MODULE 1

An Overview of Java:

- Object Oriented Programming
- A First Simple Program
- A Second Short Program
- Two Control statements
- Using Blocks of code
- Lexical Issues
- The Java Class Libraries

Data Types, Variables and Arrays:

- Java is a Strongly typed Language
- The Primitive types.
- Integers
- Floating-Point types
- Characters
- Booleans
- A Closer look at Literals
- Variables
- Type conversion and casting
- Automatic Type promotion in Expressions
- Arrays
- A few words about strings

Object Oriented Programming:

- Object oriented programming (OOP) is the core of Java programming.
- Java is a general purpose, object- oriented programming language developed by Sun Microsystems. It was invented by James Gosling and his team and was initially called as Oak.
- The most important feature that made Java very popular was the "Platform-Independent" approach.
- It was the first programming language that did not tie-up with any particular operating system(or hardware) rather Java programs can be executed anywhere and on any system.
- Java was designed for the development of the software for consumer electronic devices like TVs, VCRs, etc.

Two Paradigms:

- Every program contains 2 components code and data.
- Two approaches are there to solve the problem and in program writing: Procesure oriented and object oriented.

Procedure Oriented:

- Procedure oriented programs are written based on "whats happening" around, where the code acts on data. Ex: C etc
- Problems increases in procedure oriented as the program grows larger and more complex.

Object Oriented:

- Object oriented programs are written based on "Who is being affected" around, which manages the increasing complexity.
- It organises program around data and well defined interfaces of that data.
- Characterised as data controlling access to code. Ex: C++, JAVA, Small Talk etc

The Three OOP:

The three important features of OOP are:

- Encapsulation
- Inheritence
- Polymorphism

Encapsulation:

- Encapsulation is the mechanism that binds together code and data it manipulates, and keeps both safe from outside interference and misuse.
- In Java the basis of encapsulation is the class. A class defines the state and behavior(data & code) that will be shared by set of objects.
- Each object contains the structure and behavior defined by the class. The data defined by the class are called instance variables(member variables), the code that operates on that data are called methods(member functions).

Inheritence:

- Inheritence is the process by which one object acquires the properties of another object. This is important as it supports the concept of hierarchical classification.
- By the use of inheritence, a class has to define only those qualities that make it unique. The general qualities can ber derived from the parent class or base class.
- Ex: A child inheriting properties from parents.

Polymorphism

- Polymorphism (meaning many forms) is a feature that allows one interface to be used for a general class of actions. The specific action determined by the exact nature of the situation. This concept is often expressed as " one interface, multiple methods".
- Ex: "+" can be used for addition of 2 numbers and also concatenation of 2 strings.

System.out.println(2+4); // outputs 6 as answer

System.out.println("Hello" + "Gautham"); // outputs Hello Gautham as answer

Apart from this the additional features include:

Object:

- An object can be any real world entity.
- Ex: an animal, bank, human, box, fan etc
- An object is a software bundle of related state and behavior.
- An object is an instance of class.

Class:

- A class is a blueprint or prototype from which objects are created.
- Its just a template for an object, which describes an object.
- Ex: a class describes how an animal looks like.
- A class is a user defined data type.

Abstraction:

- Data abstraction refers to providing only essential information to the outside world and hiding their background details i.e., to represent the needed informatin in program without presenting the details.
- Ex: a database system hides certain details of how data is stored and created and maintained.

Polymorphism, Encapsulation and Inheritence work Together

- The 3 principles of OOP Polymorphism, Encapsulation and Inheritence combines together to make the programming robust and scalable.
- Encapsulation allows to migrate the implementation without disturbing the code that depends on class.
- Polymorphism allows to create clean, sensible, readable, resilient code.
- Inheritence mainly deals with the code reusability.

A First Simple Program

```
class Example
{
        public static void main(String args[])
        {
                System.out.println("Welcome to Programming in Java");
        }
}
```

1. **Open the notepad and type the above program**

2. Save the above program with **.java** extension, here file name and class name should be same,

   ex: Example.java

3. Open the command prompt and **Compile** the aboveprogram

   javac Example.java

   From the above compilation the java compiler produces a bytecode(.class file)

4. Finally run the program through the

   interpreter java Example.java

Output of the program:

Welcome to Programming in Java

Note:

- In Java all code must reside inside a class and name of that class should match the name of the file that holds the program.
- Java is case-sensitive

Compiling the program

- To compile the program, execute the compiler "javac", specifying the name of the source file on the command line as shown below

  C:\> javac Example.java

- The "javac: compiler creates a file called "Example.class" that contains the bytecode version of the program.
- To run the program, we must use the java interpreter called "java". To do so we pass the class name "Example" as a command-line argument as shown below

  C:\> java Example

- When you run the program we get the output:

  Welcome to Programming in Java

Description:

(1) **Class declaration:** "class Example" declares a class, which is an object- oriented construct. Sampleone is a Java identifier that specifies the name of the class to be

defined.

