CHAPTER 1 - OBJECT ORIENTED CONCEPTS

With the increasing complexity of software ,it is just not enough to put together a sequence of programming statements and sets of procedures ,modules. We need to use sound construction techniques such as modular programming, top down programming, bottom up programming and structured programming.

With the advent of languages such as C, structured programming became very popular in 1980s. It proved to be a powerful tool that enabled programmers to write moderately complex programs fairly easy. However as the programs grew larger, the structured approach failed to show the desired results in terms of bug free, easy to maintain and reusable programs.

Object oriented programming (OOP) is an approach to program organization and development, which attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several new concepts. It is a new way of organizing and developing programs and has nothing to do with any particular language. Languages that support OOP features include Smalltalk, Objective C, C++, Ada and Object Pascal. C++, an extension of C language, is the most popular OOP language today. C++ is basically a procedural language with object oriented extension. The latest one added to this list is Java, a pure object-oriented language.

1.1 BASIC CONCEPTS OF OBJECT-ORIENTED PROGRAMMING

The major object-oriented approach is to eliminate some of the flaws encountered in the procedural approach. OOP treats data as critical element in the program development and does not allow it to flow freely around the system.

Oops allows us to decompose a problem into a number of entities called objects and then build data and functions (known as methods in Java) around these entities.

It ties data more closely to the functions that operate on it & protects it from accidental modifications from outside functions.

Objects may communicate with each other through functions. i.e. functions of one object can access the functions of other object but data of an object can be accessed only by the functions associated with the object.

An Object is considered to be a partitioned area of computer memory that stores data & set of operations that can access that data. Since memory partitions are independent the objects can be used in a variety of different programs without modifications.

1.1.1 Objects and Classes

Objects are basic runtime entities, in an object oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program may handle. They may also represent user defined data types like lists or arrays.

The objects contain data and code to manipulate that data. The entire set of data and code of any object can be made a user-defined data type using the concept of a class. A class may be thought of as a ‘data type’ and an object as a ‘variable’ of that data type. Once the class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. e.g. Mango, apple and orange are the members of thee class fruit.

1.1.2 Data abstraction and encapsulation

The wrapping up of data and methods into a single unit (Called class) is known as encapsulation. Data encapsulation is the most striking feature of class. The data is not accessible to the outside world and only those methods , which are wrapped in the class, can access it. These methods provide the interface between the object’s data and the program. The encapsulation of the data from direct access by the program is called data hiding. Encapsulation makes it possible for objects to be treated like ‘black boxes’, each performing a specific task without any concern for internal implementation.

Abstraction refers to the act of representing essential features without including the background details or explanations.

1.1.3 Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class. Inheritance supports the feature of hierarchical classification. The concepts of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from an existing one. In Java the main class is known as ‘super class’ and derived class is known as ‘sub class’.

1.1.4 Polymorphism

Polymorphism means ability to take more than one form. E.g. an operation may exhibit different behavior in different instances. The behavior depends on type of data used in the operation. E.g. Consider the operation of addition. For two numbers , it will generate sum. For two strings it will generate third string which is concatenation of current two strings. A single function name can be used to handle different number and different types of arguments.

Polymorphism plays an important role in allowing objects having different internal structures to share thee same external interface. This means that a general class of operation may be accessed in the same manner even though specific actions associated with each operation may differ. Polymorphism plays an important role. Polymorphism is extensively used in implementing inheritance.

1.1.5 Dynamic Binding

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

1.1.6 Message Communication

An object oriented program consists of a set of objects that communicate with each other. Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another. A

message for an object is a request for execution of a procedure and will invoke a method (procedure) in the receiving object that generated the desired result.

Message passing involves specifying the name of the object, the name of the method(message) and the information to be sent. E.g.

```
Employee.salary(name);
```

Here employee is the object, salary is the message and name is the parameter that contains information.

CHAPTER 2 - JAVA FEATURES

Java is a general purpose, object-oriented programming language developed by sun microssystems of usa in 1991.

2.1 JAVA FEATURES

The inventors of Java wanted to design a language which could offer solutions to some of the problems encountered in modern programming.

2.1.1 Compiled And Interpreted

USUALLY A COMPUTER LANGUAGE EITHER COMPILED OR INTERPRETED. Java combines both these approaches thus making Java a two-stage system. First, Java compiler translates source code into what is known as bytecode instructions. Bytecodes are not machine instructions and therefore, in the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program.

2.1.2 Platform-Independent and Portable

The most significant contribution of Java over other languages is its portability. Java programs can be easily moved from one computer system to another, anywhere and anytime. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs. This is the reason why Java has become a popular language for programming for internet which interconnects different kinds of systems worldwide. We can download a Java applet from a remote computer onto our local system via internet and execute it locally. This makes the internet an extension of the user's basic system providing practically unlimited number of accessible applets and applications.

Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the size of the primitive data types are machine independent.

2.1.3 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, that we can use in our programs by inheritance. The object model in Java is simple and easy to extend.

2.1.4 Robust and Secure

Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. It is designed as a garbage collected language relieving the programmers virtually all memory management problems.

Security becomes an important issue for a language that is used for programming on internet.

