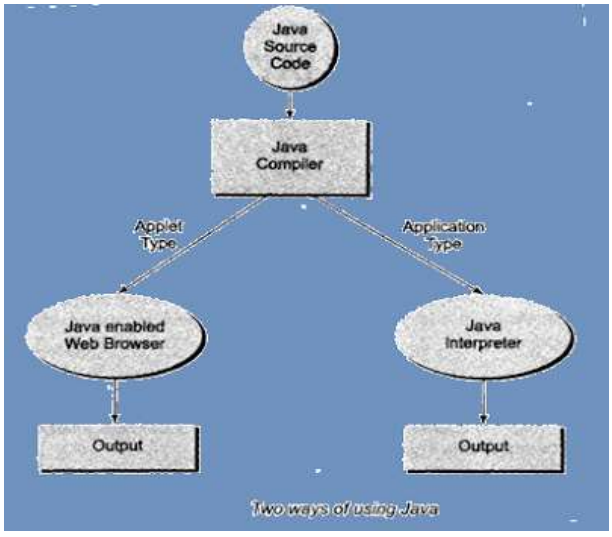
11. Monitoring and Manageability : Java supports several APIs, such as JVM Monitoring and Management API, Sun Management Platform Extension, Logging, Monitoring, and Management Interface, and Java Management Extension (JMX) to monitor and manage Java applications.

12. Desktop Client : J2SE 5.0 provides enhanced features to meet the requirements and challenges of the Java desktop client users. It provides an improved Swing look and feels called Ocean. This feature is mainly used for developing graphics applications that require OpenGL hardware accelera

Overview of Java Language

Java is a general purpose object oriented programming *language. We can develop two types of java programs:*

- Stand – alone applications
- Web applets



Stand – alone applications:-

Stand – alone application are programs written in java to carry out certain tasks on a stand – alone local computer. Executing a stand –alone java program involves two steps:

1. Compiling source code into byte code using javac compiler.
2. Executing the byte code program using java interpreter.

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

1. **class** keyword is used to declare a class in java.
2. **public** keyword is an access modifier which represents visibility. It means it is visible to all.
3. **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. So it saves memory.
4. **void** is the return type of the method. It means it doesn't return any value.
5. **main** represents the starting point of the program.
6. **String[] args** is used for command line argument. We will learn it later.
7. **System.out.println()** is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class. We will learn about the internal working of System.out.println statement later.

```
//Java Program to illustrate the use of Rectangle class which
//has length and width data members
class Rectangle{
 int length;
 int width;
 void insert(int l,int w){
  length=l;
  width=w;
```

```
 }
 void calculateArea(){System.out.println(length*width);}
}
class TestRectangle2{
 public static void main(String args[]){
  Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
  r1.insert(11,5);
  r2.insert(3,15);
  r1.calculateArea();
  r2.calculateArea();
}
}
```

Test it Now

Output:

```
55
45
```

Java Program Structure

A Java program may contain many classes of which only one class defines the main method. Classes contain data members and methods that operate on the data members of the class. Methods may contain data type declaration and executable statements. To write a java program, we first define classes and then put them together. A Java program may contain one or more section are shown as following in Fig:

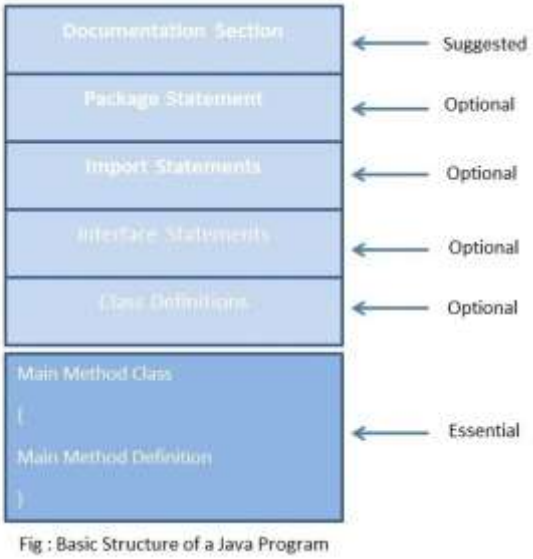


Fig : Basic Structure of a Java Program

Documentation Section :-

The documentation section comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer to at a later stage. Comments must explain why and what of classes and how of algorithms. This would greatly help in maintaining the program. in addition to the two styles of comment like "single line comment" and "double line comment", Java also uses a third style comment /**…..*/ known as documentation comment. This form of comment is used for generating documentation automatically.

Package Statement :-

The first statement allowed in a Java file is a package statement. This statement declares a package name and informs the compiler that the classes defined here belong to this package.

Example:

```
package student;
```

The package statement is optional. That is, our classes do not have to be part of a package.

Import Statement :-

The next thing after a package statement (but before any class definition) may be a number of import statements. This is similar to the #include statement in C. Example,

```
import student.test;
```

This statement instructs the interpreter to load the test class contained in the package student. Using import statement, we can have access to classes that are part of other named packages.

Interface Statement :-

An interface is like a class but includes a group of method declarations. This is also an optional section and is used only when we wish to implement the multiple inheritance feature in the program.

Class Definitions :-

A Java program may contain multiple class definitions. Classes are the primary and essential elements of a Java program. These classes are used to map the objects of real-world problems. The number of classes used depends on the complexity of the problem.

Main Method Class :-

Since every Java stand-alone program requires a main method as its starting point, this class is the essential part of Java program. A simple Java program may contain only this part. The main method creates objects of various classes and establishes communications between them. On reaching the end of main, the program terminates and the control passes back to the operating system.

Java Tokens

A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

1. Keywords
2. Identifiers
3. Constants
4. Special Symbols
5. Operators

Keyword: Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program.

abstract viod boolean
break byte case
catch char class
const continue default
do double else
enum exports extends
final finally float
for goto if

Identifiers: Identifiers are used as the general terminology for naming of variables, functions and arrays.

Examples of valid identifiers :
MyVariable

MYVARIABLE

myvariable

Constants/Literals: Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined. Constants refer to fixed values. They are also called as literals.

Special Symbols: The following special symbols are used in Java having some special meaning and thus, cannot be used for some other purpose. The Special symbols are [] () {}, ; * =.

Operators: Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-

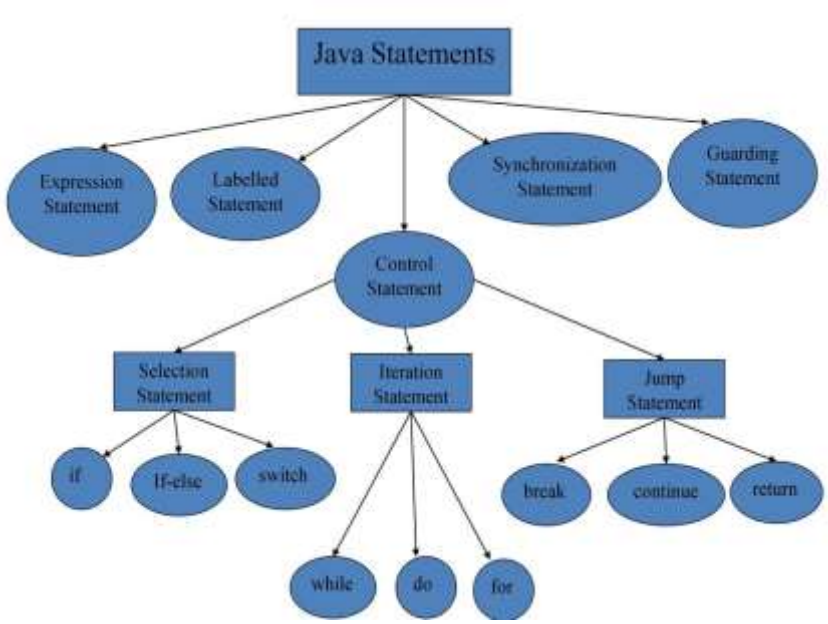
Arithmetic Operators, Unary Operators, Assignment Operator, Relational Operators, Logical Operators Ternary Operator, Bitwise Operators, Shift Operators, instance of operator, Precedence and Associativity

Period (.): It separates the package name form the sub-packages and class. It also separates a variable or method from a reference variable.

Comments: Comments allow us to specify information about the program inside our Java code. Java compiler recognizes these comments as tokens but excludes it form further processing. The Java compiler treats comments as whitespaces. Java provides the following two types of comments:

- **Line Oriented:** It begins with a pair of forwarding slashes (//).
- **Block-Oriented:** It begins with /* and continues until it founds ***/**.

JAVA STATEMENT:



Implementing Java Program

Implementation of a Java application program involves a series of steps. They include :

- **Creating the program**
- **Compiling the program**
- **Running the program**

Creating the program
We can create a program using any text editor. Assume that we have entered the following program :

```
class Javaapp
{
    public static void main(String[] args)
        {       System.out.println("Wlcome to Java");      }
}
```

We must save this program in a file called **Javaapp.java** ensuring that the file name contains the class name properly. This file is called the **source file**. Note that all Java source files will have the extension **java**.