(2) **Opening braces:** Every class definition of Java starts with opening braces and ends with matching one.

(3) **The main line:** the line " public static void main(String args[]) " defines a method name main. Java application program must include this main. This is the starting point of the interpreter from where it starts executing. A Java program can have any number of classes but only one class will have the main method.

(4) **Public:** This key word is an access specifier that declares the main method as unprotected and therefore making it accessible to the all other classes.

(5) **Static:** Static keyword defines the method as one that belongs to the entire class and not for a particular object of the class. The main must always be declared as static.

(6) **Void:** the type modifier void specifies that the method main does not return any value.

(7) **The println:** It is a method of the object out of system class. It is similar to the printf or cout of c or c++. This always appends a newline character to the end of the string i.e, any subsequent output will start on a new line.

A Second Short Program

```
/* This is a short example
      Name of file : Example2.java */
class Example2{
   public static void main(String args[])
{
     int n=3;
     System.out.println(" the value of n is "+n);
     n=n+5;
     System.out.print(" the new value is");
     System.out.println(n);
}
}
```

Output:

the value of n is 3

the new value of n is 8

The statement System.out.println(" the value of n is "+n), the sign "+" causes the value of "n" to be appended to the string that preceeds it, and the resulting string is output.(Actually n is first converted from an integer into its string equivalent and the concatenated with the string that preceeds it)

The System.out.print( ) method is just like println( ) except that it does not output a newline character after each call.

Two Control Statements

Here in this chapter initially we focus on two control statements if and for loop , the detailed control statements will be discussed in module 2.

if statement

- The if- statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code *only if* a particular test evaluates to true.

- Here is the general form of the **if** statement:

  if (*condition*) *statement*;

- Here the condition is Boolean expression.
- If the condition is true then the statement is executed, if false then statement will be skipped.

Example:

```
class Example
{
    public static void main(String args[])
    {
        int a=10;
        if(a>0)
          System.out.println("a is positive number");
        System.out.println(" End of program");
    }
```

```
}
```

In the above program since a is greater than o it prints the output as

a is positive number

End of program

If incase a is -1 or negative value the condition fails and it prints only

End of program

The for loop

- The for loop is similar to that of C/C++
- Here is the general form of the traditional **for** statement:

```
for(initialization; condition; iteration)
{
        //body
}
```

- Initialization sets the loop control variable to initial value.
- Condition is a Boolean expression which tests the loop
- Iteration expression tells hoe the control variable has to change at each iteration. Generally the increment or decrement operator is used to perform iteration.

Example:

```
class Example
{
    public static void main(String args[])
    {
        int a;
        for(a=0;a<5;a++)
          System.out.println(a);
        System.out.println(" End of program");
    }
}
```

Output:

0

1

2

3

4

End of Program

Using blocks of code

- Java supports **code blocks -** which means that two or more statements are grouped into blocks of code.
- Opening and closing braces is used to achieve this.
- Each block is treated as logical unit.
- Whenever two or more statements has to be linked blocks can be used.

Example:

```
class Example
{
        public static void main(String args[])
        {
            int a=10;
            if(a>0)
            { // begin of block
              System.out.println("a is positive number");
               System.out.println(" inside block");
            }// end of block
        }
}
```

Lexical issues:

Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.

Whitespace:

- Java is a free from language- means no need to follow any indentation rules.
- Whitespace is a space, tab, or newline.

Java character set:

- The smallest unit of Java language are its character set used to write Java tokens. This character are defined by unicode character set that tries to create character for a large number of character worldwide.
- The Unicode is a 16-bit character coding system and currently supports 34,000 defined characters derived from 24 languages of worldwide.

Key Words:

Java program is basically a collection of classes. A class is defined by a set of declaration statements and methods containing executable statements. Most statement contains an expression that contains the action carried out on data. The compiler recognizes the tokens for building up the expression and statements. Smallest individual units of programs are known as tokens. Java language includes five types of tokens. They are

(a) Reserved Keyword
(b) Identifiers
(c) Literals.
(d) Operators
(e) Separators.