2.1.5 Distributed

Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java applications can open and access remote objects on internet as easily as they can do in a local system.

2.1.6 Familiar, Simple and Small

Java does not use pointers, preprocessor header files, goto statement and many others. It also eliminates operator overloading and multiple inheritance. It is modeled on c and c++ languages, so looks familiar.

2.1.7 Multithreaded

Multithreaded means 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. This feature greatly improves the interactive performance of graphical applications. The Java runtime comes with tools that support multiprocess synchronization and construct smoothly running interactive systems.

2.1.8 High Performance

Java performance is impressive for an interpreted language, mainly due to the use of intermediate bytecode. Java architecture is also designed to reduce overheads during runtime. Further, the incorporation of multithreading enhances the overall execution speed of Java programs.

2.1.9 Dynamic and Extensible

Java is a dynamic language. Java is 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.

Java programs support functions weitten in other languages such as c and c++. These functions are known as native methods. This facility enables the programmers to use the efficient functions available in these languages. Native methods are linked dynamically at runtime.

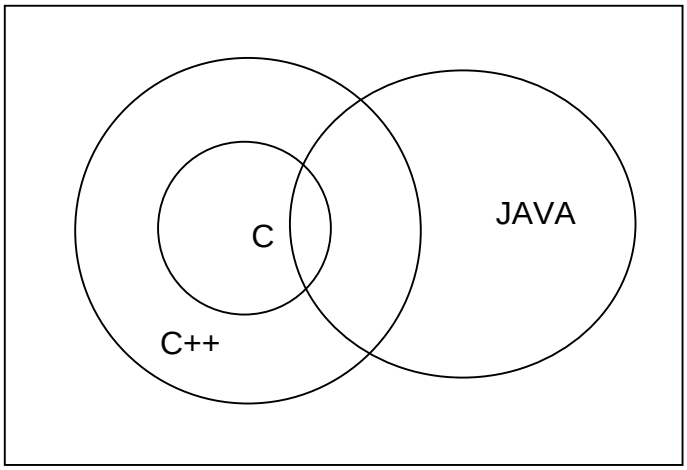
2.1.10 Java and C Differences

- Java does not include the C unique statement keywords goto, sizeof and typedef.
- Java does not contain the data types struct, union and enum.
- Java does not define the type modifiers keywords auto, extern, register, signed and unsigned.
- Java does not support an explicit pointer type.
- Java does not have a preprocessor and therefore we cannot use #define, #include and #ifdef
- Java does not support any mechanism for defining variable arguments to functions.
- Java requires that the functions with no arguments must be declared with empty parenthesis and not with the void keyword as done is C.
- Java adds new operators such as instanceof and >>>.

- Java adds labeled break and continue statements.
- Java adds many features required for object oriented programming.

2.1.11 Java and C++ Differences

- Java does not support operator overloading
- Java does not have template classes as in C++
- Java does not support multiple inheritance in classes. This is accomplished using a new feature called “Interface”
- Java does not support global variables. Every variable and method is declared within a class and forms part of that class.
- Java does not use pointer.
- Java has replaced the destructor function with a finalize() method.
- There are no header files in Java



Overlapping of C, C++ and JAVA

2.2 JAVA ENVIRONMENT

Java environment includes a large number of development tools and hundred of classes and methods. The development tools are part of the system knows and Java Development Kit (JDK) and the classes and methods are part of the Java Standard Library (JSL), also known as the Application Programming Interface (API).

2.2.1 Java Development Kit

The Java Development Kit comes with a collection of tools that are used for developing and running Java programs.

Tool	Description
Appletviewer	Enables us to run Java applets (without using Java compatible browser)
Java	Java interpreter which runs applets and applications by reading and interpreting bytecode files.
Javac	The Java compiler, which translates Java source code to bytecode files that the Interpreter can understand.
Javadoc	Creates html format documentation from Java source code files
Javah	Produces header files for use with native methods
Javap	Java disassembler, which enables us to convert bytecode files into a program description
Jdb	Java debugger, which helps us to find errors in our programs.

The following figure shows the way these tools are applied to build and run application programs.

THE PROCESS OF BUILDING AND RUNNING JAVA APPLICATION

2.3 APPLICATION PROGRAMMING INTERFACE

The Java standard library (or API) includes hundreds of classes and methods grouped into several functional packages. These packages together contain more that 1500 classes and interfaces and define more than 13,000 methods.

Most commonly used packages are :

- Language Support package – A collection of classes and methods required for implementing basic features of Java.
- Utilities Package – A collection of classes to provide utility functions such as date and time functions.

Input/ output package – A collection of classes required for input/output manipulation.

- Networking Package – A collection of classes for communicating with other computer via internet.
- AWT package – The abstract window tool kit package contains classes that implements platform independent graphical user interface.
- Applet Packge – this includes a set of classes that allows us to create Java applets.

CHAPTER 3 - OVERVIEW OF JAVA LANGUAGE

3.1 INTRODUCTION

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

- Standalone applications
- Web applets

Standalone Applications are programs written in Java to carry out certain tasks on a standalone local computer. In fact Java can be used to develop programs for all kinds of applications. Executing a standalone Java program involves two steps.