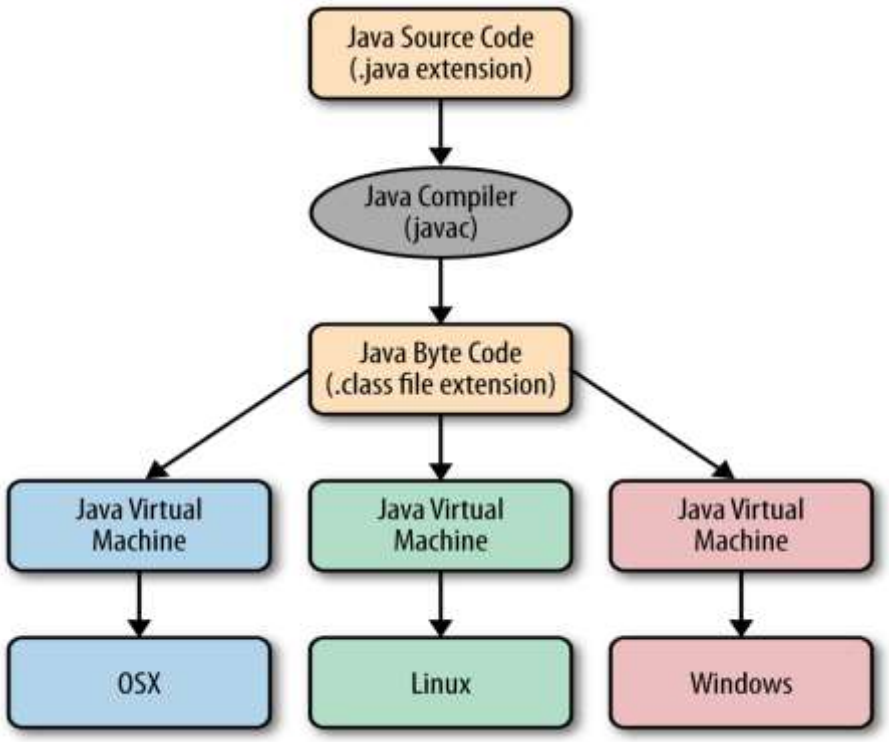
Compiling the program
To compile the program, we must run the Java Compiler **javac**, with the name of the source file on the command line the **javac** compiler creates a file called **Javaapp.class** containing the **bytecodes** of the program. Note that the compiler automatically names the bytecode file as **<classname>.class**. To compile the Program,

Syntax : Javac filename .java

Running the program

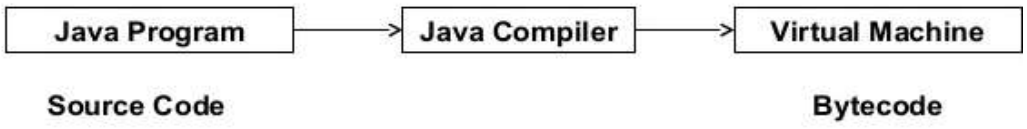
To run the program, we must run the Java interpreter **java**, with the name of the class file on the command line

Syntax: Java Filename

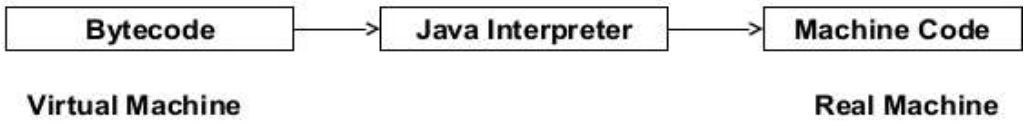


Java Virtual Machine:

All language compilers translate Source code into machine code for a specific computer. The Java compiler produce an intermediate code known as byte code as *Byte code* for a machine that does not exist. The machine is called the Java Virtual Machine (JVM) and its only exist only inside the computer memory.

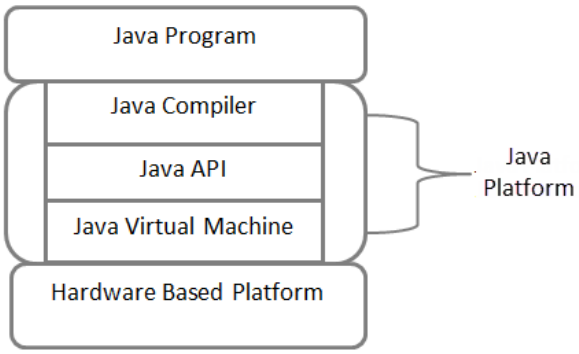


Process of Compilation



Process of Converting bytecode into machine code

The JVM which in turn acts as the intermediary the Operating System and Java Object Frame Work.



Command Line Arguments

Command-line arguments are parameters that are supplied to the application program at the time of invoking it for execution. It may be recalled that the program was invoked for execution.

```
class Test
{
public static void main(String args[])
{
    int count, i=0;
    String string;
    count=arg.length;
    System.out.println("Number of arguments ="+count);
    while(i<count)
            {
               string=args[i];
            i=i+1;
            System.out.println(i+" : "+"Java is"+string+"!");
            }
}
} // The output of the program would be :
Number of arguments=3
```

Output:

```
Java Simple Programming Language!
Java Object-Oriented Programming Language!
```

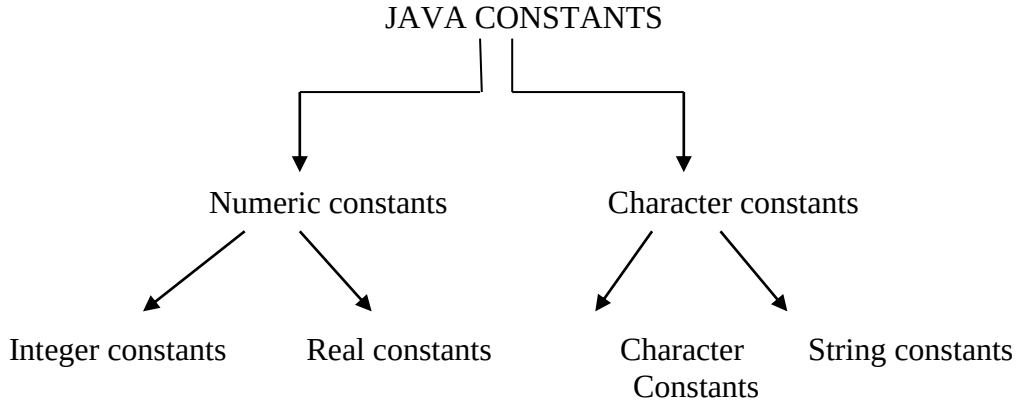
Constants, Variables, and Data types

Introduction:

A Programming Language is designed to process certain kinds of data consisting of numbers, characters and strings and to provide useful output known as information. The task of processing data is accomplished by executing a sequence of instructions constituting a program. These instructions are formed using certain symbols and words according to some rigid rules known as syntax rules.

Constants:

Constants in java refer to fixed values that do not change during the execution of a program. Java supports several types of constants.



Integer Constants

An Integer constant refers to a sequence of digits. There are three types of Integers , namely decimal Integer, Octal Integer and Hexadecimal Integer.

- **Decimal Integer** consists of set of digits, 0 through 9, preceded by optional minus sign. Valid examples of decimal Integer constants are:

  123 -321 0 654321

  Embedded spaces, commas, and non-digit characters are not permitted between digits.
  For example, 15 750 20.000 $1000 are illegal numbers

- **An Octal Integer** consists of any combination of digits from the set 0 through 7, with a leading 0. Some examples of Octal integer are 037 0 0435 0551

- A sequence of digits preceded by 0X considered as **Hexadecimal Integer.** They may also include Alphabets A through F or a through f. A letter A through F represents the numbers 10 through 15. Following are the examples of valid hex integers.

  0X2 0X9F 0Xbcd 0X We rarely use octal and hexadecimal numbers in programming.

Real Constants

Integer numbers are inadequate to represents quantities that vary continuously, such as distances, heights, temperatures and prices and so on. These quantities are represented by numbers containing fractional parts like 17.548. Such numbers are called **real or floating point constants.**
Example: 0.0083 -0.75 435.36

These numbers are shown in decimal notation, having a whole number followed by a decimal point and the fractional part, which is an integer. It is possible that the number may not have digits before the decimal points or digits after the decimal point. That is 215. .95 -.71 are all valid real numbers.

A real number may also expressed in exponential(or scientific)notation. For example, the value 215.65 may be written as 2.1565e2in exponential notation. E2 means multiply by $10^{2.}$

The general form is **Mantissa e exponent**

The **mantissa** is either real number expressed in **decimal notation or an integer**. The **exponent** is an integer with an optional **plus** or **minus** sign. The letter e separating the mantissa and the exponent can be written in either lower case or upper case. Since the exponent causes the decimal point to "float", this notation is said to represent a real number in **floating point form**.
Examples are: 0.65e4 12e-2 1.5e+5 3.18E3 -1.2E-1
Embedded white (blank) space is not allowed, in any numeric constant.

Exponential notation is useful for representing numbers that are either very large or very small in magnitude. For example, 7500000000 may be written as 7.5E9 or 75E8.
A floating point constant may comprise four parts:

- A whole number
- A decimal number
- A fractional part
- An exponent

Single character constants

A single character constant contains a single character enclosed within a pair of single quote marks.
Example: '5' 'X' ';' ''
Note that the character constant '5' is not the same as the number 5. The last constant is a blank space.
String constants

A string constant is a sequence of character enclosed between double quotes. The characters may be alphabets, digits, special characters and blank spaces.
Examples are:
"Hello Java" "2021" "WELL DONE" "?..!" "5+5" "X"

Backslash character constants

Java supports some special backslash character constants that are used in output methods. For example, the symbol `\n' stands for newline character. Each of them represents one character, although they consists of two characters. These characters combination is known as **escape sequences.**

Backslash character constants

Constant	Meaning
'\b'	Back space
'\f'	Form feed
'\n'	New line
'\r'	Carriage return
'\t'	Horizontal tab
'\''	Single quote
'\"'	Double quote
'\\'	backslash

VARIABLES

A variable is an identifier that denotes a storage location used to store a data value. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during the execution of the program.

