CONTENTS

ADVANCED JAVA PROGRAMMING

SYLLABUS

UNIT I

Features of Java: History – Characteristics of Java – Developing and Running a java program – Structure of a java program – Variables – Features of java-datatype-type conversion and casting – Arrays-operators – Bitwise operators – Leftshit-right shift – unsigned right shift operators – Relation-Boolean logic – Ternary operators.

UNIT II

Branching and Looping Statements: If, If – else, nested-if-else, if else it statement – switch case – while loop – Do while – for loop – Break, continue and return statements. Classes methods and objects: examples-declaring objects – methods in classes – constructors – this keyword – class structure.

UNIT III

Extension to classes and methods: Methods overloading passing objects to methods – passing arguments – returning objects – recursion-nested classes – string handling – command line execution. Inheritance: basic concepts – multilevel hierarchy – method overriding abstract classes, packages and interfaces.

UNIT IV

Errors and exception Handling: Compile time, runtime errors exceptions – try and catch, multiple catch-throw – java's built-in-exceptions. Multiple thread programming: java threads - creating several threads-deadlock – controls on treads.

UNIT V

Input-Output operations reading characters, sentences writing to console, file processing, copying files, Applets: Various appelets: Chkr, cs, de, font, ga, lbg, rc, rcc, sp, common.html file. Graphics and Text: lines, rectangle, ellipse, arcs, polygons, paintmode, fonts, text.

UNIT I

LESSON

1

ADVANCED JAVA PROGRAMMING - AN INTRODUCTION

CONTENTS

1.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain the features and characteristics of Java
- Discuss the history of Java
- Develop and run a Java program
- Describe the structure of Java program

1.1 INTRODUCTION

By the end of 1980s and the early 1990s, Object-oriented programming using C++ took hold. For a brief moment, it seemed as if programmers had found the perfect language finally. C++ was a language that could be used to create a wide range of programs. However, forces were brewing that

would drive computer language evolution forward once again. Within few years WWW (World Wide Web) and the Internet would reach critical mass. This event would cause another revolution in programming.

1.2 FEATURES OF JAVA

The main features of Java programming language are as follows:

1. ***Simple Small and Familiar:*** Java was designed to be easy for the professional programmer to learn and use effectively. If you already understand the basic concepts of object-oriented programming, learning Java will be even easier. If you are an experienced C++ programmer, moving to Java will require a very little effort. Because Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble learning Java. Also, some of the more confusing concepts of C++ are either left out of Java or implemented in a cleaner, more approachable manner. In Java, there are a small number of clearly defined ways to accomplish a given task.

2. ***Compiled and Interpreted:*** The key that allows Java to solve both the security and the portability problems is that the output of a Java **compiler** is not executable code. Rater, it is bytecode. **Bytecode** is highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an **Interpreter** for bytecode. This may come as a bit of a surprise.

   If a Java program were compiled to native code, then different versions of the same program would have to exit for each type of CPU connected to the Internet. This is, of course, not a feasible solution. Thus, the interpretation of byte code is easier way to create truly portable programs.

3. ***Portable:*** Many types of computers and operating systems are in use throughout the world and many are connected to the Internet. For programs to be dynamically downloaded to all the various types of platforms connected to the Internet, some means of generating portable executable code is needed. Java's solution to this problem is both elegant and efficient.

4. ***Platform-independent:*** Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java byte code. This code can be interpreted on any system that provides a Java Virtual Machine (JVM).

   While it is true that Java was engineered for Interpretation, the Java byte code was carefully designed so that it would be easy to translate directly into native machine code for high performance by using a just-in-time compiler. Java run-time systems that provide this feature lose none of the benefits of the platform-independent code.

5. ***Object-oriented:*** Although influenced by its predecessors, Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank slate. One outcome of this was a clean, usable, pragmatic approach to objects. Borrowing liberally from many seminal object-software environments of the last few decades, Java manages to strike a balance between the purist's "everything is an object" paradigm and the pragmatist's "stay out of my way" model. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance non objects.

6. ***Robust:*** The multi-platformed environment of the web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. Thus, the ability to create robust programs was given a high priority in the design of Java.

