UNIT- I

An Overview of Java

* JAVA was conceived by James Gosling, Patrick Naughton, Chris Heath, Ed Frank & Mike Sheridan at Sun Microsystems Inc in 1991. Initially it was called "OAK" but renamed as Java in 1995.

* problems with 'C' & C++ is that they are not efficient & cost-effective (designed to be compiled for specific target).
* JAVA can run on variety of CPU's under differing environments.

The Byte Code.

→ The o/p of the Java compiler is not executable code. It is a Bytecode

→ Bytecode is a highly optimized set of instructions designed to be executed by the JAVA run-time system, which is called as Java Virtual Machine. (JVM).

→ JVM is an interpreter for Bytecode.

Object- oriented paradigms. (code & data are two paradigms).

→ OOP is the core of JAVA.

→ 'C' is procedure-oriented (i.e. sequential execution of code).

→ Object- oriented programming style will reduce the complexity.

→ Object- oriented programming organizes a program around its data & a set of well- defined interfaces to that data.

Abstraction:- An essential element of object- oriented program-ming. Complexity can be managed through abstraction.

* the data from a traditional process- oriented program

can be transformed by abstraction into its component objects. Thus, each of these objects describes its own unique behavior.

* objects are concrete entities that respond to messages telling them to do something.

The three OOP principles.

* the principles of OOP's are encapsulation, inheritance & polymorphism.

Encapsulation

* Encapsulation is the mechanism that binds together the code & the data it manipulates, & keeps both safe from outside interference & misuse.

→ It is a protective layer & wrapper that prevents the code & data from being arbitrarily accessed by other code defined outside the wrapper.

→ The Bases of encapsulation is a class.

* A class defines the structure & behavior (data & code) that will be shared by a set of objects.

* objects are referred to as the instances of a class.

* class is a logical construct & object has physical reality.

* the elements are called members of the class & the data defined by the class are called as member variables or instance variables.

* the codes that operate on the class are called member methods or just methods.

* Each Method or Variable in a class may be marked private or public.

* the public interface of a class represents everything that external users of the class need to know or may know.

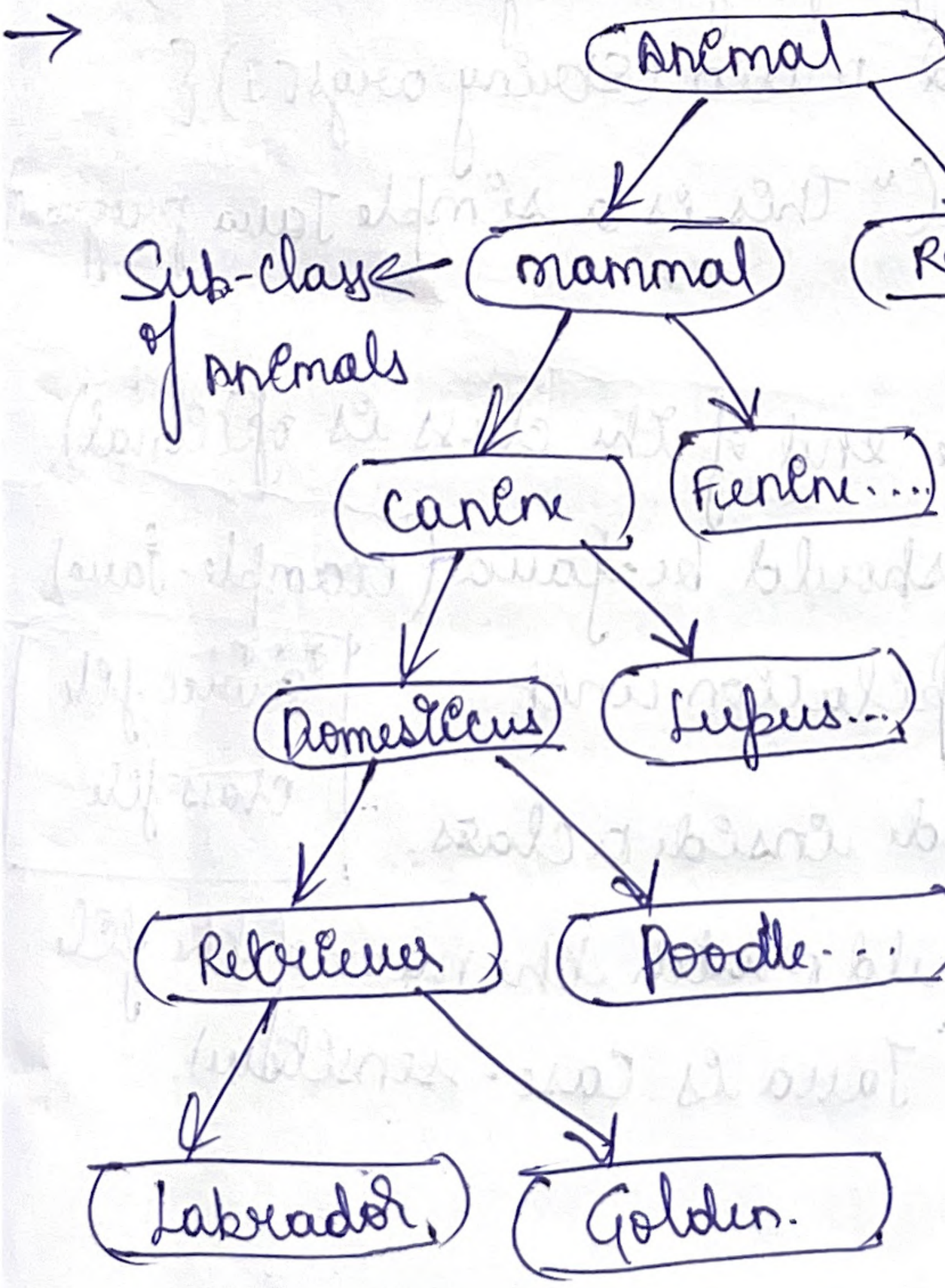
* the private methods & data can only be accessed by code that is a member of the class.