Some examples of variable names are:

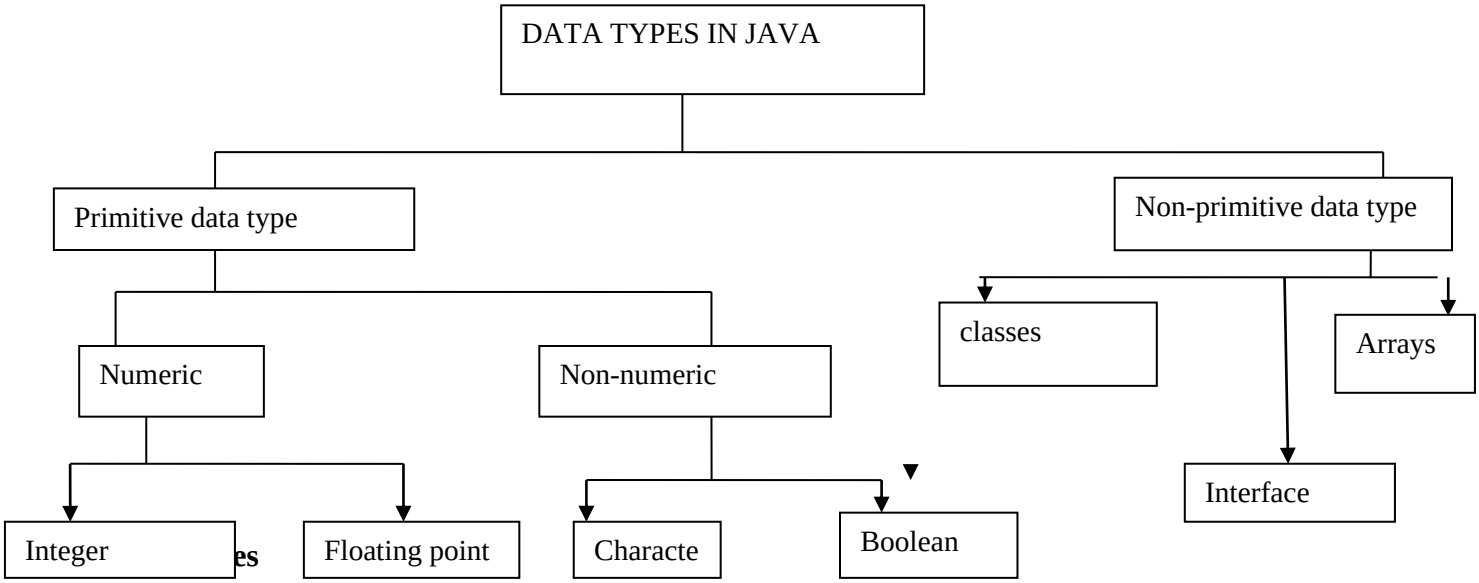
- Average
- Height
- Total -height
- Class Strength

Variable names may consist of alphabets, digits, the underscore(_)and dollar characters, subject to the following conditions:

1 .They must not begin with a digit.
2. Uppercase and lowercase are distinct. This means that the variable Total is not the same as total or TOTAL.
3. It should not be a keyword.
4. White space is not allowed.
5. Variable names can be of any length.

DATA TYPES

Every variable in java has a data type. Data types specify the size and type of values that can be stored. Java language is rich in its data types.



Integer types can holt whole numbers such as 123, -96, 5639 . the size of the values that can be stored depends on the integer data type. Java supports four types of integer

Size and range of integer types

Type	Size	Minimum value	Maximum value
byte	One byte	-128	127
short	Two bytes	-32, 768	32, 767
int	Four bytes	-2, 147, 483, 648	2,147, 483, 647
long	Eight bytes	-9, 223, 372, 036, 854, 775, 808	9, 223, 372, 036, 854, 775, 807

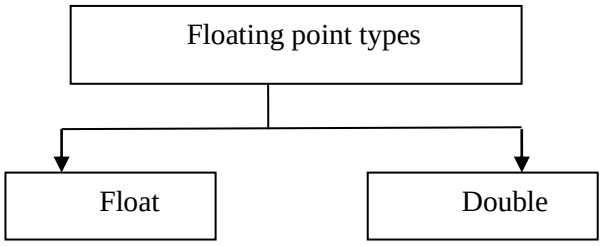
Floating point types

Floating point types contain fractional part such as 27.59, -1.375. There are two kinds of floating point storage. The float type values are single-precision numbers while the double type represents double-precision numbers.

Size and range of Floating point types

Type	Size	Minimum value	Maximum value
Float	4 bytes	3.4e-038	1.7e+0.38
double	8 bytes	3.4e-038	1.7e+308

Floating point data types



Character Type

To store character constant in memory, java provides a character data type called **char.** The char type assumes the size of two bytes, basically, it can hold only a single character.

Boolean Type

We want to test a particular condition during the execution of a program. There are only two values that a Boolean type can take: **true or false** . Boolean type is denoted by the keyword Boolean and uses only one bit of storage.

Declaration of Variables

Variables are name of the storage location variables can be used to store a value of any data type.the general form of declaration of variable is

type variable1, variable2 …variable N ;

Variables are separated by commas. A declaration statement must end with semicolon. Some valid declarations are:

int count;
float x,y;
double pi;
byte b;
char c1,c2,c3;

Giving value to variables

A variable must be given a value after it has been declared but before it is used in an expression. This can be achieved in two ways:

1. By using an assignment statement
2. By using a read statement.

Assignment statement

A simple method of giving value to a variable is through the assignment statement as follows:

variableName = value

For example:
initialvalue = 0;
finalvalue = 100;

yes='x'
string assignment expression is shown below:

```
x=y=z=0;
```

it is also possible to assign a value to a variable at the time of its declaration. This takes the form:

type variablename = value;

Examples:

```
int finalvalue = 100;
char yes = `x`;
double total = 75.36
```

The process of giving initial values to a variable is known as initialization.

```
float x, y, z;    //declares three float variables.
int m= 5, n=10;          //declares and initializes two int variables.
int m,n=10;     //declares m and n and initializes n
```

Read statement

We may also give values to variables interactively through the keyboard using the **readLine()** method as illustrated in program.

Reading data from the keyboard

```
import java.io.DataInputStream;
class Reading
{
Public static void main(String args[])
{
DatInputStream in = new DataInputStream(System.in);
Int intnumber =0;
Float floatnumber= 0.0f;
try
{
System.out.println("Enter an Integer: ");
Intnumber=Integer.parseInt(in.readLine());
System.out.println("Enter a float number:");
floatnumber=  float.valueOf(in.readLine()).floatvalue();
}
Catch(Exception e)
{}
System.out.println("intnumber="+intnumber);
System.out.println("floatnumber="+floatnumber);
}
}
```

The input and output of the program
Enter an integer:
123
Enter a float number:
123.45
Intnumber=123
Floatnumber=123.45

Scope of the variable

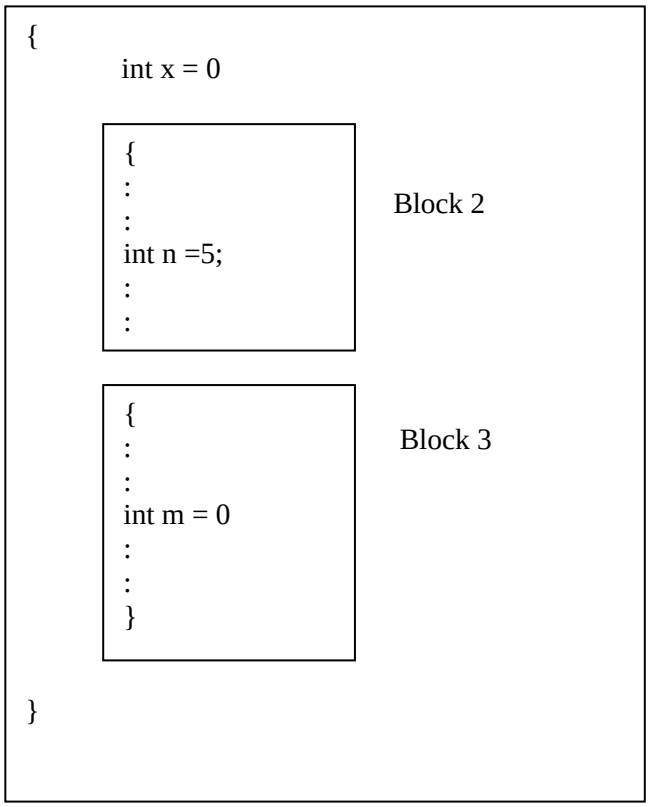
Java variables are actually classified into three kinds

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

Instance and class variables are declared inside the class. Instance variables are created when the objects are instantiated and therefore they are associated with the objects. They take different values for each object. On the other hand, class variables are global to the class and belong to the entire set of objects that the class creates. Only one memory location is created for each class variable.