1. Compiling source code into Bytecode using javac compiler
2. Executing the Bytecode program using Java interpreter

Applets are small Java programs developed for Internet applications. An applet located on a distant computer (server) can be downloaded via Internet and executed on a local computer (client) using a Java-capable browser. We can develop applets for doing everything from simple animated graphics to complex games and utilities. Since applets are embedded in an HTML (Hypertext Markup Language) document and run inside a web page, creating and running applets are more complex than creating application.

Stand-alone Java programs can read and write files and perform certain operations that applets can not do. An applet can only run within a web browser.

3.2 SIMPLE JAVA PROGRAM

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

The best way to learn is a hello world example.

This is the simplest Java program. Nevertheless, it brings out the basic features of the language. So we will discuss the program line by line and understand the unique features that constitute a Java program.

Since Java is a case sensitive language, we need to type in the same case.

3.2.1 Class Declaration

The first line

class SampleOne

Declares a class, which is an object-oriented construct. As stated earlier, Java is true object-oriented language and therefore, everything must be placed inside a class. Class is a keyword and declares that a new class definition follows. SampleOne is a Java Identifier that specifies the name of the class to be defined.

3.2.2 Opening Brace

Every class defination in Java begins with an opening brace "{" and ends with a matching closing brace "}", appearing in the last line in that example.

3.2.3 The main line

The third line

Public static void main(String args[])

Defines a method named main. This is similar to the main() function in C,C++.

Every Java application program must include the main() method. This is starting point for the interpreter to begin the execution of the program. A Java application can have any number of classes but only one of them must include a main method to initiate the execution. (Java applets will not use the main method at all)

This line also includes following keywords.

Public	The keyword public is an access specifier that declares the main method as unprotected and therefore making it accessible to all

	other classes.
Static	Next appears the keyword static, which declares this method as one that belongs to the entire class and not a part of any objects of the class. The main must always be declared as static since the interpreter uses this method before any objects are created.
Void	The type modifier void states that them main method does not return any value (but simply prints some text to the screen.)

All parameters to a method are declared inside a pair of parentheses. Here, String args[] declares a parameter names args, which contains an array of objects of the class type String.

3.2.4 The output Line

The only executable statement in the program is

System.out.println("Hello World !");

Since Java is true object oriented language, every method must be part of an object. The println method is a member of the out object, which is a static data member of System class. The line prints the string

Hello World !

to the screen. The method println always appends a new line character to the end of the string. **Every Java statement must end with a semicolon.**

3.3 MORE OF JAVA

Assume that we would like to compute and print the square root of a number. A Java program to accomplish this is shown on next page.

3.3.1 Program Details

The statement

Double x = 5 ;

Declares a variable x ans intializes it to the value 5 and the statement

Double y ;

```
/*
* More java statements
* A program to compute square root
*/
import java.lang.Math;
class SquareRoot
{
        public static void main(String args[])
        {
                double x = 5;
                double y ;
                y = Math.sqrt(x);
                System.out.println("Y = " +y);
        }
}
```

Merely declares a variable y. The statement

Y = math.sqrt(x);

Invoke the method sqrt of the Math class, computes square root of x and then assigns the result to the variable y. The output statement

System.out.println(“ y = “+y);

Displays the result on the screen as

Y = 2.236067...

Note the use of + symbol. Here it acts as the concatenation operator of two strings. The value of y is converted into a string representation before concatenation.

3.3.2 Use of Math Function

Note that the first statement in the program is import java.lang.Math;

The purpose of this statement is to instruct the interpreter to load the math class from the package lang. Math class contains the sqrt method required in the program.

3.3.3 Comments

Java permits both the single-line comments and multi-line comments. Single line comments begin with // and end at the end of the line as shown on the lines declaring x and y. for longer comments, we can create long multi-line comments by starting with a /* and ending with */.