Reserved keyword:

Java language has 50 words as reserved keywords. They implement specific feature of the language. The keywords combined with operators and separators according to syntax build the Java language.

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

Identifiers:

Identifiers are programmer-designed token used for naming classes methods variable, objects, labels etc. The rules for identifiers are

1. They can have alphabets, digits, dollar sign and underscores.
2. They must not begin with digit.
3. Uppercase and lower case letters are distinct.
4. They can be any lengths.
5. Name of all public method starts with lowercase.
6. In case of more than one word starts with uppercase in next word.
7. All private and local variables use only lowercase and underscore.
8. All classes and interfaces start with leading uppercases.
9. Constant identifier uses uppercase letters only.

Example for valid identifiers:

Var_1, count, $value etc

Example for invalid identifiers:

6name, var@value, my/name etc

Literals:

Literals in Java are sequence of characters that represents constant values to be stored in variables. Java language specifies five major types of Literals. They are:

1. Integer Literals.
2. Floating-point Literals.
3. Character Literals.
4. String Literals.
5. Boolean Literals.

Operators:

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

Separators:

Separators are the symbols that indicates where group of code are divided and arranged.

Some of the operators are:

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

Comments:

- Java supports 3 styles of comments
- **Multiline comment:** this type of comment begins with /* and ends with */

  Ex: /* Welcome to
  Java Programming */
- **Single line comments:** this type of comment begins with // and ends at the end of current line

  Ex: // Welcome to java Programming
- **Documentation Comment:** this type of comment is used to produce an HTML file that documents your program. The documentation comment begins with /** and ends with */

Java Class libraries:

Java environment has several built in class libraries.

Java standard library includes hundreds of classes and methods grouped into several functional packages. Most commonly used packages are:

(a) Language support Package.

(b) Utilities packages.

(c) Input/output packages

(d) Networking packages

(e) AWT packages.

(f) Applet packages.