Variables declared and used inside the methods are called local variables. They are not available for use outside the method definition. Local variables can also be declared inside the program blocks that are defined between an opening brace{ and a closing brace }. These variables are visible to the program only from the beginning of its program block to the end of the program block. When the program control leaves a block, all the variables in the block will cease to exist. The area of the program where the variable is accessible(ie Usable)is called its scope.

Nested program blocks



We can have program blocks within other blocks (called nesting). Each block can contain its own set of local variable declarations. We cannot however, declare a variable to have the same name as one in an outer block. The variable declared x declared in Block1 is available in all the three blocks. However the variable n declared in block2 is available only in block2, because it goes out of the scope at the end of the block2. Similarly , m is accessible only in block3.

Symbolic constants

These constants may appear repeatedly in a number of places in the program. One example of such a constant is 3.142, representing the value of the mathematical constant "pi" . Another example is the total number of students whose mark-sheets are analysed by the "test analysis program"The number of students say 50, may be used for calculating the class total, class average, standard deviation etc. we face two problems in the subsequent use of such programs. They are:

1. Problem in modification of the program
2. Problem in understanding the problem.

Modifiability

We may like to change the value of "pi" from 3.142 to 3.14159 to improve the accuracy of calculations or the number 50 to 100 to process the test result of another class. In both the cases, we will have to search throughout the program and explicitly change the value of the constant wherever it has been used. If any value is left unchanged , the program may produce disastrous outputs.

Understandability

When a numeric values appeared in a program, its use is not always clear, especially when the same value means different things in different places. Example, the number 50 may mean the number of students at one place and the 'pass marks` at another place of the same program. We may forget what a certain number meant, when we read the program some days later.

Assignment of a symbolic name to such constants frees us from these problems. Example, we may use the name STRENGTH to denote the number of students and PASS-MARK to denote the pass marks required in a subject. Constant values are assigned to these names at the beginning of the program. Subsequent use of the names STRENGTH and PASS-MARK in the program has the effect of causing their defined values to be automatically substituted at the appropriate points. A constants is declared as follows

```
final type symbolic-name = value;
```

Valid examples of constant declaration are:

final int STRENGTH = 100;
final int PASS-MARK = 50;
final float PI = 3.14159;

Type Casting

There is a need to store a value of one type into a variable of another type. We must cast the value to be stored by proceeding it with the type name in parentheses. The syntax is:

```
type variable1 =(type)
```

The process of converting one data type to another is called casting. Example:

```
int m=50;
byte n=(byte)m;
```

long count =(long)m;

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

Automatic conversion

It is possible to assign a value of one type to a variable of a different type without a cast. Java does the conversion of the assigned value automatically . this is known as automatic type conversion.
Example: byte b= 75;
int a= b;

Getting values of variable

A computer program is written to manipulate a given set of data and to display or print the results. Java supports two output methods that can be used to send the result to the screen.

- **print()method** //print and wait
- **println() method** //print a line and move to the next line.
  The print() sends the information into the buffer. This buffer is not flushed until a newline character is sent. As a result, the print() method prints output on one line until a newline character is encountered.

Example:

```
System.out.print("Hello");
System.out.print("Java");
```

Will display the words Hello Java on one line and waits for displaying further information on the same line. The next line by printing a newline character as shown below:

```
System.out.print("\n");
```

Example:

```
System.out.print("Hello");
System.out.print("\n");
System.out.print("Java");
```

Will display the output in two lines as follows
Hello
Java

The **println() method by** contrast, takes the information provided and displays it on a line followed by a line feed.

```
System.out.println("Hello");
System.out.println("Java");
```

Will display the output in two lines as follows
Hello
Java

The statement **System.out.println( ); will print a blank line.**

Getting the result to the screen

```
class Displaying
{
public static void main(String args[])
{
System.out.println("Screen Display");
for(int i=1; i<=9; i++)
{
for(int j=1; j<=n; j++)
{
System.out.println(" ");
System.out.println("i");
}
System.out.println("\n");
}
System.out.println("Screen Display Done");
}
}
```

The output will be

```
Screen Display
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
7 7 7 7 7 7 7
8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9
Screen Display Done
```

Standard Default Values

Every variable has a default value. If we don't initialize a variable when it is first created, java provides default value to that variable type automatically.

Default values for various types

Types of variable	Default value
byte	Zero: (byte) 0
short	Zero: (short) 0
int	Zero : 0
long	Zero : 0L
float	0.0f
double	0.0d

char	null character
boolean	false
reference	null

Operators and Expressions

Java supports rich set of operators. we have already used several of them such as = , +, -, and * . an operator is a symbol that tells a computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. Java operators can be classified into

1. Arithmetic operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment and Decrement operators
6. Conditional Operators
7. Bitwise operators
8. Special operators.

Arithmetic operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. Java provides all the basic arithmetic operations.The following table lists the arithmetic operators:

Operator	Result
+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

Arithmetic operators are used as shown below

a+b a-b
a*b a/b
a%b -a*b

here a and b may be variables or constants and are known as operands

Integer Arithmetic

When both the operands in a single arithmetic expression such as a+b are integers. The expression is called integer expression. And the operation is called integer arithmetic. Integer arithmetic

always yields an integer value.if a=14 and b=4 we have the following results:
a-b=10
a+b=18
a*b=56
a/b=3(decimal part truncated)
a%b=2(remainder of integer division)

Real arithmetic

An arithmetic operation involving only real operands is called real arithmetic. A real operand may assume values either in decimal or exponential notation.

Floating point arithmetic

```
class FloatPoint
{
public static void main(String args[])
{
float a = 20.5f, b=6.4f;
System.out.println(“a=” +a);
System.out.println(“b=” +b);
System.out.println(“a+b=” +(a+b));
System.out.println(“a-b=” + (a-b));
System.out.println(“a*b=” +(a*b));
System.out.println(“a/b=” +(a/b));
System.out.println(“a%b=” +(a%b));
}
}
```

The output is given below
a=20.5
b=6.4
a+b=26.9
a-b=14.1
a*b=131.2
a/b=3.20313
a%b=1.3

Mixed-mode Arithmetic

When one of the operand is real and the other is integer, the expression is called mixed-mode arithmetic expression. If either operand is of the real type, then the other operand is converted into real and the real arithmetic is performed. The result will be a real.
15/10.0 produces the result 1.5
15/10 produces the result 1

Relational operators

The *relational operators* determine the relationship that one operand has to the other.
The relational operators are shown here:

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

The general form is:

ae-1 relational operator ae-2

Implementation of relational operators

```
class RelationalOperators
{
public static void main(string args[])
{
float a=15.0f, b=20.75f, c=15.0f;
System.out.println(“a=” +a);
System.out.println(“b=” +b);
System.out.println(“c=” +c);
System.out.println(“a<b is ” +(a<b));
System.out.println(“a>b is” +(a>b));
System.out.println(“a==c is” +(a==c));
System.out.println(“a<=c is” + (a<=c));
System.out.println(“a>=b is” +(a>=b));
System.out.println(“b!=c” +(b!=c));
System.out.println(“b==a+c is” +(b==a+c));
```

The output of the program

```
a=15
b=20.75
c=15
a<b is true
a>b is false
a==c is true
a<=c is true
a>=b is false
a!=c is true
b==a+c is false
```

Logical Operators

Java has three logical operators. The logical operators && and || are used when we want to form compound condition by combining two or more relations.

An example is: a>b && x==10

An expression of this kind which combines two or more relational expressions is termed as logical expression or a compound relational expression. A logical expression also yields a value of true or false according to the truth table.

Logical operators

Operator	Meaning
&&	logical AND
||	logical OR
!	logical NOT

Truth table

Op-1	Op-2	Op-1&&op-2	Op-1||op-2
True	True	True	true
True	False	False	true
False	True	False	true
False	False	False	false

The Assignment Operator

Assignment operators are used to assign the value of an expression to a variable. we have seen the usual assignment operator" =". Java has a set of shorthand assignment operators which are used in the form

V op = exp

Where v is a variable, exp is an expression and op is a java binary operator. The operator op=is known as the shorthand assignment operator.

The assignment statement

v op=exp; is equivalent to v=v op (exp); with v accessed only once.

Example : x + = y+1;

This is same as the statement

x=x+ (y+1);

Increment and decrement operator

Java has two very useful operators not generally found in many other languages. These are the increment and decrement operators:

++ and –

The operator ++ adds 1 to the operand while – subtracts 1. Both are unary operators and are used in the following form:

++m; or m++;

--m; or m--;

++m; is equivalent to m=m+1; (or m+=1)
--m; is eqiuivalent to m = m-1; (or m-=1)
We use the increment and decrement operators extensively in for and while loops.
While ++m and m++ mean the same thing when they form statement independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement.
Consider the following
m=5;
y=++m;
in this case, the value of y and m would be 6. Suppose , if we rewrite the above statement as
m=5;
y= m++;
then, the value of y would be 5 and m would be 6. A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left. On the other hand, a postfix operator first assigns the value to the variable on left and then increment the operand.