3.4 AN APPLICATION WITH TWO CLASSES

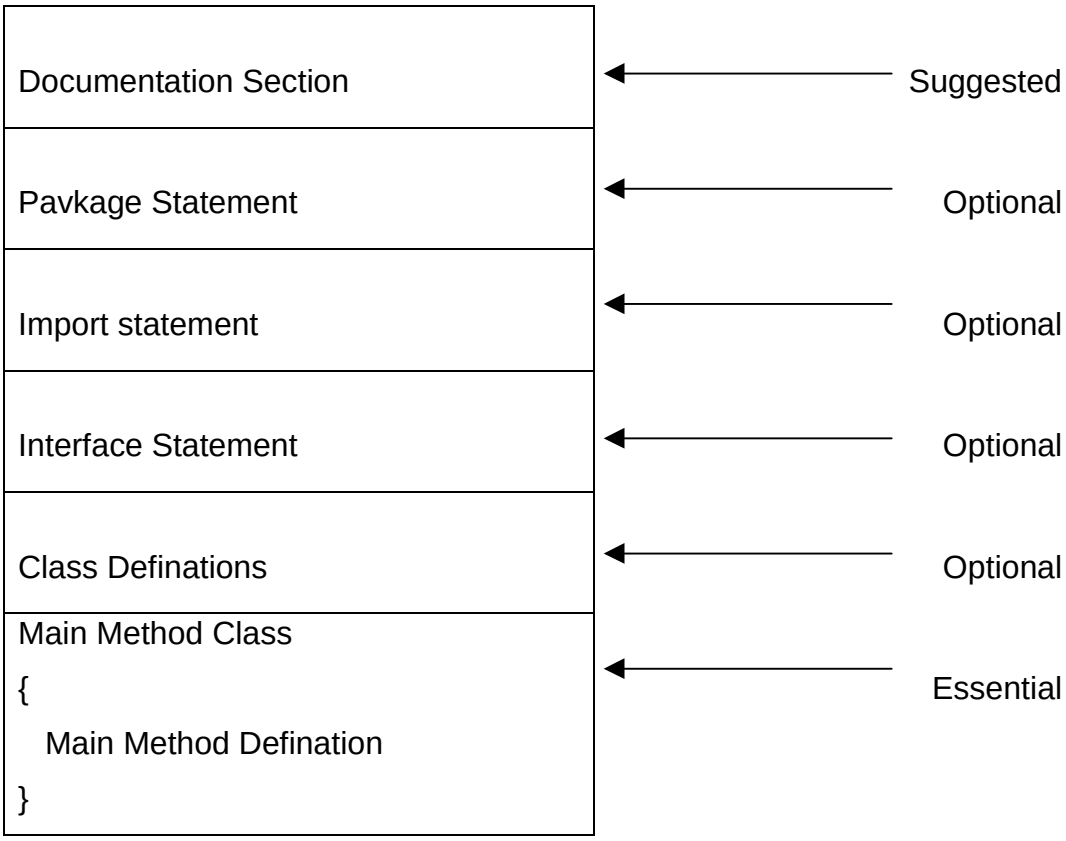
Both the examples discussed above use only one class that contains the main method. A real-life application will generally require multiple classes.

```
class Room
{
        float length;
        float breadth;

        void getdata(float a, float b)
        {
                length  = a;
                breadth= b;
        }
}

class RoomArea
{
        public static void main(String args[])
        {
                float area;
                Room room1 = new Room(); // Creates an object room1
                room1.getdata(14,10);
                area = room1.length * room1.breadth;
                System.out.println("Area = "+area);
        }
}
```

3.5 JAVA PROGRAM STRUCTURE



3.5.1 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. In addition to the two styles of comments discussed earlier, Java also uses a third style of comment /** …*/ known as documentation comment. This form of comment is used for generating documentation automatically.

3.5.2 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.

Package student;

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

3.5.3 Import Statements

The next thing after a package statement may be number of import statements. This statement instructs the interpreter to load the class or classes from given package.

3.5.4 Interface Statements

An interface is like a class but includes a group of method declarations. This section is used only if there is a need to implement the multiple inheritance feature.

3.5.5 Class Definations

A Java program may contain multiple class definations. 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.

3.5.6 Main method class

Every Java standalone program requires a main method as its starting point. 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.

3.5.7 Java tokens

A Java program is basically collection of classes. A class is defined by a set of declaration statements and methods containing executable statements. Most statements contain expressions, which describe the actions carried out on data. Smallest individual units in a program are known as tokens. The compiler recognizes them for building up expressions and statements.

A java program is collections of tokens, comments and white spaces. Java language includes five types of tokens. They are

- Reserved Keywords
- Indentifiers
- Literals
- Operators
- Separators

3.5.8 Java character set

The smallest units of Java language are the characters used to write Java tokens. These characters are defined by the Unicode character set, an emerging standard that tries to create characters for large number of scripts worldwide. The Unicode is a 16-bint character coding system and currently supports more than 34,000 defined characters derived from 24 languages.

3.5.9 Keywords

Keywords are an essential part of a language defination. They implement specific features of the languages. Java has reserved 60 words as keywords. Since keywords have specific meaning, we can not use them as names for variables.
e.g.

abstract, boolean, byte, case, catch, char, class, static, super, switch etc

3.5.10 Identifiers

Identifiers are programmer-designed tokens.

They are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program. Java identifiers follow following rules.

They can have alphabets, digits and the underscore and dollar sign characters.

They must not begin with a digit.

Uppercase and lowercase letters are distinct

The can be of any length

Identifiers must be meaningful, short enough to be quickly and easily types and long enough to be descriptive and easily read. Java developers have followed some naming conventions

Names of all public methods and instance variables start with a leading lowercase letter. E.g. average, sum

When more than one word is used in a name, the second words are marked with leading uppercase letter. E.g. dayTemperature, totalMark

All private and local variables use only lowercase letter combined with underscore. E.g. length, batch_strength

All classes and interfaces start with a leading uppercase letter e.g. Student, HelloJava

Variable that represent constant values use all upperscore letters and underscores between words. E.g. TOTAL, F_MAX

All these are conventions and not rules.

3.5.11 Literals

Literals are a sequence of characters (digits, letters and other characters) that represent constant values to be stored in variables. Java language specifies five major types of literals.

Integer Literals, Floating_point literals, Character literals, String literals, Boolean literals

3.5.12 Operators

An operator is a symbol that takes one or more arguments and operates on them to produce a result.