7. ***Secure:*** As you are likely aware, every time that you download a "normal" program, you are risking a viral infection. Prior to Java, most users did not download executable programs frequently, and those who did scan them for viruses prior to execution. Even so, most users still worried about the possibility of infection with virus. In addition to viruses, another type of malicious program exists that must be guarded against. This type of program can gather private information. Java answers both of these concerns by providing a "Fire wall" between a networked application and your computer.

   When you use a Java-compatible web browser, you can safely download Java applets without fear of viral infection or malicious matter. No security will be breached is considered by many to be the single most important aspect of Java.

8. ***Distributed:*** Java is designed fro the distributed environment of the Internet, because it handles TCP/IP protocols. The original version of Java (Oak) included features for intra-address-space messaging. This allowed objects on two different computers to execute procedures remotely. Java has recently revived these interfaces in a package called ***Remote Method Invocation (RMI):*** This method brings an unparalleled level of abstraction to client/server programming.

9. ***Multithreaded and Interactive:*** Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously. The Java run-time system comes with an elegant yet sophisticated solution for multi-process synchronization that enables you to construct smoothly running interactive systems. Java's easy-to-use approach to multithreading allows you to think about the specific behavior of your program, not the multitasking subsystem.

1.3 HISTORY OF JAVA

Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Fronk and Mike Sherfidan it Sun Microsystems. Inc., in 1991. It took 18 months to develop the first working version. This language was initially called 'Oak' but was renamed "Java" in 1995. Between the initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995, many more people contributed to the design and evolution of the language.

Surprisingly, the original impetus for Java was not Internet! Instead, the primary motivation was the need for a platform independent (architecture neutral) language that could be used to create embedded softwares in a cost-efficient way, gosling and others began work on a portable, platform-independent language. The effort ultimately led to the creation of Java. While the desire for an architecture-neutral language provided the initial spark, the Internet ultimately led to Java's large-scale success.

1.4 CHARACTERISTICS OF JAVA

How Java differs from C and C++?

As mentioned earlier, Java derives much of its character from C and C++. This is by Internet. The Java designers knew that using the familiar syntax of C and echoing the object-oriented features of C++ would make their language appealing to the legion of experienced C/C++ programmers. In addition to the surface similarities, Java shares some of the other attributes that helped make C and C++ successful.

Because of the similarities between Java and C++, it is tempting to think of Java as simply the "Internet Version of C++". However, to do so would be a large mistake. Java has significant practical and philosophical differences. While it is true that Java was influenced by C++, it is not an enhanced version of C++. For example, Java is neither upwardly nor downwardly compatible with C++. Of course, the similarities with C++ are significant, and if you are a C++ programmer, then you will feel right at home with Java. One other point: Java was not designed to replace C++. Java was designed to solve a certain set of problems. C++ was designed to solve a different set of problems. Both will coexist for many years to come.

However, Java embodies changed in the way that people approach the writing of programs, specifically, Java enhances and refines the paradigm used by C and C++. Also, Java is to Internet programming what C was to systems programming: A revolutionary force that will change the world.

Java and Internet

Java has become inseparably linked with the outline environment of the Internet. Somewhat surprisingly, the original impetus for Java was not the Internet! Instead the primary motivation was the need for a platform-Independent language that could be used to create software to be embedded in various consumer electronic devices. However, with the advent of the Internet and the Web, the old problem of portability returned with a vengeance. After all, the Internet consists of a diverse, distributed universe populated with many types of computers, operating systems and CPUs. Even though many types of platforms are attached to the Internet, users would like them all to be able to run the same program. What was once an irritating but low-priority problem had become a high-profile necessity.

In fact, Java was initially designed to solve on a small scale could also be applied to the Internet on a large scale. This realization caused the focus of Java to switch from consumer electronics to Internet programming. So, while the desire for an architectural-neutral programming language provided the initial spark, the Internet ultimately led to Java's large-scale success.