Increment operator illustrated

```
class incrementoperator
{
public static void main(String args[])
{
int m=10, n=20;
System.out.println("m=" +m);
System.out.println("n" +n);
System.out.println("++m="+ +m);
System.out.println("n++=" +n++);
System.out.println("m=" +m);
System.out.println("m=" +m);
}
}
```

Output will be
m=10
n=20
++m=11
n++=10
m=11
n=21

conditional operator

The character pair ?: is a ternary operator available in java. This operator is used to construct conditional expression of the form

exp1 ?exp2 : exp3

Where exp1,exp2 and exp3 are expressions.
exp1 is evaluated first. If it is nonzero(true) , then the expression exp2 is evaluated and becomes the value of conditional expression. Note that only one of the expressions are evaluated first.
Example: a=10;

b=15;

x= (a>b) ? a : b ; in this example, x will be assigned the value of b. this can be achieved using the if…else statement as follows:

```
if(a>b)
x=a;
else
x=b;
```

Bitwies operators

This operators are used for testing the bits, or shifting them to the right or left. This operator may not be applied to float or double.

Operator	Result
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left

Special Operators

Java supports some special operators of interest such as instanceof operator and member selection operator(.)

Instanceof operator

The instanceof is an object reference and returns true if the object on the left-hand side is an instance of the class given on the right-hand side. This operator allows us to determine whether the object belongs to a particular class or not.

Example:

person instanceof student

is true if the object person belongs to the class student; otherwise it is false.

Dot operator

The dot operator(.) is used to acess the instance variable and methods of class objects.

Example:

```
person1.age//reference to the variable age
person1.salary()//reference to the method salary()
```

Arithematic expressions

An arithematic expression is a combination of variables, constants, and operators arranged as per the syntax of the language.

Expressions

Algebric expression	java expression
ab-1	a*b-1
(m+n)(x+y)	(m+n)*(x+y)
ab/c	a*b/c
$3x^2+2x+1$	3*x*x+2*x+1

Evaluation of expressions

Expressions are evaluated using an assignment statement of the form

```
variable=expression
```

Variable is any valid java name. when the statement is encountered, the expression is evaluated first and the result then replaces the variable on the left-hand side. All the variable used in the expression must be assigned values before evaluation is attempted. Examples of evaluation statements are

```
x=a*b-c
y=b/c*a;
z=a-b/c+d;
```

Precedence and arithmetic operators

An arithmetic expression without any parenthesis will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in java:

High priority * / %
Low priority + -

The basic evaluation procedure includes two left-to-right passes through the expression. During the first pass, the high priority operators(if any) are applied as they are encountered.

During the second pass, the low priority operator(if any) are applied as they are encountered. Consider the following evaluation statement:

x=a-b/3+c*2-1

when a=9, b=12, and c=3, the statement becomes

x=9-12/3+3*2-1 and is evaluated as follows:

second pass

step3: x= 5+6-1

step4: x=11-1

step5: x=10

the order of evaluation can be changed by introducing parenthesis into an expression

first pass:

step1: 9-12/6*(2-1)

step2: 9-12/6*1

second pass

step3: 9-2*1

step4: 9-2

third pass

step5: 7

the procedure consists of three left to right passes. However, the number of evaluation steps remain the same as 5.

Parenthesis may be nested, and in such cases, evaluation of expression will proceed outward from the innermost set of parenthesis. Make sure that every opening parentheses has a matching closing one. For example

9-(12/(3+3)*2)-1=4

Whereas 9-((12/3)+3*2)-1= -2

Type conversion in Expressions

Automatic Type Conversion

Java permits mixing of constants and variables of different types in an expression, but during evaluation it adheres to very strict rules of type conversion. We know that the computer, considers one operator at a time, involving two operands are of different type, the 'lower' type is automatically converted to the 'higher' type before the operation proceeds. The results is of the higher type.

If byte short and int variables are used in an expression, the results is always promoted to int, to avoid overflow. If a single long is used in the expression, the whole expression is promoted to long.

Remember that all integer values are considered to be int unless they have the 1 or L appended to them.

If an expression contains a float operand, the entire expression is promoted to float. If any operand is double, result is double.

Automatic Type conversion

	char	byte	short	int	long	float	double
char	int	int	int	int	long	double	double
byte	int	int	int	int	long	double	double
short	int	int	int	int	long	double	double
int	int	int	Int	int	long	double	double
long	long	long	long	long	long	double	double
float	float	float	float	float	float	double	double
double	double	double	double	double	double	double	double

The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it. However , the following changes are introduced during the final assignment.

1. Float to int causes truncation of the fractional part.
2. Double to float causes rounding of digits.
3. Long to int causes dropping of the excess higher order bits.

Casting a value

Java performs type conversion automatically. However, there are instances when we want to force a type conversion in a way that is different from the automatic conversion. Consider, for example, the calculation of ratio of female to males in a town.

ratio =female _ number / male _ number

Since female _ number and male _ number are declared as integers in the program, the decimal part of the result of the division would be lost ratio would not represent a correct figure. This problem can be solved by converting locally one of the variables to the floating points as shown below;

ratio=(float) female _number / male _ number

The operator (float) converts the female _ number to floating points for the purpose of evaluation of

the expression. Then using the rule of automatic conversion, the division is performed in foalting point mode, thus retaining the factional part of result.

Note that (float) converts the female _ number to floating point for the purpose of evaluation of the expression. Then using the rule of automatic conversion, the division is performed in floating point mode, thus retaining the fractional part of result.

Note that in no way dose the operator (float) affect the value of the variable female _ number. And also, the type of female _ number remains as int in theother parts program.

The process of such a local conversion is known as casting value. The general form of a cast is;

(Type _ number) expression

Where type- name is one of the standard data types. The expression may be a constant, variable or an expression.

Examples:

x= (int) 7.5 *7.5 is converted to integer by truncation.*

y=(int) (a+b) *the result of a+b is converted to integer.*

z= (int) a+b *a is converted to integer and then added to b.*

Casting can be used to round – off a given value to an integer . consider the following statement;

x = (int) (y + 0.5) ;

If y is 27.6, y + 0.5 is 28.1 and casting, the results becomes 28, the value that is assigned to x. Of course, the expression being is not changed.

When combining two different types of variables in an expression, never assume the rules of automatic conversion. It is always a good practice to explicitly force the conversion. It is more safer. For example, when y and p are double and m is int, the following two statements are equivalent.

Y=p + m;

Y=p+ (double) m;

Operator precedence and associativity

This precedence is used to determine hoe an expression involving more than one operator is evaluated. There are distinct level of precedence and an operator may belong to one of the levels.the operators at the higher level of precedence are evaluated first. The operators of the same precedence are evaluated either from left to right or from right to left, depending on the level. This is known as the associativity property of an operator.

Mathematical Functions

Mathematical functions such as cos, sqrt, log, etc., are frequently used in analysis of real-life problems. Java supports these basic math functions through math class defined in the java.lang package. These functions should be used as Math.function_name()

Example:

Double y=Math.sqrt(x);

Math functions

sin(x)	Returns the sine of angle x in radians
cos(x)	Returns the cosine of angle x in radians
tan(x)	Returns the tangent of angle x in radians
asin(y)	Returns the angle whose sine is y
acos(y)	Returns the angle whose cosine is y
atan(y)	Returns the angle whose tangent is y

Decision Making and Branching:

decision making statements in Java:

Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Branching:

When a Program breaks the sequential flow and jumps to another part of the code ,it is called branching.There are Two types of Branching .

1.**Conditional Branching**

When a program breaks the sequential flow and jumps to another part of the code,it is called conditional Branching.

2.**Unconditional Branching**

If Branching takes place without any decision,it is known as unconditional Branching.

There are the 5 ways of exercising decision making in Java:
1. if
2. if-else
3. nested-if
4. else-if-Ladder
5. switch-case

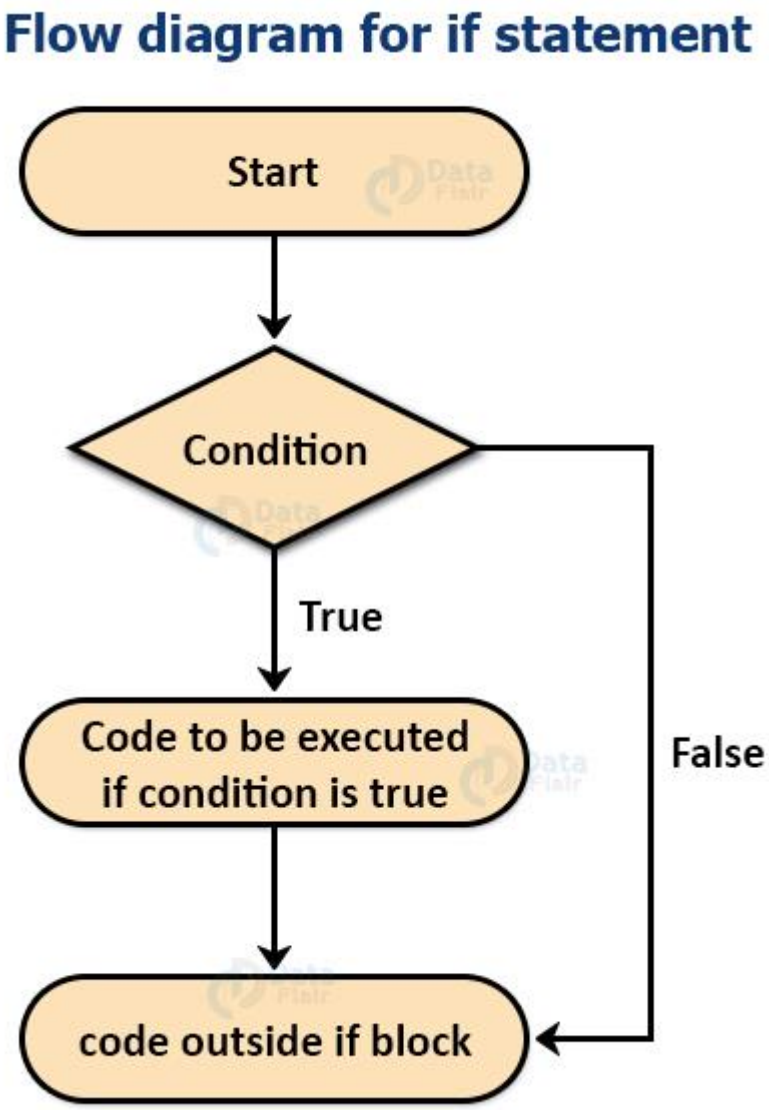
1. If Statement in Java

Java if statement is the simplest decision making statement. It encompasses a boolean condition followed by a scope of code which is executed only when the condition evaluates to true. However if

there are no curly braces to limit the scope of sentences to be executed if the condition evaluates to true, then only the first line is executed.