3.5.13 Separators

Separators are symbols used to indicate where the group of code are divided and arranged. They basically define the shape and function of our code. Eg. Parentheses (), braces {}, brackets [], semicolon; etc

3.5.14 Java statements

The statements in Java are like sentences in natural language. A statement is an execution combination of tokens ending with a semicolon (;) mark. Statements are usually executed in sequence in the order in which they appear.
The next page contains the classification of Java statements. The implementation of these statements is same like C and C++.

3.6 IMPLEMENTING A JAVA PROGRAM

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

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

3.6.1 Creating the Program

The program must be saved with Java extension. This file is called as source file. All Java source files will have extension as “java”. Note also that if a program contains multiple classes, the file name must be the classname of the class containing the main method or the public class.

3.6.2 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 as shown below.

javac Test.java

If every thing is ok, the javac compiler creates a file called Test.class containing the bytecode of the program. Note that the compiler automatically names the bytecode file as <classname>.class.

3.6.3 Running the Program

We need to use the Java interpreter to run a stand-alone program. At the command prompt, type

java Test

Now, the interpreter looks for the main in the program and begins execution from there. When executed, the program displays results.

3.6.4 Machine neutral

The compiler converts the source code files into bytecode files, these codes are machine independent and therefore can be run on any machine. That is a program compiled on IBM machine will run on a Macintosh machine.

Java interpreter reads the bytecode files and translates them into machine code for the specific machine on which the Java program is running. The interpreter is written for each type of machine.

3.7 JAVA VIRTUAL MACHINE

All languages compilers translate source code into machine code for a specific computer. Java compiler also does the same thing. Then how does Java achieve architecture neutrality? The answer is that the Java compiler produces an intermediate code known as bytecode for a machine that does not exist. This machine is called as the Java virtual machine and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer.

The virtual machine code (byte code) is not machine specific. The Java interpreter generates the machine specific code (known as machine code) by acting as intermediary between the virtual machine and the real machine.

3.8 COMMAND LINE ARGUMENTS

There may be occasions when we may like our program to act in a particular way depending on the input provided at the time of execution. This is achieved in Java programs by using what are known as command line arguments. Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution.

Consider the command line

JAVA ComLineTest BASIC FORTRAN C++ JAVA

The command line contains four arguments. These are assigned to the array as follows

BASIC	-	args[0]
FORTRAN	-	args[1]
C++	-	args[2]
JAVA	-	args[3]

Below is a program using command line arguments.

```
/* This program uses command line argumnts */
class ComLineTest
{
 public static void main(String args[ ])
{
    int count,i=0;
    String string;
    count=args.length;
    System.out.println("Number  of  arguments  =  "  +
count);
    while(i<count)
   {
       string=args[i];
       i=i+1;
       System.out.println(i  +  "  :  "  +  "  Java  is  "
+string+"!");
    }
}
}
```

CHAPTER 4 - CONSTANTS,VARIABLES AND DATA TYPES

Like any other language, Java has its own vocabulary and grammar. In this chapter, we will discuss the concepts of constants and variables and their types as they relate to Java language.

4.1 CONSTANTS

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

4.1.1 Integer Constants

123 -321 0 678932

4.1.2 Real Constants

0.00034 -0.55 435.222

4.1.3 Single Character Constants

'5' 'x' ';' ' '

4.1.4 String Constants

"Hello Java" "2000" "5+4"

4.1.5 Backslash Character Constants

Java supports some special backslash character constants that are used in output methods. A list of such backslash character constants is as follows.

Constants	Meaning
'\b'	back space
'\f'	form feed
'\n'	New line

'\r'	carriage return
'\t'	horizontal tab
'\''	single quote
'\"'	double quote
'\\'	backslash

4.2 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.

The programmer can choose a variable name in a meaningful way so as to reflect what it represents in the program. Eg. Average, height, total_height etc

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

- They must not begin with a digit.
- Uppercase and lowercase are distinct. This means Total is not same as total or TOTAL.
- It should not be a keyword.
- White space is not allowed.
- Variables names can be of any length.

4.3 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. Following table shows main data types with categories.

Data Types In Java
- Premitive (Intrinsic)
  - Numeric
    - Integer
    - Floating Point
  - Non Numeric
    - Character
    - Boolean
- Non-Primitive (Derived)
  - Classes
  - Interfaces
  - Arrays

4.3.1 Integer Types

Integer types can hold whole numbers such as 123, -98 and 4567. They further divided into subtypes as follows:

Type	Size	Range
Byte	one byte	-128 to 127
Short	two bytes	-32,768 to 32,767
Int	four bytes	-2,147,483,648 to 2,147,483,647
Long	Eight bytes	-9,223,372,036,854,808 to 9,223,372,036,854,807

4.3.2 Floating Point Types

Floating point type can hold number with decimal points such as 21.34 –1.234.

Type	Size	Range
Float	four bytes	3.4e-038 to 3.4e+038
Double	eight bytes	1.7e+308

4.3.3 Character Type

In order to store character constants in memory, Java provides a character data type called char. The char type size assumes size 2 bytes, but it holds only single character.

4.3.4 Boolean Type

Boolean type is used when we want to test a particular condition during the execution of the program. There are only two values that a boolean type can take – true and false. Both these are keywords. It uses only one bit of storage.