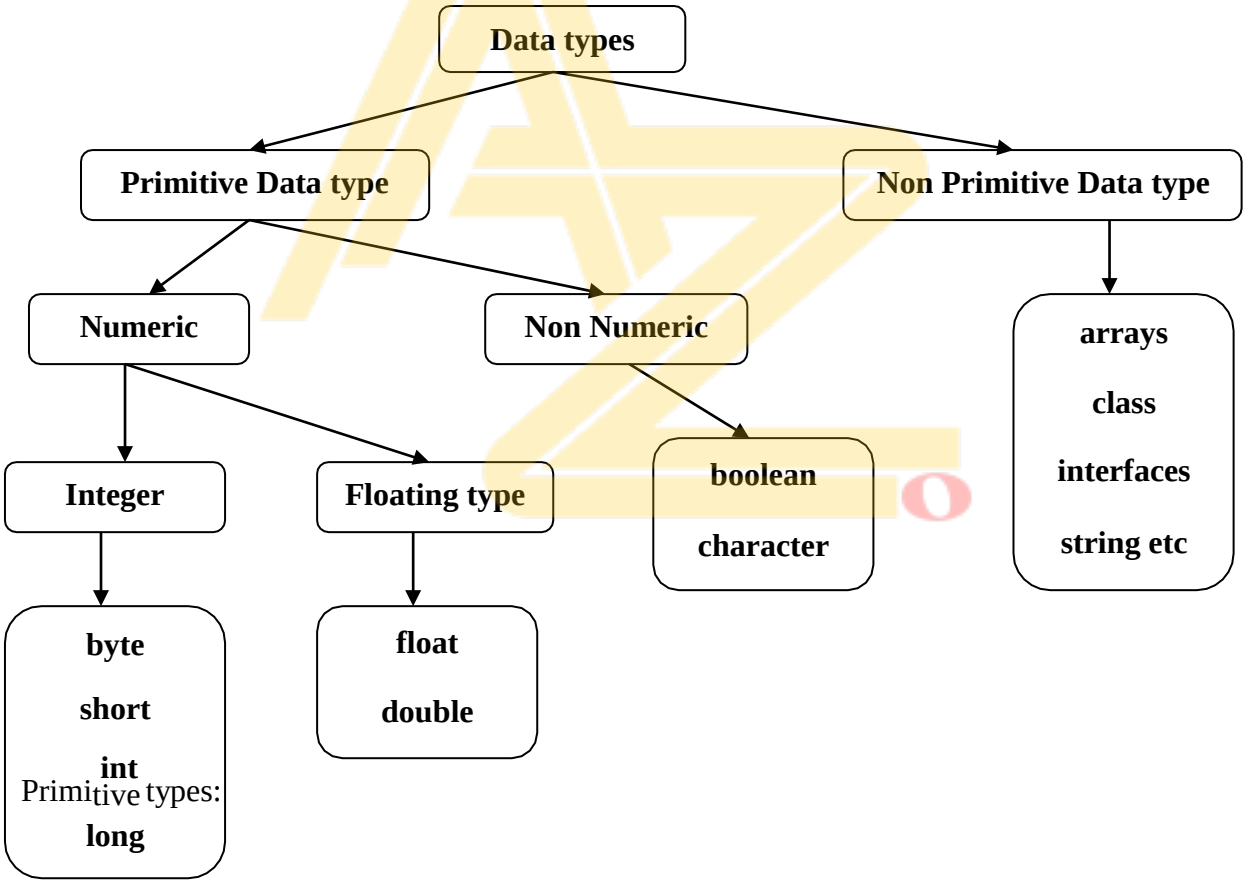
Data types, variables and arrays

Java is a strongly typed language:

- The strongly typed nature of Java gives it the robustness and safety for it.
- Every variable and expression has strictly defined type.
- Assignments, parameter passing or explicit value passing are checked for type compatibility.
- Java compiler checks all expressions and parameters to ensure type compatibility.

Data types

The various data types supported in java is as follows



Java defines eight *primitive* types of data: **byte**, **short**, **int**, **long**, **char**, **float**, **double**, and **boolean**. As shown in above figure.

- The primitive types represent single values—not complex objects. Although Java is otherwise completely object-oriented, the primitive types are not.

- They are analogous to the simple types found in most other non–object-oriented languages.
- The reason for this is efficiency. Making the primitive types into objects would have degraded performance too much. The primitive types are defined to have an explicit range and mathematical behavior.
- Because of Java's portability requirement, all data types have a strictly defined range. For example, an **int** is always 32 bits, regardless of the particular platform.

Integers

- Java defines four integer types: **byte**, **short**, **int**, and **long**.
- All of these are signed, positive and negative values. Java does not support unsigned, positive-only integers.
- Many other computer languages support both signed and unsigned integers.
- However, Java's designers felt that unsigned integers were unnecessary. Specifically, they felt that the concept of *unsigned* was used mostly to specify the behavior of the *high-order bit,* which defines the *sign* of an integer value.

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

byte

- The smallest integer type is **byte**.
- This is a signed 8-bit type that has a range from –128 to127.
- Variables of type **byte** are especially useful when you're working with a stream of data from a network or file.
- Byte variables are declared by use of the **byte** keyword.
- For example, the following declares two **byte** variables called **b** and **c**: byte b, c;

short

- **short** is a signed 16-bit type.
- It has a range from –32,768 to 32,767.
- It is probably the least-used Java type.

- Here are some examples of **short** variable declarations:

```
short s;
short t;
```

Floating-Point Types

- Floating-point numbers, also known as *real* numbers, are used when evaluating expressions that require fractional precision.
- For example, calculations such as square root, or transcendental such as sine and cosine, result in a value whose precision requires a floating-point type.
- There are two kinds of floating-point types, **float** and **double**, which represent single- and double-precision numbers, respectively.

float

- The type **float** specifies a *single-precision* value that uses 32 bits of storage.

double

- Double precision, as denoted by the **double** keyword, uses 64 bits to store a value.
- Double precision is actually faster than single precision on some modern processors that have been optimized for high-speed mathematical calculations.

Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

Characters

- In Java, the data type used to store characters is **char**.
- However, C/C++ programmers beware: **char** in Java is not the same as **char** in C or C++.
- In C/C++, **char** is 8 bits wide. This is *not* the case in Java. Instead, Java uses Unicode to represent characters.
- *Unicode* defines a fully international character set that can represent all of the characters found in all human languages.
- It is a unification of dozens of character sets, such as Latin, Greek Arabic, Cyrillic,Hebrew, Katakana, Hangul, and many more. For this purpose, it requires 16 bits.

- Thus, in Java **char** is a 16-bit type. The range of a **char** is 0 to 65,536. There are no negative

Booleans:

Java has a simple type called **boolean** for logical values. It can have only one of two possible values. They are true or false.

Data Type	Default Value	Default size
boolean	False	1 bit
Char	'\u0000'	2 byte
Byte	0	1 byte
short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte
double	0.0d	8 byte

Literals:

A constant value in Java is created by using a literal representation of it. There are 5 types of literals.

- Integer Literals.
- Floating-point Literals.
- Character Literals.
- String Literals.
- Boolean Literals.