Java and World Wide Web

About the time that the details of Java were being worked out, a second, and ultimately, more important, factor was emerging that would play a crucial role in the future of Java. This second force was, of course, the World Wide Web. Had the web not taken shape at about the same time Java was being implemented, with the emergence of the World Wide Web? Java was propelled to the forefront of computer language design, because the web, too, demanded portable programs.

Web Browsers

To access the World Wide Web, you use what is called a Web browser (already discussed in the previous chapter). Browsers are sometimes also called Web clients, since they get information from a server. When you start a WWW browser or follow a hyperlink, the browser (acting like a client) sends a request to a site on the Internet. That site (acting like a server) returns a file which the browser then has to display. In order for you to see or hear what's in the file, the browser should be capable of interrupting its contents. This differs depending on the type of file, text, graphics and/or images that may be displayed. If the file is written using HyperText Markup Language (HTML), the browser interprets the file so that graphics and images are displayed along with the text.

Basic Features of Browsers

Before we get involved in all the details, let is discuss some important browser features.

- The Web browser should be able to look at the Web pages throughout the Internet or to connect to various sites to access information, explore resources, and have fun.
- The Web browser must enable you to follow the hyperlinks on a Web page and also to type in a URL for it to follow.
- Another feature of browser is to have a number of other commands readily available through menus, icons, and buttons.
- Your browser ought to include an easy way to get on-line help as well as built-in links to other resources on the Web that can give you help or answers to your questions.
- You will definitely want a way to save links to the sites you have visited on the WWW so that you can get back to them during other sessions. Web browsers take care of those in two ways, through a history list, which keeps a record of some of the Web pages you've come across in the current session, and a bookmark list, which you use to keep a list of WWW pages you want to access any time you use your browser. The name of the site and its URL are kept in these lists. The bookmark list is particularly important and the browser will contain tools to manage and arrange it.
- One of the main features of a browser is to search the information on the current page as well as search the WWW itself.
- Browsers give you the facility to save a Web page in a file on your computer, print a Web page on your computer, and send the contents of a Web page by e-mail to others on the Internet.
- Few Web browsers (like Netscape Communicator) are complete Internet package, means they come with components like e-mail client, newsgroup client, an HTML composer, telnet client, ftp client, etc.
- Web browser should be able to handle text, images of the World Wide Web, as well as the hyperlinks to digital video, or other types of information.
- To take advantage of some of the most exciting things on the World Wide Web, your browser needs to properly display and handle Web pages that contain animated or interactive items. Netscape Navigator can incorporate these features through its ability to interpret programs written in Java and Java Script.
- Web browsers interact not just with the Web, but also with your computer's operating system and with other programs, called plug-ins, that gives the browser enhanced features.
- Another important feature to insist on in your browser is caching. A browser that caches keeps copies of the pages you visit so that it does not have to download them again if you want to return to them. Reloading a page from the cache is much quicker than downloading it again from the original source.
- The most important feature of any browser is ease of use. While all Web browsers are fundamentally simple to use, the one you settle on should be very easy to work with; it should function as a transparent window onto the Web.

- If you will be browsing the Web from within a secured network, you may have to configure your browser to work through a special computer on your network called a proxy server. Most popular browsers let you configure them to work with a proxy server, but some don't, so find out if you will be working through a proxy before deciding on your browser. If you are, your ISP or system administrator will tell you if you need to do anything special to use your browser.

Java Support Systems

Java Support Systems includes

- Applets
- Servlets
- Java Beans
- EJB
- JSP
- XML
- SOAP
- CORBA

1.5 IMPLEMENTING A JAVA PROGRAM

The steps involved in the implementation of a Java program are as follows:

- Developing a program
- Compiling the program
- Running the program.

Remember to install the Java Development Kit (JDK) properly before creating the program in Java.

1.5.1 Developing the Java Program

A program can be created using any text editor. Let us consider an example program

```
// A Simple test program
Class First
{
        public static void main (String agrs[ ])
        System.out.println("Welcome to Java World");
        System.out.println("Hello!");
        System.out.println("Let us begin learning Java");
}
```

The above program must be saved with the file name "First;Java". This file is called the source file. Note that the file name must be the class name of the class containing the main method along with the file extension "java".