Syntax:

```
if(condition)
{
//code to be executed
}
```

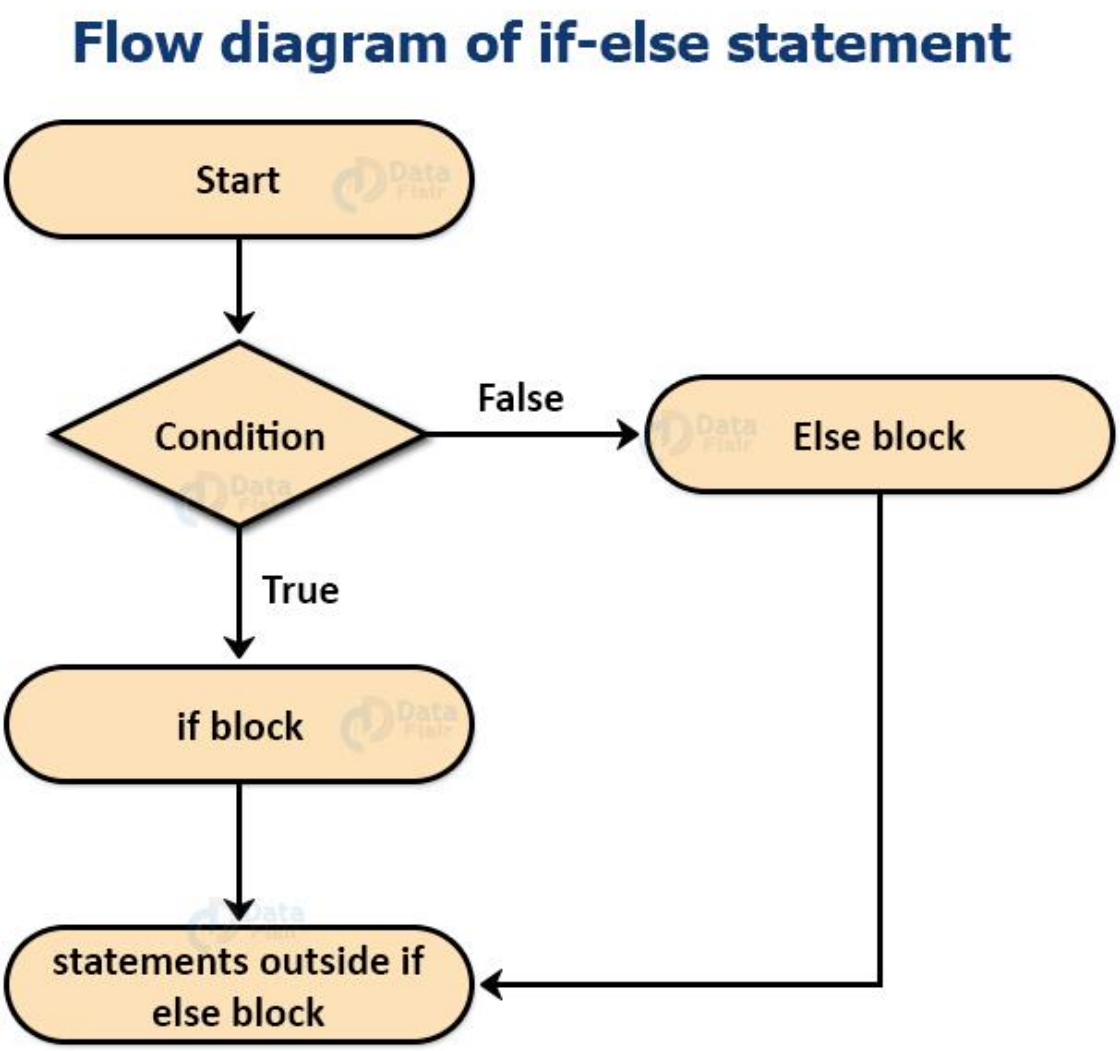


2. **if else statement in Java**

This pair of keywords is used to divide a program to be executed into two parts, one being the code to be executed if the condition evaluates to true and the other one to be executed if the value is false.

However if no curly braces are given the first statement after the if or else keyword is executed.

```
if(condition)
{
//code to be executed if the condition is true
}
else
{
//code to be executed if the condition is false
}
```



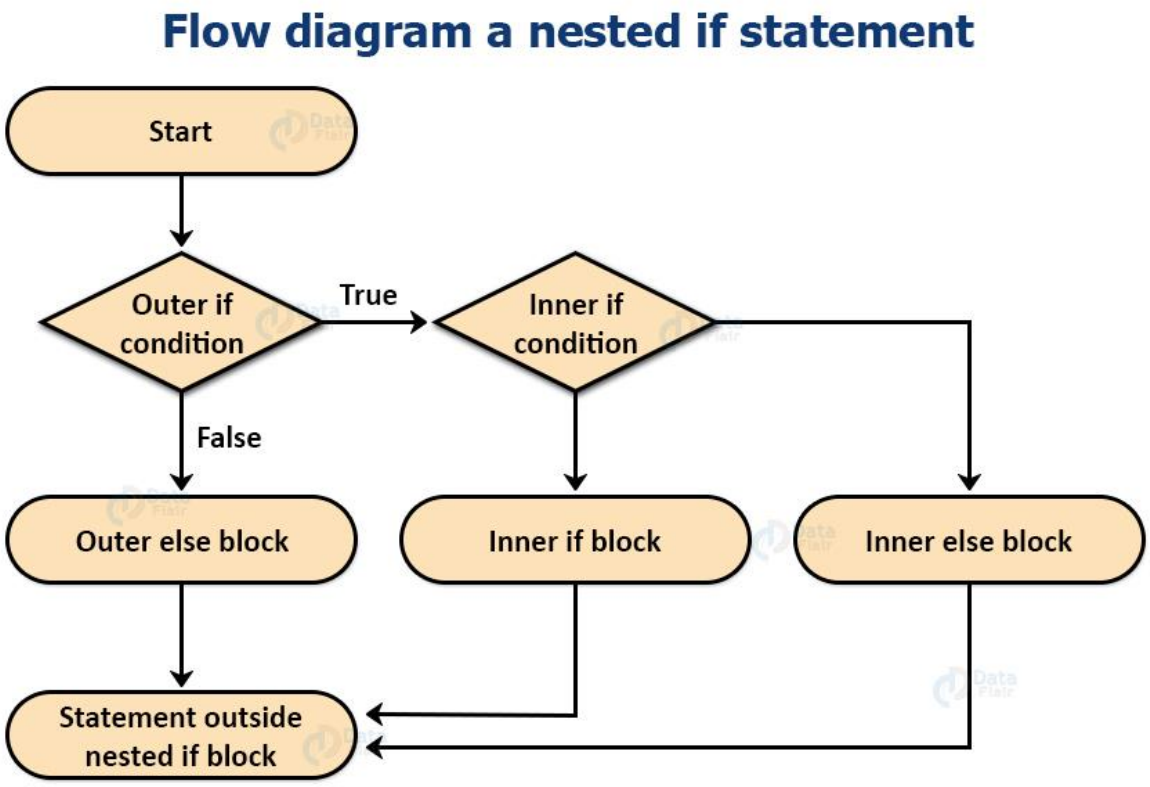
3. **Nested if Statements in Java**

- The nested if is similar to the nested loops we learnt about in the previous chapters. If the condition of the outer if statement evaluates to true then the inner if statement is evaluated.
- Nested if's are important if we have to declare extended conditions to a previous condition

Syntax:

```
if(condition)
{
//code to be executed
if(condition)
{
//code to be executed
}
}
```

Flow diagram of Java nested if statement:



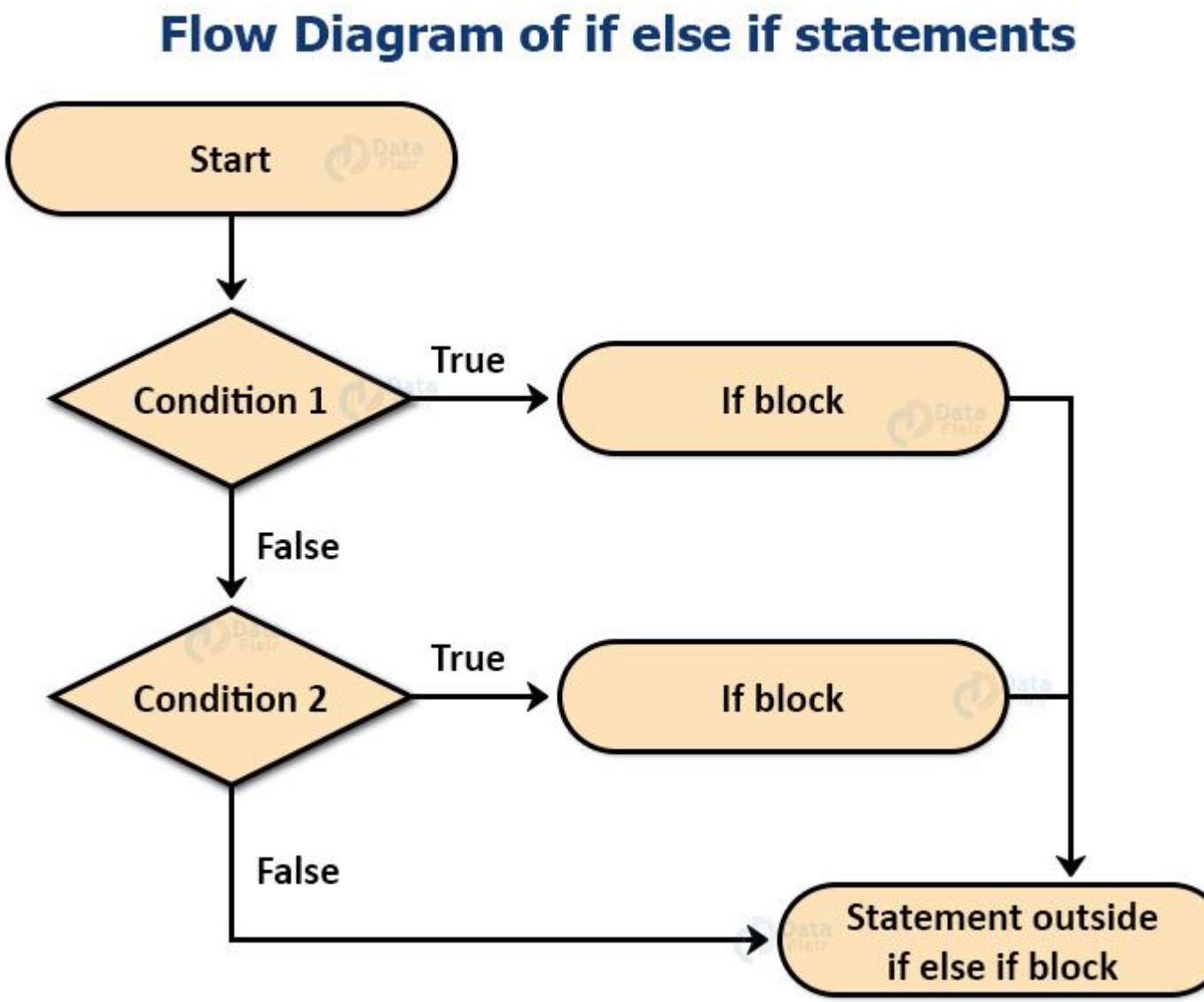
4. else-if Ladder Statements in Java

These statements are similar to the if else statements . The only difference lies in the fact that each of the else statements can be paired with a different if condition statement. This renders the ladder as a multiple choice option for the user. As soon as one of the if conditions evaluates to true the equivalent code is executed and the rest of the ladder is ignored.

Syntax:

```
if
{
//code to be executed
}
else if(condition)
{
//code to be executed
}
else if(condition)
{
//code to be executed
}
else
{
//code to be executed
}
```

Flow Diagram of Java else if Ladder statements:



5. **Switch Statement in Java**

Java switch statement is a different form of the if else if ladder statements.

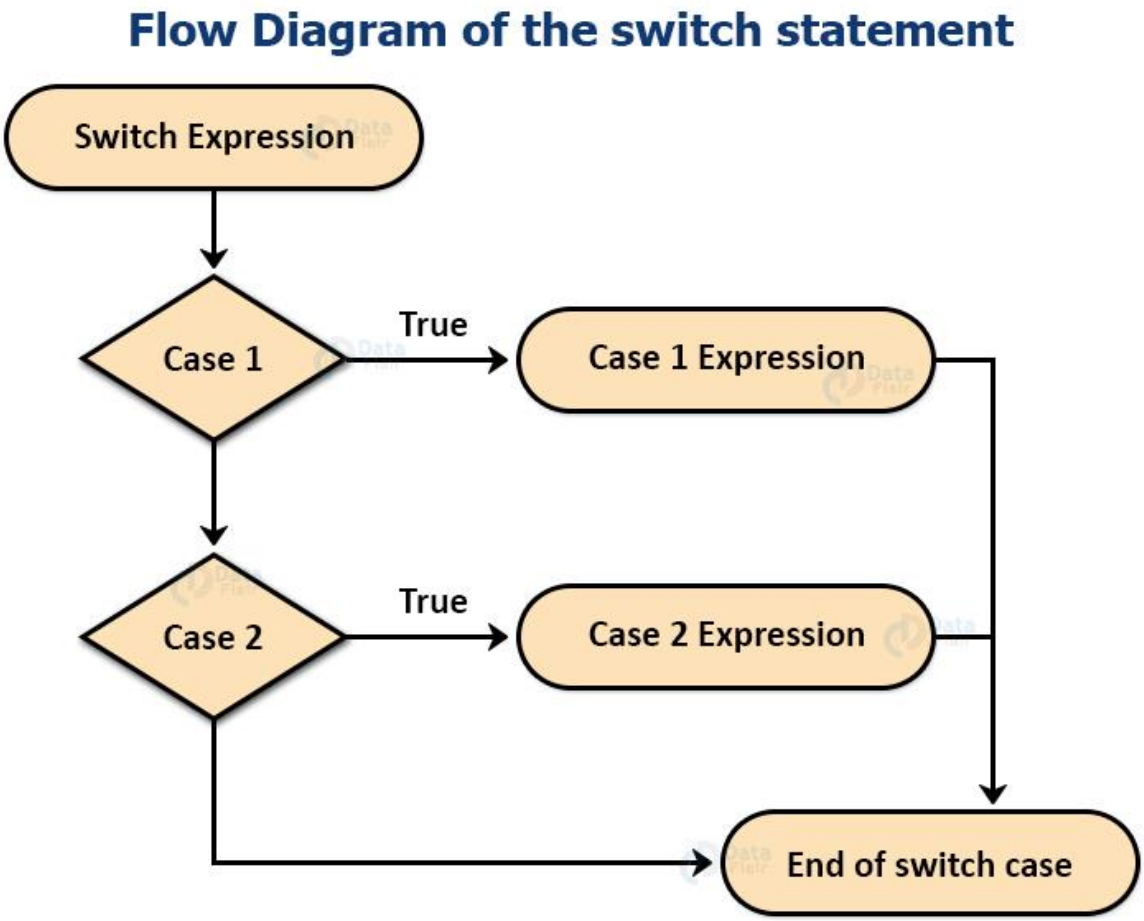
- It branches the flow of the program to multiple points as and when required or specified by the conditions.
- It bases the flow of the program based on the output of the expression.
- As soon as the expression is evaluated, the result is matched with each and every case listed in the scope. If the output matches with any of the cases mentioned, then the particular block is executed.

- A break statement is written after every end of the case block so that the remaining statements are not executed.
- The default case is written which is executed if none of the cases are the result of the expression. This is generally the block where error statements are written.

Syntax:

```
switch(expression)
{
case <value1>:
//code to be executed
break;
case <value2>:
//code to be executed
break;
default:
//code to be defaultly executed
}
```

Flow Diagram of Java switch statement:



The Conditional Operator(?: Operator)

Java Conditional Operator or ternary operator minimizes and mimics the if else statement. It consists of a condition followed by a question mark(?). It contains two expressions separated by a colon(:). If the condition evaluates to true, then the first expression is executed, else the second expression is executed.

Syntax :
(Condition)?(expression 1):(expression 2);

For example, the segment
If(x<0)
    Flag=0;
else
    Flag=1;
Can be written as flag=(x<0)?0:1;

Decision making and Looping

In computer programming, loops are used to repeat a block of code. For example, if you want to show a message 100 times, then rather than typing the same code 100 times, you can use a loop.

In Java, there are three types of loops.