Integer literals:

- Any whole number value is an integer literal.
- These are all decimal values describing a base 10 number.
- There are two other bases which can be used in integer literal, octal(base 8) where 0 is prefixed with the value, hexadecimal (base 16) where 0X or 0x is prefixed with the integer value.

Example:

```
int decimal = 100;
int octal = 0144;
int hexa =  0x64;
```

Floating point literals:

- The default type when you write a floating-point literal is double, but you can designate it explicitly by appending the D (or d) suffix
- However, the suffix F (or f) is appended to designate the data type of a floating-point literal as float.
- We can also specify a floating-point literal in scientific notation using Exponent (short E ore), for instance: the double literal 0.0314E2 is interpreted as:

Example:

$0.0314 * 10^2$ (i.e 3.14).

6.5E+32 (or 6.5E32) Double-precision floating-point literal

7D Double-precision floating-point literal

.01f Floating-point literal

Character literals:

- char data type is a single 16-bit Unicode character.
- We can specify a character literal as a single printable character in a pair of single quote characters such as 'a', '#', and '3'.
- You must know about the ASCII character set. The ASCII character set includes 128 characters including letters, numerals, punctuation etc.
- Below table shows a set of these special characters.

Escape	Meaning
\n	New line
\t	Tab
\b	Backspace
\r	Carriage return
\f	Formfeed
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\d	Octal
\xd	Hexadecimal
\ud	Unicode character

Boolean Literals:

- The values true and false are treated as literals in Java programming.
- When we assign a value to a boolean variable, we can only use these two values.
- Unlike C, we can't presume that the value of 1 is equivalent to true and 0 is equivalent to false in Java.
- We have to use the values true and false to represent a Boolean value.

Example

boolean chosen = true;

String Literal

- The set of characters in represented as String literals in Java.
- Always use "double quotes" for String literals.
- There are few methods provided in Java to combine strings, modify strings and to know whether to strings have the same values.

Example:

“hello world”

“Java”

Variables:

A variable is an identifier that denotes a storage location used to store a data value. A variable may have different value in the different phase of the program. To declare one identifier as a variable there are certain rules. They are:

1. They must not begin with a digit.
2. Uppercase and lowercase are distinct.
3. It should not be a keyword.
4. White space is not allowed.

Declaring Variable: One variable should be declared before using.

The syntax is

type identifier [= *value*][, *identifier* [= *value*] ...] ;

Example:

```
int a,b,c;
float quot, div;
```

Initializing a variable:A variable can be initialize in two ways. They are

(a) Initializing by Assignment statements.

(b) Dynamic Initialisation

Initializing by assignment statements:

- One variable can be initialize using assignment statements. The syntax is :

Variable-name = Value;

Example: int a=10,b,c=16;

Double pi=3.147;

Dynamic initialization:

- Java allows variables to be initialized dynamically, using expression valid at the time variable is declared.

Example:

```
class Example
{
    public static void main(String args[])
    {
        double a=10, b=2.6;
        double c=a/b;
        System.out.println("value of c is"+c);
    }
}
```

The Scope and Lifetime of Variables

- Java allows variables to be declared within any block. A block is begun with an opening curly brace and ended by a closing curly brace. A block defines a *scope.*
- A scope determines what objects are visible to other parts of your program. It also determines the lifetime of those objects.
- Many other computer languages define two general categories of scopes: **global** and **local**. However, these traditional scopes do not fit well with Java's strict, object-oriented model.
- As a general rule, variables declared inside a scope are not visible (that is, accessible) to code that is defined outside that scope. Thus, when you declare a variable within a

scope, you are localizing that variable and protecting it from unauthorized access and/or modification.

```
class Scope
{
public static void main(String args[])
{
        int x; // known to all code within main x = 10;
        if(x == 10)             // start new scope
        {
                int y = 20;    // known only to this block
                                // x and y both known here.
                System.out.println("x and y: " + x + " " + y); x = y * 2;
        }
        // y = 100; // Error! y not known here
        // x is still known here. System.out.println("x is " + x);
}
}
```

Note:

- ***There should not be two variables with the same name in different scope.***
- ***The variable at outer scope can be accessed in inner scope but vice versa is not possible.***

Type Conversion and casting

It is often necessary to store a value of one type into the variable of another type. In these situations the value that to be stored should be casted to destination type. Assigning a value of one type to a variable of another type is known as **Type Casting** .Type casting can be done in two ways.