1.5.2 Compiling the Java Program

To compile the program, type the following instruction at the command prompt:

```
Javac First.java
```

If the program is compiled successfully, the javac compiler creates a file called First.class containing the bytecode of the program.

1.5.3 Running the Java Program

Java interpreter is used to run a stand-alone program. To run the program, type the following instruction at the command prompt:

```
Java First
```

When executed, our program displays the following:

```
Welcome to Java world.
Hello!
Let us begin learning Java.
```

1.5.4 Machine Neutral

The compiler converts the source code files into byte code files. These byte codes are machine independent and therefore can be executed on any machine, either an IBM machine or a Machintosh machine or any other type.

Java interpreter translates the bytecode into binary code for the specific machine on which the Java program is running. Therefore, the interpreter is specially written for each type of machine.

1.5.5 A simple Java Program

Let us introduce a simple example program to understand the basic structure of a Java program.

```
// A Simple Java Program
Class Simpleone
{
public static void main (String args[ ])
      {
      System.out.println ("A simple Java Program");
      }
}
```

We will explain the above program step by step.

Comment

The first line is a single line comment which will be ignored during compilation. Comments make you understand what your program does. Use /*-----*/ for multiple line comments.

Class Declaration

In Java, everything must be declared in a class, which is declared by the keyword Class. Public is a keyword which makes this class available and accessible to any one. (Simple) one is a Java identifier that specifies the name of the class to be defined.

Opening Brace

In java, every class definition begins with an opening brace "{" and ends with a closing brace "}". Pair of "{ }" braces define the scope of any method or class.

The Main Line

The next line of code is shown here:

```
Public static void main (String args[ ])
```

This line begins the main( ) method. This is the line at which the program will begin executing. All Java applications begin execution by calling main( ). The main( ) must be declared as public, since it must be called by code outside of its class when the program is started. The keyword void simply tells the compiler that main ( ) does not return a value. The keyword static allows main( ) to be called without having to instantiate a particular instance of the class.

In main( ), there is only one parameter, String args[ ] which declares a parameter named args, which is an array of instances of the class String.

The Output Line

The next line of code is shown here:

```
System.out.println ("A Simple Java Program.");
```

This line outputs the string "A Simple Java Program", followed by a new line on the screen. Output is accomplished by the built-in println( ) method, System is a predefined class that provides access to the system, and out is the output stream that is connected to the console. All statements in Java end with a semicolon (;).

1.6 JAVA PROGRAM STRUCTURE

As illustrated in the previous examples, a Java program may contain many classes of which main( ) is defined in only one class. Classes contain data members and methods that operate on the data members of the class. Methods may contain datatype declarations and executable statements. To write a Java program, we first define classes and then put them together. The different sections of a Java program are shown in the Figure 1.1.

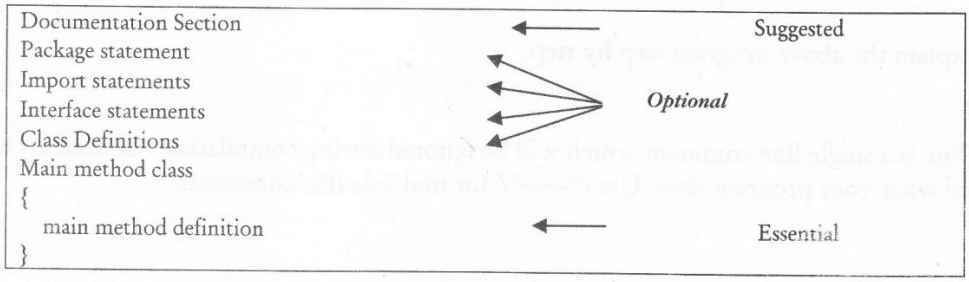


Figure 1.1: Java Program Structure

Document Section

The document section comprises a set of comment lines giving the name of the program, the author and the 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.

Package Statement

Package statement is the first statement allowed in java. This statement declares a package name and informs the compiler that the classes defined here belong to this package. Example:

```
package course;
```

The package statement is optional i.e., our classes may or may not be a part of package.