- for loop
- while loop
- do...while loop

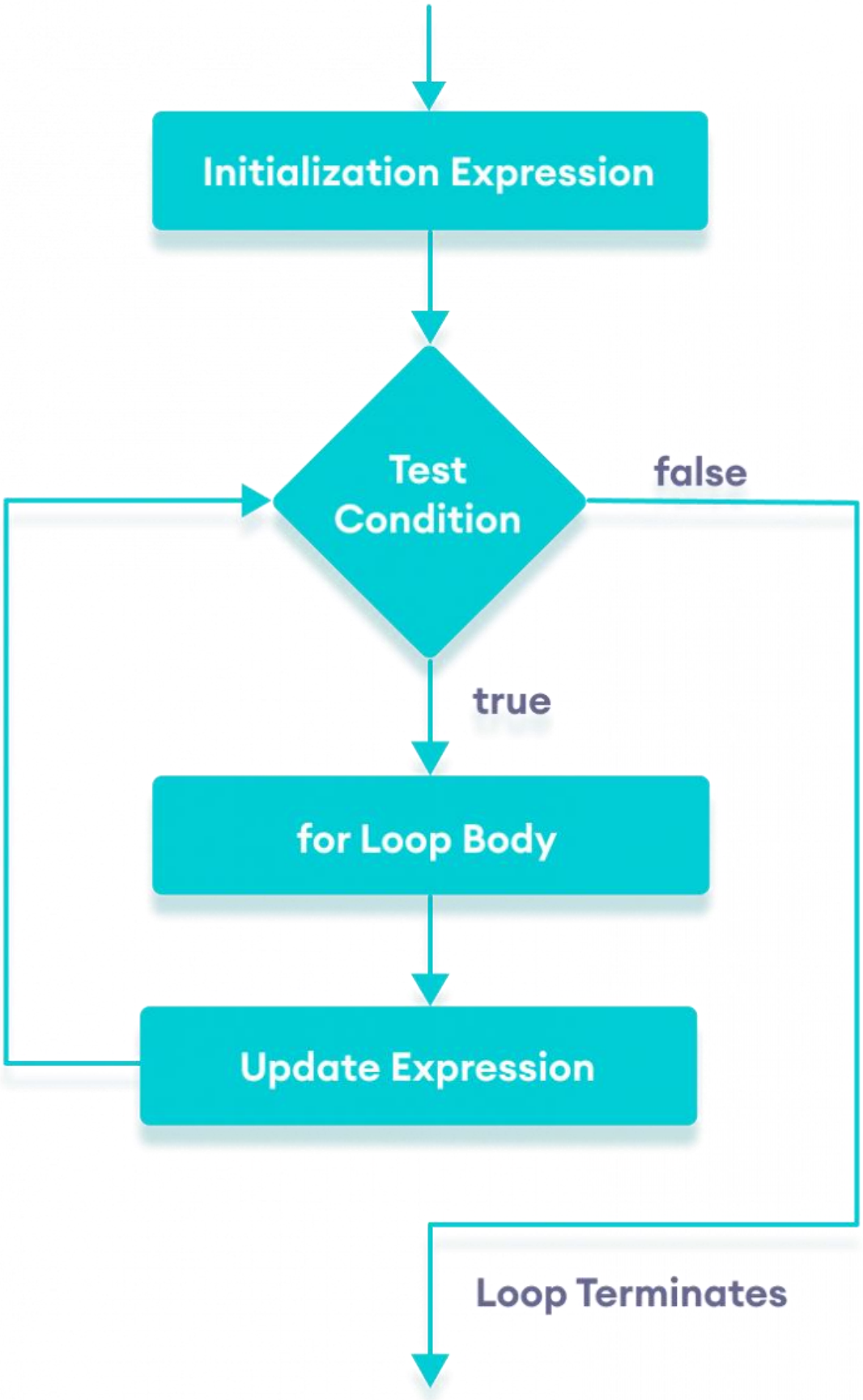
for Loop

Java `for` loop is used to run a block of code for a certain number of times. The syntax of `for` loop is:

```
for (initialExpression; testExpression; updateExpression)
{
   // body of the loop
}
```

Here,

1. **The** initialExpression **initializes and/or declares variables and executes only once.**
2. **The** condition **is evaluated. If the** condition **is `true`, the body of the `for` loop is executed.**
3. **The** updateExpression **updates the value of** initialExpression**.**
4. **The** condition **is evaluated again. The process continues until the** condition **is `false`.**


Initialization Expression
Test Condition
false
true
for Loop Body
Update Expression
Loop Terminates

Example 1: Display a Text Five Times

```
// Program to print a text 5 times

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

    int n = 5;
    // for loop
    for (int i = 1; i <= n; ++i) {
      System.out.println("Java is fun");
    }
  }
}
```

Output

```
Java is fun
Java is fun
Java is fun
Java is fun
Java is fun
```

While Loop

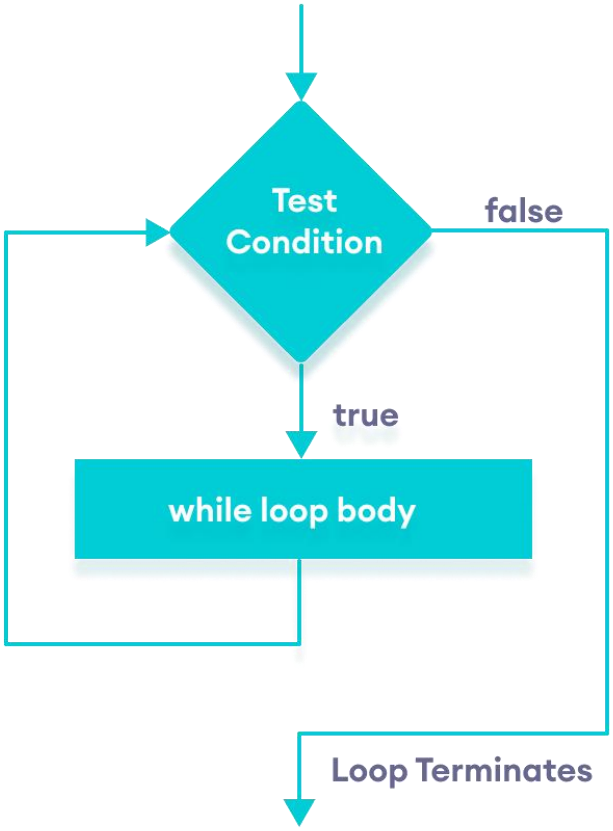
Java while loop is used to run a specific code until a certain condition is met. The syntax of the while loop is:

```
while (testExpression) {
   // body of loop
}
```

Here,

1. A while loop evaluates the **textExpression** inside the parenthesis ().
2. If the **textExpression** evaluates to true, the code inside the while loop is executed.
3. The **textExpression** is evaluated again.
4. This process continues until the **textExpression** is false.
5. When the **textExpression** evaluates to false, the loop stops.

Flowchart of while loop



Example 1: Display Numbers from 1 to 5

```
// Program to display numbers from 1 to 5

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

    // declare variables
    int i = 1, n = 5;

    // while loop from 1 to 5
    while(i <= n) {
```

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

Output

```
1
2
3
4
5
```

Java do...while loop

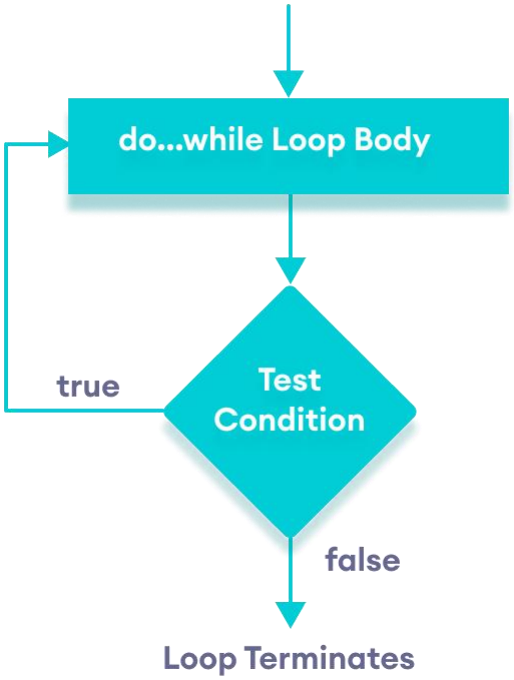
The do...while loop is similar to while loop. However, the body of do...while loop is executed once before the test expression is checked. For example,

```
do {
   // body of loop
} while(textExpression)
```

Here,

1. The body of the loop is executed at first. Then the **textExpression** is evaluated.
2. If the **textExpression** evaluates to true, the body of the loop inside the do statement is executed again.
3. The **textExpression** is evaluated once again.
4. If the **textExpression** evaluates to true, the body of the loop inside the do statement is executed again.
5. This process continues until the **textExpression** evaluates to false. Then the loop stops.

Flowchart of do...while loop



Example 3: Display Numbers from 1 to 5

```
// Java Program to display numbers from 1 to 5

// Program to find the sum of natural numbers from 1 to 100.

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

    int i = 1, n = 5;

    // do...while loop from 1 to 5
    do {
      System.out.println(i);
      i++;
    } while(i <= n);
  }
}
```

Output

1
2
3
4
5

Jumps in Loop

- The jumping statements are the control statements which transfer the program execution control to a specific statements.
- **jump**: **Java** supports three **jump** statement: break, continue and return. These three statements transfer control to other part of the program. Break: In **Java**, break is majorly used for: Terminate a sequence in a switch statement.

Java Break Statement

- We can use break statement in the following cases.
- Inside the switch case to come out of the switch block.
- Within the loops to break the loop execution based on some condition.
- Inside labelled blocks to break that block execution based on some condition.

The break cannot be used outside the loops and switch statement

Java Continue Jumping Statement

- This statement is used only within looping statements.
- When the continue statement is encountered, then it skip the current iteration and the next iteration starts.
- The remaining statements in the loop are skipped. The execution starts from the top of loop again.
- We can use continue statement to skip current iteration and continue the next iteration inside loops.

Labeled Loops

We can give a label to a block of statements.A label is any valid java variable name.To give a label to a loop,place it before the loop with a colon at the end.

For ex.

```
Outer:for(int m=1;m<11;m++)
{
For(int n=1;n<11;n++)
{
System.out.print(“”+m*n );
If (n==m)
Continue outer;
}
}
```