In Java, type casting is classified into two types,

1. Widening Casting(Implicit)

2. Narrowing Casting(Explicitly done)

Widening or Automatic type converion

Automatic Type casting take place when,

- the two types are compatible
- the target type is larger than the source type

Example :

```
public class Test
{
    public static void main(String[] args)
    {
        int i = 100;
        long l = i;          //no explicit type casting required
        float f = l;         //no explicit type casting required
        System.out.println("Int value "+i);
        System.out.println("Long value "+l);
        System.out.println("Float value "+f);
    }

}
```

Output :

Int value 100

Long value 100

Float value 100.0

Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

Example :

```
public class Test
{
      public static void main(String[] args)
      {
         double d = 100.04;
         long l = (long)d;          //explicit type casting required
         int i = (int)l;            //explicit type casting required
         System.out.println("Double value "+d);
         System.out.println("Long    value    "+l);
         System.out.println("Int value "+i);

      }

}
```

Output :

Double value 100.04

Long value 100

Int value 100

Automatic type promotion in expressions:

- Type conversions also occurs in expressions.
- Java automatically promotes each byte, short, or char operand to int when evaluating an expression.

```
byte b = 50;
b = b * 2; // Error! Cannot assign an int to a byte!
```

the operands were automatically promoted to int when the expression was evaluated, the result has also been promoted to int. Thus, the result of the expression is now of type int, which cannot be assigned to a byte without the use of a cast.

```
byte b = 50;
b = (byte)(b * 2); which yields the correct value of 100.
```

Java defines several type promotion rules that apply to expressions. They are as follows:

- First, all byte, short, and char values are promoted to int, as just described.
- Then, if one operand is a long, the whole expression is promoted to long.
- If one operand is a float, the entire expression is promoted to float.
- If any of the operands is double, the result is double.

Arrays in Java

Arraywhich stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Declaring Array Variables:

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
dataType[] arrayRefVar;        or        dataType arrayRefVar[];
```

Example:

The following code snippets are examples of this syntax:

```
int[] myList;                  or                int myList[];
```

Creating Arrays:

You can create an array by using the new operator with the following syntax:

arrayRefVar = new dataType[arraySize];

The above statement does two things:

- It creates an array using new **dataType[arraySize];**
- It assigns the reference of the newly created array to the variable **arrayRefVar**.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

dataType[] arrayRefVar = new dataType[arraySize];

Alternatively you can create arrays as follows:

dataType[] arrayRefVar = {value0, value1, ..., valuek};

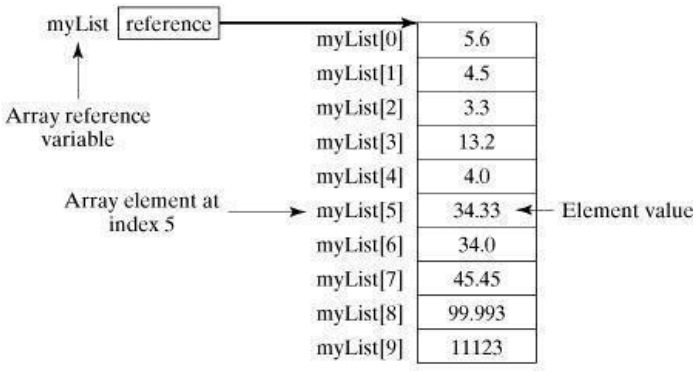
The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayRefVar.length-1**.

Example:

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:

double[] myList = new double[10];

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



Processing Arrays:

When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

Example:

Here is a complete example of showing how to create, initialize and process arrays:

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

                // Print all the array elements
                for (int i = 0; i < 4; i++)
                {
                System.out.println(myList[i] + " ");
        }
}
```

Multidimensional Arrays

Java does not support multidimensional arrays. However, you can declare and create an array of arrays (and those arrays can contain arrays, and so on, for however many dimensions you need), and access them as you would C-style multidimensional arrays:

```
int coords[] [] = new int[12] [12];
```

coords[0] [0] = 1; coords[0] [1] = 2;

A few words about strings:

- Java supports string type which is an object. It is used to declare string variables
- Array of strings can also be declared.
- A string variable can be assigned to another string variable.
- String variable can also be used as argument.

  Example:

  String name1=”gautham”, name2;

  Name2=name1; // sets name2 withvalue gautham

  System.out.println(name2); // string variable passed as parameter.