Import Statements

After a package statement and before any class definition, may appear a number of import statements.

Example:

```
import course.selection;
```

This statement instructs the interpreter to load the selection class contained in the package course. We can access classes that are part of other named packages using import statements.

Interface Statements

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 multiple inheritance features in the program.

Class Definitions

A Java program may contain single or multiple class definitions. They 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 program.

Main Method Class

This class is the essential part of a Java program since every Java stand-alone program requires a main method as its starting point. The main method creates objects of various classes and establishes communication between them. The program terminates on reaching the end of main, and the control passes back to the operating system.

Check Your Progress

Fill in the blanks:

1. ..... is highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM).
2. To access the World Wide Web, you use what is called a .....
3. ..... method brings an unparalleled level of abstraction to client/server programming.

1.7 LET US SUM UP

Java is a general-purpose, object-oriented language. In this lesson we have described the basic structure of a Java program. The file name should be same as the class which contains the main( ) method, alongwith the extension 'java'. We have also discussed the basic elements of the Java language and steps involved in creating and executing a Java application program. Various types of Java statements are also discussed. Implementation of Java program was discussed. Then at last, JVM, its usage and how it makes Java secure and platform independent were discussed.

1.8 KEYWORDS

Class: A collection of variables and methods that an object can have, or a template for building objects.

Argument: A value that is sent to a method when the method is called.

Identifier: A symbol that represents a program object.

Package: A Java keyword used to assign the contents of a file to a package.

Virtual Machine: An abstract, logical model of a computer used to execute Java bytecodes.

Method: A routine that belongs to a class.

Bytecode: The machine-independent output of the Java compiler and input to the Java interpreter.

1.9 QUESTIONS FOR DISCUSSION

1. Describe the history of Java.
2. Describe various features of Java.
3. How is Java differ from C++?
4. Describe the basic structure of a Java program.
5. What is the function of main method in Java?
6. "Java programs are machine neutral". Comment on this statement.

Check Your Progress: Model Answers

1. Bytecode
2. Web browser
3. Remote Method Invocation (RMI)

1.10 SUGGESTED READINGS

E. Balaguruswami, *Programming with Java*, Tata McGraw Hill.

Herbert Schilelt, *The Complete Reference Java 2*, Tata McGraw Hill.

LESSON

2

VARIABLES, ARRAYS AND DATA TYPES

CONTENTS

2.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain variables in java
- Describe datatypes, type conversions and casting
- Discuss arrays

2.1 INTRODUCTION

When working with computers, either for something as simple as writing a college paper or as complex as solving quantum theory equations, the single most important thing for the computer to do is deal with data. Data to a computer can be numbers, characters or simply values. Like any other programming language, Java supports its own set of data types. In this lesson we will discuss various data types available in Java, their scope, arrays and variables and other related features.

2.2 VARIABLES

Variables are the identifiers that are used to store a data value. A variable may take different values at different times during the execution of the program.

A rule for writing variable names in Java is given below:

"A variable name is an arbitrary long sequence of letters and digits. The first character must be a letter; the underscore '_' counts as a letter. Upper-case and lower-case letters are different. All characters are significant."

Java is case sensitive as it treats upper and lower case characters differently. The following are some valid variable names:

Myfile	_DP	A2 To A9
MyFILE	DATE E9_04_79	_HP2_JM
_CHK	FILE_13	.

2.2.1 Declaration of Variables

Identifiers are the names of variables. They must be composed of letters, numbers, the underscore and the dollar sign ($). They cannot contain white spaces, identifiers may only begin with a letter, the underscore or a dollar sign. You cannot begin a variable name with a number. All variable names are case sensitive, for example myvar is not equal to MyVar.

After designing suitable variable names, we must declare them to the compiler. Declaration does three things:

- It tells the compiler the name of the variable.
- It specifies the data type of the variable.
- It tells about the scope of the variable.

Variables can be declared through the Syntax:

```
type variable 1, variable 2, ……., variable N;
```

Variables are separated by commas. Type may be any valid data type.