4.4 DECLARATION OF VARIABLES

Variables are the names of storage locations. After designing suitable variable names, we must declare them to the compiler. Declaration does three things

- It tells the compiler what the variable name is.
- It specifies what type of data the variable will hold.
- The place of declaration decides the scope of the variable.

A variable must be declared before it is used in the program.

A variable can be used to store a value of any data type. The general form of declaration of a variable is

type variable, variable, ….,variableN;

e.g.

```
int count;
float x, y;
```

4.5 GIVING VALUES TO VARAIBLES

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

- By using an assignment statement
- By using a read statement

4.5.1 Assignment statement

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

variablename = value;

e.g.

initialvalue = 0;

yes = ‘x’;

We can also string assignment expressions as shown below

X = y = z =0;

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

Type variablename = value;

e.g. int finalvalue = 100;

char yes = ‘x’;

The process of giving initial values to variables is known as initialization. The ones that are not initialized are automatically set to zero.

4.5.2 Read Statement

We can give values to variables through keyboard using the readLine() method as follows

The readLine() method (which is invoked using an object of class DataInputStream) reads the input from the keyboard as a string which is then converted in to corresponding data type using the data wrapper classes.

Note that we have used the keywords try and catch to handle any errors that might occur during the reading process.

```
import  java.io.DataInputStream;
class Reading
{
   public static void main(String args[ ])
        {
        DataInputStream 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);
        }
}
```

4.6 SCOPE OF VARIABLES

Java variables are actually classified into three kinds

- Instance variables
- Class variables
- Local variables

Instance and Class variables are declared inside a 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 a class and belong to the entire set of objects that class creates. Only one memory location is created for each class variable.

Variables declared and used inside methods are called local variable. They are called so because they are not available for use outside the method defination. Local variables can also be declared inside 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(usable) is called its scope. We can have program blocks within other program blocks which is called nesting. We cannot declare a variable to have the same name as one in an outer block. (which was valid in C, C++)

4.7 SYMBOLIC CONSTANTS

We often use certain unique constants in a program. These constants may appear repeatedly in a number of places in the program. E.g. pi or max number etc. We face two problems in the subsequent use of such programs. They are

- Problem in modification of the program.
- Problem in understanding the program

A constant is defined as follows:

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

Example:

```
final  float  PI=3.14;
final   int    SIZE=20;
```

In above declaration final is keyword.

Note that:

- Symbolic names take the same form as variables names. But they are written in CAPITALS to visually distinguish them from normal variable names.
- They can not be declared inside a method. They should be used only as class data members in the beginning of the class.

4.8 TYPE CASTING

We often encounter situations where there is a need to store a value of one type into a variable of another type. In such situations, we must cast value to be stored by proceeding it with parentheses.

The syntax is:

```
type variable1= (type) variable2;
```

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

Example:

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

Casting into smaller type can result in a loss of data. Casting of float into int will result in loss of fractional part. The list of casts, which are guaranteed to no loss of data as follows.

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

4.8.1 Automatic Convertion

For some types, it is possible to assign a value of one type to a variable of different type without a cast. Java does the conversion of assigned value automatically. This is known as automatic type conversion. This is possible only if destination type has enough precision to store the source value. E.g. int is large enough to hold a byte value. Therefore,

```
Byte b = 75;
int a = b;
```

are valid statements.

The process of assigning a smaller type to a larger type is called widening or promotion and reverse is called narrowing.

4.9 PRINTING VALUES OF VARIABLES

A computer program is written to manipulate a given set of data and to display or print the results. Java supports 2 methods that can be used to send results to the screen.

print( ) method //print and wait

println( ) method //print a line and move to next line.

The print() method sends information into buffer. This buffer is not flushed until a newline character is sent.

As a result, print() method prints output on one line until a newline character is encountered.

```
class Displaying
{
  public static void main(String args[])
{
    System.out.println("Screen Display");
    for(int i=0;i<=9;i++)
    {
      for(int j=0;j<=i;j++)
       {
         System.out.print("  ");
         System.out.print(i);
       }
    System.out.print("\n");
    }
  System.out.println("Screen Display Done");
}
}
```

Example:

System.out.print(“Hello”);

System.out.print(“Java”);

Output:

Hello Java

The println( ) method, by contrast displays information followed by line feed.

Example:

System.out.println(“Hello”);

System.out.println(“Java”);

Output:

Hello

Java

4.10 STANDARD DEFAULT VALUES

In Java, 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 as shown in following table.

Type 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

CHAPTER 5 - OPERATORS AND EXPRESSIONS

5.1 INTRODUCTION

Java supports a rich set of operators. We have already used several of them, such as =, +, - and *. An operator is a symbol that tells computer to perform certain mathematical or logical manipulations. They usually form a part of mathematical or logical expressions.

Java operators can be classified into a number of related categories as below:

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.

5.2 ARITHMATIC OPERATORS

Java provides all the basic arithmatic operators. These can operate on any built in numeric data type of Java. We cannot use these operators on Boolean type.

The list of operators is as follows:

Operators	Meaning
+	Addition or unary plus.
-	Subtraction or unary minus.
*	Multiplication.
/	Division.
%	Modulo division.