Programs:

// Compute distance light travels using long variables.

```
class Light
{
        public static void main(String args[])
        {
                int lightspeed;
                long days;
                long seconds;
                 long distance; // approximate speed of light in miles per second
                 lightspeed = 186000;
                days = 1000; // specify number of days here
                seconds = days * 24 * 60 * 60; // convert to seconds
                distance = lightspeed * seconds; // compute distance
                System.out.print("In " + days);
                System.out.print(" days light will travel about ");
                System.out.println(distance + " miles.");
        }
}
```

This program generates the following output:

In 1000 days light will travel about 16070400000000 miles.

// Compute the area of a circle.

```
class Area
{
        public static void main(String args[])
        {
        double pi, r, a; r = 10.8; // radius of circle
        pi = 3.1416; // pi, approximately
        a = pi * r * r; // compute area
        System.out.println("Area of circle is " + a);
        }
}
```

// Demonstrate char data type.

```
class CharDemo
{
        public static void main(String args[])
        {
                char ch1, ch2; ch1 = 88; // code for X
                ch2 = 'Y';
                System.out.print("ch1 and ch2: ");
                System.out.println(ch1 + " " + ch2);
        }
}
```

This program displays the following output:

ch1 and ch2: X Y

// char variables behave like integers.

```
class CharDemo2
{
        public static void main(String args[])
        {
                char ch1; ch1 = 'X';
```

```
            System.out.println("ch1 contains " + ch1);
            ch1++; // increment ch1
            System.out.println("ch1 is now " + ch1);
        }
}
```

The output generated by this program is shown here:

ch1 contains X ch1 is now Y

// Demonstrate boolean values.

```
class BoolTest
{
        public static void main(String args[])
        {
            boolean b;
            b = false;
            System.out.println("b is " + b);
            b = true;
            System.out.println("b is " + b); // a boolean value can control the if statement
            if(b)
                System.out.println("This is executed.");
            b = false;
            if(b)
                System.out.println("This is not executed.");
            System.out.println("10 > 9 is " + (10 > 9));
        }
}
```

The output generated by this program is shown here:

b is false

b is true

This is executed.

10 > 9 is true

Scope of variable

```
class LifeTime
{
        public static void main(String args[])
        {
                int x;
                for(x = 0; x < 3; x++)
                {
                        int y = -1; // y is initialized each time block is entered
                        System.out.println("y is: " + y); // this always prints -1
                        y = 100;
                        System.out.println("y is now: " + y);
                }
        }
}
```

The output generated by this program is shown here:

```
y is: -1
y is now: 100
y is: -1
y is now: 100
 y is: -1
 y is now: 100
```

Type conversion

```
class Conversion
{
        public static void main(String args[])
        {
                byte b;
                int i = 257;
                double d = 323.142;
                System.out.println("\nConversion of int to byte.");
                b = (byte) i;
                System.out.println("i and b " + i + " " + b);
```

```
        System.out.println("\nConversion of double to int.");
        i = (int) d;
        System.out.println("d and i " + d + " " + i);
        System.out.println("\nConversion of double to byte.");
        b = (byte) d; System.out.println("d and b " + d + " " + b);
    }
}
```

This program generates the following output:

Conversion of int to byte.

i and b 257 1

Conversion of double to int.

d and i 323.142 323

Conversion of double to byte.

d and b 323.142 67

Additional:

Applications of Java

- Some of the applications of Java is that internet users can use Java to create applet programs and run them using a web-browser.
- The first application program written in Java was HotJava, a web browser to run applet on internet.(An applet is a special kind of Java program that is designed to be transmitted over the internet and automatically executed by a Jva compatible web browser)
- Further internet users can also set up their websites containing java applets,that could be used by other remote users of the internet. Hence Java is popularly called as " Language of Internet".
- Before the invention of Java, world wide web was limited to displaying text and still images. However, the incorporation of Java into web pages has made web capable of supporting animations, graphics, games and wide range of special effects.

We can develop two types of Java application. They are:

(1). Stand alone Java application.

(2). Web applets.

Stand alone Java application: Stand alone Java application are programs written in Java to carry out certain tasks on a certain stand alone system. Executing a stand-alone Java program contains two phases:

(a) Compiling source coded into bytecode using javac compiler.

(b) Executing the bytecodede program using Java interpreter.

Java applet: Applets are small Java program developed for Internet application. An applet located on a distant computer can be downloaded via Internet and execute on local computer.

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 known as **Java Development Kit(JDK)**
- The classes and methods area apart of the **Java Standard Library (JSL)** also known as Application Program Interface (API)
- **JRE(java runtime environment)** consists of tools required to execute the java code, contianing JVM, runtime class libraries, user interface toolkits.