2.2.2 Giving values to Variables

After declaring a variable, it must be given a value before it is used in an expression. There are two ways to give values of the variables:

1. By using Assignment Statement
2. By using Read Statement.

Assignment Statement

The simplest method of giving value to a variable is through the assignment statement. Variables can be assigned a value through the Syntax:

```
Variable Name = Value;
```

Example

```
initValue  = 0;
FirstValue = 100;
Char c = 'X';
```

We can also string assignment expressions as shown below:

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

Variables can also be assigned a value at the time of their declaration. The Syntax is:

```
type Variable Name = Value;
```

For example, int initvalue = 0;

This process of giving initial values to variables is known as the initialization. The variables that are not initialized are automatically set to 0.

Read Statement

We may also give values to the variables directly through the keyboard using the following methods:

- read () : to read a character from the keyboard.
- readline () : to read a string from the keyboard.

Let's consider a program to illustrate the usage of above two methods:

```
//program to read input from the keyboard
import java.io.*;
Class Reading
{
      public static void main(String args[ ])
      throws IOException
      {
      Char c;
      BufferReader br = new BufferReader(new) Input StreamReader(System.in));
      //to input characters
      System.out.println("Enter characters, 'q' to quit".);
      do
            {
```

```
            c = (char) br.read( );
            system.out.println (c) ;
            } while (c!='q');
      }
}
```

Here is a sample run:

```
      Enter characters, 'q' to quit.
      1 2 3 a b c q
      1
      2
      3
      a
      b
      c
      q
// to input a String
Class Readlines
      {
            public static void main(String args[ ])
            throws IOException
      {
BufferReader br = new BufferReader(new InputStreamReader (System.in));
String str;
System.out.println("Enter lines of text.");
System.out.println("Enter 'stop' to quit".);
do {
            str = br.readline ( )
            System.out.println (str);
            } while (!str.equals ("Stop");
      }
}
```

The above programs demonstrate BufferReader and the two input methods, read () and readline ().

2.2.3 Getting values of Variables

A computer program is written to manipulate the given data and to give output (result). Java supports two output methods that can be used to send the results to the screen.

- print ( ) : to display the String without following the new line.
- println ( ) : to display a string followed by a new line character.

The print ( ) method sends information into a buffer. This buffer is flushed only when a newline (or end-of-line) character is sent. As a result, the print ( ) method prints output on one line until a newline character is encountered. For example, the statements:

```
System.out.print("Let us");
System.out.print("Learn Java".);
```

will display the words "Let us Learn Java" on one line and waits for displaying further information on the same line. The display can be brought to the new line if we write:

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

in between the two lines. Then the output will be as follows:

```
Let us
Learn Java
```

The println( ) method takes the information provided and displays it on a line followed by a line feed (carriage return). Therefore, the statement:

```
System.out.println("Let us");
System.out.println("Learn Java");
```

will display the following output:

```
Let us
Learn Java.
```

The statement

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

will print a blank line.

2.2.4 Standard Default Values

Every variable has a default value in Java which the variables are assigned to if we do not initialize a variable. Table 2.1 shows the variables types along with their default values.

Table 2.1: Default Values for Variable Types

Type of Variable	Default Value
Byte	Zero : (byte) 0
Short	Zero : (short) 0
int	Zero : 0
Long	zero OL
Float	0.0 f
Double	0.0d
Char	null character
Boolean	False
Reference	Null

2.2.5 Scope of Variables

Define Scope

There are three kinds of variables in Java– instance variables, class variables, local variables. Java does not have global variables (which can be used in all parts of a program). Instance variables and class variables are used to communicate information from one object to another and these replace the need for global variables. Local variables are used inside method definitions or even smaller blocks of statements within a method. The modifiers define the scope of variables. A variable declared without any access control modifier is available to any other class in the same package. To completely hide a variable from being used by another class use PRIVATE.

In some cases, you may want a variable in a class to be completely available to any other class that wants to use it, then it should be declared as public.