5.2.1 Integer Arithmetic

When the operands in a single arithmetic expression such as a+b are integers, the expression is called an *integer expression*, and the operation is called *integer arithmetic*. Integer arithmetic always yields an integer value. In the above example, if a and b are integers, then for a=14 and b=4 we the have the following results:

a + b = 18
a – b = 10
a * b = 56
a / b = 3 (decimal part truncated)
a % b = 2 (reminder of integer division)

a/b, when a and b are integer types, gives the result of division of a by b after truncating the divisor. This operation is integer division.

For modulo division, the sign of result is always the sign of first operand (the dividend). That is

-14 % 3 = 2
-14 % -3 = -2
14 % -3 = 2

(Note that modulo division is defined as: a%b = a-(a/b)*b, where a/b is integer division).

5.2.2 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. Since floating point values are rounded to the number of significant digits permissible, the final value is approximation of the correct result.

Modules operator % can be applied to floating point data as well. The floating-point module operator returns the floating-point equivalent of an integer division. What this means is that the division is carried out with both floating point operand, but the resulting divisor is treated as an integer, resulting in a floating point remainder.

```
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 of Program is as follows:

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

5.2.3 Mixed – Mode Arithmetic

When one of the operands is real and the other is integer, the expression is called *mixed-mode arithmetic* expression. If either operand is of real type, then the other operand is converted to real and the real arithmetic is performed. The result will be real. Thus

15 / 10.0 produces the result 1.5

whereas

15 / 10 produces the result 1

More about mixed operations will be discussed when we deal with the evaluation of expressions.

5.3 RELATIONAL OPERATORS

We often compare two quantities, and depending on their relation, take certain decisions. These comparisons can be done with the help of relational operators. An expression such as

a < b or x < 20

Containing a relational operator is termed as a relational expression. The value of relational expression is either true or false. These operators and their meanings are as follows:

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is grater than
>=	is grater than or equal to
==	is equal to
!=	not equal to

A simple relational expression contains only one relational operator is of the following form:

ae-1 relational operator ae-2

ae-1 and ae-2 are arithmetic expression, which may be simple constants, variables or combination of them.

5.3.1 Relational Expression

Expressions	Value
4.5 <= 10	True
4.5 < -10	False
-35 >=0	False
10 < 7 + 5	True

a + b == c + d	True

When certain arithmetic expressions are used on either side of relational operator, the arithmetic expressions will be evaluated first and then results compared. That is, arithmetic operators have higher priority over 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==b));
      System.out.println("  a <= c is  " + (a<=c));
      System.out.println("  a >= b is  " + (a>=b));
      System.out.println("  b != c is  " + (b!=c));
      System.out.println("  b == a+c is  " + (b==a+c));
     }
}
```

The output of 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

b != c is true

b == a + c is false

5.4 LOGICAL OPERATORS:

Java has three logical operators.

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

Like the simple relational expressions, a logical expression also yields a value of true or false, according to the 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

Note:

- op-1 && op-2 is true if both op-1 and op-2 are true and false otherwise.
- op-1 || op-2 is false if both op-1 and op-2 are false and true otherwise.

5.5 ASSIGNMENT OPERATORS

Assignment operators are used to assign the value of an expression to a variable. We have seen the usual assignment operator, '='. In addition, Java has a set of 'shorthand' assignment operators, which are used in 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 shorthand assignment operator .

The assignment statement

v op = exp;

is equivalent to

v = v op(exp);

with v accessed only once. Consider an example

x += y+1;

This is same as the statement

x = x+(y+1);

Shorthand Assignment Operators

Statement with simple assignment operator	Statement with shorthand assignment operator
a = a + 1	a + = 1
a = a – 1	a -= 1
a = a * (n+1)	a * = n+1
a = a / (n+1)	a / = n+1
a = a % b	a % = b

The use of shorthand assignment operators has three advantages

- What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
- The statement is more concise and easier to read.
- Use of shorthand operator results in a more efficient code.

5.6 INCEREMENT AND DECREMENT OPERATORS

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 use in following form:

++m; or m++;

--m; or m--;

++m is equal to m = m + 1; (or m += 1)

--m is equal to m = m- 1; (or m -=1)

While ++m and m++ mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement. Consider 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 value of y would 5 and m would be 6.

```
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("  n = " + n);
    }
}
```

Output of Program:

m = 10

n = 20

++m = 11

n++ = 20

m = 11

n = 21

5.7 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,exp3 are expression.

e.g.

```
a = 10;
b = 20;
x = (a > b ) ? a : b;
```

This is similar to say,

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

5.8 BITWISE OPERATORS

Java supports bitwise operaors for manipulation of data at values of bit level. These operators are used for testing the bits, or shifting them to the right or left. Bitwise operators may not be applied to float or double.

Operator	Meaning
&	Bitwise AND
!	Bitwise OR
^	Bitwise exclusive OR
~	One’s compliment
<<	Shift left
>>	Shift right
>>>	Shift right with zero fill

5.9 SPECIAL OPERATORS

Java supports some special operators like instanceof operator and member selection operator(.).