Inheritance :-

→ is the process by which one object acquires the proper-ties of another object.

For eg:- Animal → mammal → Dog → Golden Retrievers.

→ The general attributes can be inherited from its parent. The objects need to define only those qualities that make it unique inside the class.

→ 

* A deeply inherited subclass inherits all of the attributes from each of its ancestors, in the class hierarchy.

Polymorphism:- ("poly" means "many" in Greek) that allows one interface to be used for a general class of actions.

* The concept of polymorphism is often expressed by the phrase "one interface, multiple methods".

→ A general interface for group of related activities & it's the job of compiler to know which the related one.

Eg:- Same method name "area".

A First simple program.

```
/* This is a simple Java program.
   Call this file "Example.java".
*/
class Example {                    // your program begins with main()
public static void main(String args[]) {
    System.out.println("This is a simple Java program");
}
};
```

(Comments - Multi-line comments)

(Command line argument)

(Semi-colon at the end of the class is optional).

* Name of the source file should be .java (example.java).

→ The source file is a compilation unit.

* In Java, all code must reside inside a class.

** The name of the class should match the name of the file that holds the program. (Java is case-sensitive).

Source file
Class file

Compiling the program

→ To compile the Example program, execute the compiler, javac, specifying the name of the source file on command line as:

C:\> javac Example.java.

→ The javac compiler creates a file called Example.class that contains the bytecode version of the program. So, o/p of java compiler is not the executable code but a byte code.

→ C:\> java Example.

o/p is the program is:

This is a simple Java program.

→ In class Example {

↓ ↓

Keyword Identifier for name of the class.

(Note: * there will be a single class in Java).

* Single line comment //

→ Now, Lets consider. → No return value.

public static void main(String args[]){

* Execution starts from main() function.

* public is a keyword & is an access-specifier, which allows the programmer to control the visibility of class members.

* When a class member is preceded by public, then that member may be accessed by code outside the class in which it is declared.

* The keyword static allows main() to be called without having to instantiate a particular instance of a class.

→ This is necessary because main() is called by JVM before any objects are made.

* Any information that you need to pass to a method is received by variable specified within set of paran-theses that follow the name of the method. These vari-ables are called parameters.
* String args[] declares a parameter named args, which is array of instances of the class string.
* args receives command line arguments.
* println() is a o/p method. & it should end with ; (semi-colon)

System.out.println("This is a simple Java program");

System → is a predefined class.
out → output stream.