2.2.6 Symbolic Constants

Variables are useful when you need to store information that can be changed as a program runs. If the value should never change during a program's run time, you can use a special type of variable called a constant. It is used when the value need not to be changed. Variables are also useful in defining shared values for all methods of an object. To declare a constant use the final keyword before the variable declaration. For example

```
final float Pi = 3.141592
```

2.3 DATA TYPES IN JAVA

2.3.1 Primitive Data Types

Java has eight primitive data types.

Reserved Word	Data Type	Size	Range of Values
byte	Byte length integer	1 byte	-2^8 to 2^7 -1
short	Short integer	2 bytes	-2^{16} to 2^{16} -1
int	Integer	4 bytes	-2^{32} to 2^{31} -1
long	long integers	8 bytes	-2^{64} to 2^{63}-1
float	Single precision	4 bytes	-2^{32} to 2^{31} -1
double	Real number with double	8 bytes	-2^{64} to 2^{62} -1
char	character (16 bit unicode)	2 bytes	0 to 216 -1
boolean	has value true or false	A boolean value	true or false

2.3.2 Non-primitive Data Type

Data types which are built with the help of primitive data types but are not actually primitive are known as non primitive data types.

Literals

Literals are pieces of Java source code that indicate explicit values. For instance "Hello World!" is a string literal and its meaning is the words Hello World. Java has four different types of literals

- ***String Literal:*** The string literal is always enclosed in double quotes. Java uses a String Class to implement strings, whereas C and C++ use an array of characters. For example

```
"Hello World".
```

- ***Character Literal:*** Character literals are similar to string literals except that they are enclosed in single quotes and must have exactly one character. For example 'C' is a character literal that means the letter C.
- ***Boolean Literal:*** Boolean literal can have either of the values: true or false. They do not correspond to the numeric values 1 and 0 in C and C++.
- ***Numeric Literals:*** There are two types of numeric literals: integers and floating point numbers. For example 34 is an integer literal and it means the number thirty four. 1.5 is a floating point literal.

Character

Characters in Java are a special set. They can be treated as either a 16 bit unsigned integer with a value from 0-655 35 or as a unicode character. The unicode standard makes room for the use of alphabets of many different languages. The syntax is

```
Char ch = 'b';
```

- ***Boolean:*** A Boolean variable is not a number. It can have one of two values: true or false. There are two ways to set it.

For example

```
mybool is a variable To change the variable to false type
mybool = false;
    The second way is to assign the same value as in another variable
mybool = mybool1;
    You can also make the variable have a value based on the equality of
other numbers
mybool = 6 > 7;
```

2.4 TYPE CONVERSION AND CASTING

Type Casting

An int divided by an int is still an int and a double divided by a double is still a double. But what about an int divided by double or a double divided by an int? When doing arithmetic on unlike types, Java tends to widen the types involved so as to avoid loosing information.

The basic rule is that if any of the variables on the right hand side of an equal sign is double then Java treats all the values on the right hand side as doubles. If none of those values are double but some are floats, then Java treats all the values as floats. If there are no floats or doubles but there are longs, then

Java treats all the values as longs. Finally, if there are no doubles, floats or longs, then Java treats everything as int.

However, if the right hand side may not be able to fit into the left hand side, then a series of operations takes place to chop the right hand side down to size. For a conversion between a floating point number and an int or a long, the fractional part of the floating point number is truncated. For example

```
int i = (int) 9.0/4.0
```

The result will be converted to integer.

Converting Strings to Numbers

```
int i = integer.valueOf ("22").intValue( );
long l = long.valueOf ("22").longValue( );
double x = double.valueOf ("22").doubleValue( );
```

Type Conversion

When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

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

When these two conditions are met, a widening conversion takes place. For example, an int type is always large enough to hold all valid byte values, so no explicit statement is required.