Java Development Kit:

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

- ➔ **applet viewer**: Enables us to run java applet (without actually using a 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.
- ➔ **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 to find errors in programs.

Java Standard Library(JSL):

Java standard library includes hundreds of classes and methods grouped into several functional packages. Most commonly used packages are:

(g) Language support Package.

(h) Utilities packages.

(i) Input/output packages

(j) Networking packages

(k) AWT packages.

(l) Applet packages.

JVM(Java Virtual Machine):

The concept of Write-once-run-anywhere (known as the Platform independent) is one of the important key feature of java language that makes java as the most powerful language. Not even a single language is idle to this feature but java is closer to this feature. The programs written on one platform can run on any platform provided the platform must have the JVM (**Java Virtual Machine**). A Java virtual machine (JVM) is a virtual machine that can execute Java bytecode. It is the code execution component of the Java software platform. The below figure shows the JVM

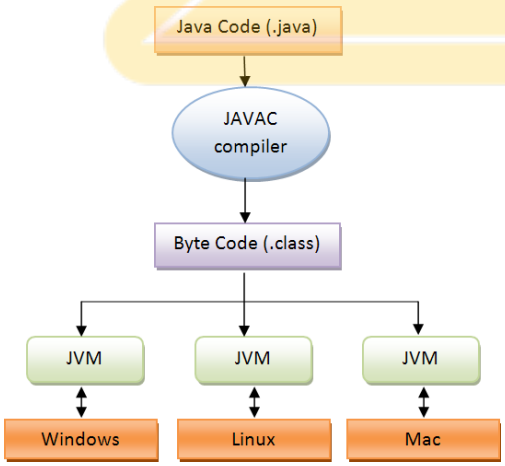


Fig: Java Virtual Machine

Java is iterpreted:

Java as a language initially gained popularity mainly due to its platform independent architecture or portaibility feature. The reason for Java to be portable is that it is interpreted. Firstly Java compiler translates source code into bytecode(an intermediate representation). This byte code is given as an input to the Java interpreter that generates the machine code that can be executed by the native system. So we call java as an interpreter language.

The Bytecode:

This concept allows java to solve both security and portability problems. When the source program is given as an input to the Java compiler, it will never generate the machine level language executable code, rather it generates "bytecode".

Bytecodes are highly optimized set if instructions designed to be executed by the java runtime system called JVM. Translating a java program into bytecode makes it much easier to run a program in a wide variety of environment because only the JVM needs to be implemented for each platform.

Java Features:

(1) Compiled and Interpreted
(2) Architecture Neutral/Platform independent and portable
(3) Object oriented
(4) Robust and secure.
(5) Distributed.
(6) Familiar, simple and small.
(7) Multithreaded and interactive.
(8) High performance
(9) Dynamic and extendible.

1. **Compiled and Interpreted**

   Usually a computer language is either compiled or interpreted. Java combines both these approaches; first java compiler translates source code into 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. **Architecture Neutral/Platform independent and portable**

   Java programs can be easily moved from one computer to another, anywhere and anytime. Changes in operating systems,

3. **Object oriented**

   In java everything is an Object. Java can be easily extended since it is based on the Object model. Java is a true object oriented language. All data resides in objects and classes

4. **Robust and secure.**

   Java is a robust language; Java makes an effort to eliminate error situations by emphasizing mainly on compile time error checking and runtime checking. Because of absence of pointers in java we can easily achieve the security.

5. **Distributed.**

   Java is designed for the distributed environment of the internet.java applications can open and access remote objects on internet as easily as they can do in the local system.

6. **Familiar, simple and small.**

   Java is designed to be easy to learn. If you understand the basic concept of OOP java would be easy to master.

7. **Multithreaded and interactive.**

   With Java's multi-threaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.

8. **High performance**

Because of the intermediate bytecode java language provides high performance

9. **Dynamic and extendible.**

Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Implementing a Java program: Java program implementation contains three stages. They are:

1. Create the source code.
2. Compile the source code.
3. Execute the program.

(1) Create the source code:

1. Any editor can be used to create the Java source code.
2. After coding the Java program must be saved in a file having the same name of the class containing main() method.
3. Java code file must have .Java extension.

(2) Compile the source code:

1. Compilation of source code will generate the bytecode.
2. JDK must be installed before completion.
3. Java program can be compiled by typing javac <filename>.java
4. It will create a file called <filename>.class containing thebytecode.

(3) Executing the program:

1. Java program once compiled can be run at any system.
2. Java program can be execute by typing Java <filename>