5.9.1 instanceof Operator

The instanceof is an object reference operator and returns true if the object on the left-hand side is an instance of the class given on the right-hand side.

person instanceof student

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

5.9.2 Dot operator

The dot operator (.) is used to access the instace variables and methods of class objects.

e.g. person1.age, person1.salary()

5.10 PRECEDENCE OF ARITHMETIC OPERATORS

An arithmetic expression without any parentheses 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	+ -

CHAPTER 6 - DECISION MAKING AND BRANCHING

A Java program is a set of statements, which are normally executed sequentially in the order in which they appear. This happens when options or repetitions of certain calculations are not necessary. However, in practice we have number of situations, where we may have to change the order of execution of statement based on certain condition. This involves decision-making. When a program breaks the sequential flow and jumps to another part of program, it is called branching.

Java language possesses such decision making capabilities and supports the following statements called as control or decision making statements.

1. if statement
2. switch statement
3. Conditional operator statement

6.1 DECISION MAKING WITH IF STATEMENT

The if statement is a powerful decision making statement and is used to control the flow of execution of statements. It is a two-way decision statement. It takes following form :

if (test expression)

It allows computer to evaluate test expression first and depending on the value of the expression is true or false, it transfers control to particular statement.

The if statement may be implemented into various form as follows:

1. Simple if statement
2. if…else statement
3. nested if…else statement
4. else if ladder

6.2 SIMPLE IF STATEMENT

The general form of a simple statement is

```
If (test expression)
{
        statement-block;
}
statement-x;
```

The statement-block may be single statement or a multiple statements. If test expression is true, the statement-block will be executed, otherwise statement – block will be skipped and execution will jump to statement-x.

Consider the following example:

```
If (category==sports)
{
marks = marks + bonus_marks;
}
System.out.println(marks);
```

In the above case expression category==sports becomes true then if block will be executed and bonus_marks will be added into marks otherwise statement after if block that is println statement will be executed.

6.3 THE IF….ELSE STATEMENT

The if….else statement is an extension of simple if statement . The general form

```
if ( test exprssion)
{
    True-block statement(s)
}
else
{
     False-block statement(s)
}
statement-x
```

If the test expression is true, then the true-block statement(s) are executed; otherwise, the false-block statement(s) will be executed. In either case true or false block will be executed not both. In both cases, the control is transferred subsequently to the statement-x.

```
class IfElseTest
{
   public static void main(String args[])
    {
       int number=50;
       if(number % 2==0)
           {
                  System.out.println("Even Number");
           }
         else
           {
                  System.out.println("Odd Number");
           }
    }
}
```

6.4 NESTING OF IF….ELSE STATEMENTS

When a series of decisions are involved, we may have to use more than one if…else statement in nested form as follows.

```
if ( test condition1 )
{
    if ( test condition2 )
     {
        statement-1;
     }
    else
    {
```

```
        statement-2;
    }
}
else
{
         statement-3;
}
statement-x;
```

The logic of execution is that: If the condition1 is false , the statement3 will be executed; otherwise it continues to perform the second test. If condition2 is true , the statement-1 will be executed; otherwise the statemnt-2 will be evaluated and control transfer to statement-x.

```
class IfElseNesting
{
   public static void main(String args[])
    {
       int a=50 , b=100 ,c=75;
       if( a > b)   {
          if(a>c)   {
               System.out.println(a);
           }
           else  {
                System.out.println(c);
           }
        }
       else    {
            if( c > b )  {
                System.out.println(c);
             }
            else {
                System.out.println(b);
            }
       }
    }
}
```

6.5 THE ELSE IF LADDER

There is another way of putting ifs together when multipath decisions are involved. It takes the following form:

```
if ( condition 1 )
        statement-1;
else if ( condition 2 )
        statement-2;
else if ( condition 3 )
        statement-3;
else if ( condition n )
        statement-n;
statement-x;
```

This construct is known as else if ladder. The condition are evaluated from the top, downwards. As soon as the true condition is found, the statement associated with it is executed and control is transferred to statement-x.

6.6 THE SWITCH STATEMENT

We have seen that when one many alternatives is to be selected, we can design program using if structure to control selection. However, complexity of such program increases dramatically as number of alternative increases.

Java has a built-in multiway decision statement known as switch. The switch tests value of expression against the list of case values and when a match is found, block statement associated with that case is executed. The general form of the switch statement is as follows:

```
switch(expression)
    {
      case value-1  :
                block-1
                break;
```

```
        case value-2  :
                block-2
                break;
        case value-n  :
                block-n
                break;
        default :
                default-block
                break;
    }
statement-x;
```

The expression is an integer expression or character. Value-1,value-2….. are constants and known as case labels. Each of these should unique within a switch statement. Block-1, block-2….. are statement lists and may contain zero or more statements. Each case label ends with colon (:).

When the switch is executed, the value of expression, is compared against the values value1,vallue2…. If case is found whose value matches with the value of expression, then block of statement that follows case are executed.

The break statement at the end of each block signals the end of particular case and causes exit from switch statement to statement-x.

The default is an optional case. When present, it will be executed if value of expression doesn't match any case values. If not present no action takes place when all matches fail and control goes to statement-x.