For widening conversions, the numeric types including integer and floating point types are compatible with each other. However, the numeric types are not compatible with char or boolean. Also char and boolean are not compatible with each other. Java also performs an automatic type conversion when storing a literal Integer constant into variables of type byte, short or long. Let's consider a program to understand typecasting.

```
//Demonstrates type casting.
Class Conversion {
    public static void main(String args[ ])
    {
        byte b;
        int i = 327;
        double d = 423.124;
        System.out.println("In Conversion of int to byte".);
        b= (byte)i;
        System.out.println("i and b" + i "+" " " +b);
        System.out.printin("\n conversion of double to int.");
        i = (int)d;
        System.out.println("\n Conversion of double to byte".);
```

```
        b = (byte)d;
        System.out.println("d and b" +d+ " " +b);
    }
}
```

The output of the above program is as follows:

```
Conversion of int to byte.
i and b  327  71
Conversion of double to int.
d and i  423.124  423
Conversion of double to byte.
d and b  423.124  167
```

Here, when the value 327 is cast into a byte variable, the result is the remainder of the division of 327 by 256 (the range of a byte), which is 71 in this case. When 'd' is converted to an int, its fractional component is lost. When 'd' is converted to a byte, its fractional component is lost, and the value is reduced modulo 256, which is 167 in this case.

2.5 ARRAYS

An array is a group of like typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions. A specific element in an array is accessed by its index.

2.5.1 One Dimensional Array

A one dimensional array is essentially a list of like typed variables. To create an array you must first create an array variable of the desired type. The general form of a one-dimensional array declaration is

```
type var-name [ ];
int month-days [ ];
// Demonstrate a one dimensional array clas array {
public static void main(String s[ ])
{
    int month-days[ ];
    month-days[0] = 31;
    month-days[1] = 28;
    month-days[2] = 31;
    System.out.println ("march has" + month-days[3]+ "days"];
}
}
class promote  {
```

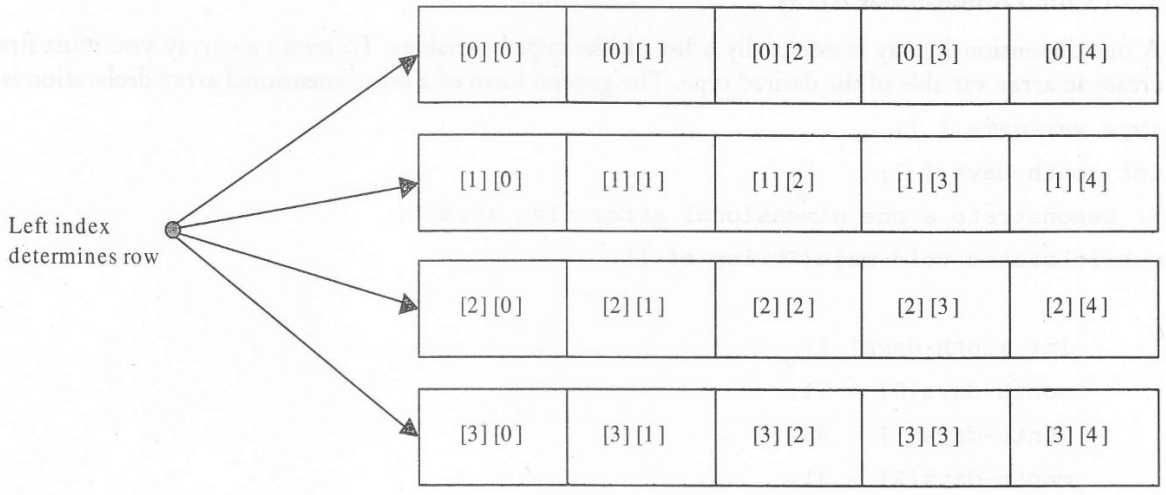
```
public static void main(String args[ ])
{
      byte b = 42;
      Char c = 'a';
      Short s = 1024;
      int i = 50000;
      float f = 5.67f
      double d = .1234;
      double result = (f*b) + (i/c) - (d*s);
      System.out.println (f*b) + "+" + (i/c) + " - " (d*x));
      System.out.println ("result = " + result);
}
}
```

2.5.2 Multidimensional Arrays

In Java, multimensional arrays are actually arrays of arrays. As you might expect, they look and act like regular multidimensional arrays. However as you will see there are a couple of subtle differences. To declare a multidimensional array variable, specify each additional index using another set of square brackets.

```
      inttwoD[ ] [ ] = new int [4] [5];
```

This allocates a 4 by 5 array and assigns it to twoD.




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