Features of JAVA

* The features of JAVA are called as Buzzwords.

1. Simple:- According to Sun Microsystems, Java language is simple because syntax is based on C++, & removed few confusing concepts such as explicit pointers, operator overloading etc & no need to remove unreferenced object because there is automatic garbage collection.

2. Object-Oriented:- OOP is a methodology that simplify S/W development & maintainance by providing some basic rules of OOP such as: Objects, class Inheritance, polymorphi-sm, Abstraction Encapsulation.

3. Platform Independent:- Platform is a H/W & S/W Environment in which a program runs. Java provides S/W based

platform. The two main components of JAVA are:-
i) Runtime Environment ii) Application programming Interface (API).

*Java code can be run on multiple platforms such as Windows, Solaris, Linux, mac OS etc. Java code is compiled by the compiler & converted into "Bytecode". The Bytecode is a platform Independent code. (write once run anywhere).

4. Secured:- Java is secured because 'no explicit pointer' & program run inside virtual m/c sendbox'. It is composed of the following:-
- class loadr:- Adds security by separating the package for classes of local file s/m from those that are imported from N/W sources.
- Byte verifier:- checks the code fragments for illegal code that can violate access rights to objects.
- Security Manager:- Determines what resources a class can access such as reading & writing to local disk.

5. Robust:- Robust simply means its strong. Java uses strong memory management. There are 'lack of pointers' that avoids security problem. The inclusion of concepts like exception handling, Type checking, Automatic garbage Collection makes Java Robust.

6. portable:- Byte code can be ported to any platform.

7. High performance:- Java is faster than traditional interpretation, since byte code is close to Native code.

8. Distributed:- He can create distributed applications in Java. He may access files by calling the methods from any machine on Internet.

Multithreaded:- A thread is like a separate program executing concurrently. We can write Multiple Java programs that deal with many tasks at once by defining multiple threads.

* The main advantage of multithreading is that it shares the same memory. Threads are important for multimedia, web applications etc.

Using Blocks of Code:-

* JAVA allows two or more statements to be grouped into blocks of code, also called code Blocks.
* This is done by enclosing the statements between opening & closing curly braces, & that becomes a logical unit.

Eg:-
```
if (x<y)  // begin a block.
{
  x=y;
  y=0;
} // end of block.
```
} These two statements cannot execute Individually (Becomes logical unit)

Lexical Issues

* Java programs are a collection of whitespace, Identifiers, literals, comments, operators, separators, & keywords.

Whitespace:- Java is a free-form language. They need not have to follow any special Indentations rules.
→ Whitespace is a space, tab or new line.

Identifiers:- Identifiers are used for class names, method names, & variable names.
→ An identifier may be any descriptive sequence.

of uppercase & lowercase letters, numbers & the underscore & dollar-sign characters.

* It must not begin with a number.

Eg:- a4, count, $test, this_is_ok. // valid, 2count // invalid.

Literals:- A constant value in Java is created by using a literal representation of it. For eg: here are some literals.

Eg:- 100 98.6 'X' "This is a Test"

Int literal Float literal Char const String.

Comments:- Single-line, multiline & documentation (/** ...*/)

Separators:- In Java, there are few characters that are used as separators. The most commonly used separator is semi-colon. (used to terminate statements)

The separators are shown in the following table:-

Symbol	Name	Purpose.
()	parentheses	used to contain lists of parameters in method definition & invocation. Also used for defining precedence in expressions, containing expressions in control statements, & surrounding cast types.
{}	Braces	used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, & local scopes.
[]	Brackets	used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.

3 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 sub-packages & classes. Also used to separate a variable or method from a reference variable

The Java Keywords

* There are 50 keywords currently defined in the Java language.
* The keywords cannot be used as names for a variable, class or method.
* The keywords const & goto are reserved but not used

Data Types, Variables & Arrays

Java is a strongly typed language.

i) Every variable has a type, every expression has a type & every type is strictly defined.

ii) All assignments, whether explicit or via param-eter passing in method calls, are checked for type compatability.

iii) There are no automatic conversions of conflict-ing types.

iv) The Java compiler checks all expressions & parameters to ensure that the types are compatible.

v) Any type mismatches are errors that must be corrected before the compiler will finish compiling the class.

The primitive Types. (simple Types)

→ Java defines eight primitive types of data: byte, short, int, long, char, float, double & boolean.

These can be put in four groups:

- Integers:- This group includes byte, short, int & long, which are for whole-valued signed numbers. (supports signed +ve & -ve values but doesn't support unsigned positive values).
- Floating-point numbers:- This group includes float & double, which represent numbers with fractional precision.
- characters:- This group includes char, which represents symbols in a character set like letters & numbers.
- Boolean:- This group includes boolean, which is a special type for representing true/false values.

* The primitive types represent single values - not complex objects & they have an explicit range along with mathematical behaviour.

The data types & their widths are as below:

long 64, int 32, short 16, byte 8. (-128 to 127).

i) Byte:- variables of type byte are especially useful when working with a stream of data from a N/W or a file.

* Byte variables are declared using the keyword "byte". Eg:- byte b, c;

ii) Short:- is a signed 16-bit type. It has range from -32768 to 32767.

Eg:- Short s; Short t;

Int:- The most commonly used Integer type is Int. Type Int is commonly employed to control loops & to index arrays.

Long:- Eg program

```
// Compute distance light travels using long variables.
class light{
public static void main(String args[]){
  int lightspeed;
  long days;
  long seconds;
  long distance;
  lightspeed = 186000;
  days = 1000;
  seconds = days * 24 * 60 * 60; // Convert to seconds.
  distance = lightspeed * seconds;
  System.out.println("in" + days);
System.out.println("days light will travel about");
System.out.println(distance + "miles.");
}
}
```

O/P:

In 1000 days light will travel about 160704 00000000 miles.

Floating-point types. (Also known as real numbers).

→ Fractional precision.

→ There are two kinds of Floating point type.

double → 64 (width in bits) float (32)

Float is used when you require a fractional component but don't require a large degree of precision.

Double:- is used when you need to maintain accuracy over many iterative calculations. It are manipulating large-valued numbers.

characters:- char in Java is not the same as char in C/C++. In C/C++ char is 8 bits but its 16 bits in Java, ranging from 0 to 65,536.

* There are no negative chars.

* Java uses unicode to represent characters.

* unicode defines a fully international character set that can represent all the characters found in all human languages. It is unification of dozens of charac-ter sets such as latin, Greek, Arabic, cyrillic, Hebrew, Katakana etc.

Eg:-

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

O/P:- Ch1 and Ch2 : X Y.

Booleans:- used for logical values i.e. T or F. It is the type returned by all relational operators, as in case of a<b.

```
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);
        if(b) system.out.println("This is executed");
        b = false;
[Note:- if(b==true) if(b) system.out.println("This is not executed");]
not required System.out.println("10>9 is "+(10>9));
    }
}
```

O/P:- b is false
b is true.
This is executed.
10>9 is true.

A closer look into literals.

* A Constant value in Java is represented using a literal representation of it. There are 5 types & they are as below:-

1) Integer literal:- Any whole number value is an integer

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

Eg:- int decimal = 100; int octal = 0144; int hexa = 0X64;

2) Floating point literals:- *The default type when you write a floating point literal is double, but you can designate it explicitly by appending D(or d) suffix.

* The suffix (F) or (f) is appended to designate the data type of a floating point literal as float.

* The Floating point number can be represented in scientific notation using Exponent (E or e) followed by decimal number). For eg:- double literal 0.0314E2 is interpreted as: 0.0314 * 10^2 (i.e. 3.14). 6.5E+32 (or 6.5E32) Double-precision floating-point literal.

3) Character literals:- Char data type is a single 16-bit unicode character. Char literal is represented within single quote 'a', '#', & '3'.

* ASCII character set includes 128 characters including letters, numerals, punctuation etc. Below table shows a set of these special characters.

Escape Sequence	Description.
\ddd	octal character ddd
\uxxxx	Hexadecimal encode(xxxx)
\'	single quote.
\"	Double quote.
\\	Backslash.

\r	carriage return.
\n	New line
\f	form feed.
\t	tab
\b	Backspace.

5) String literals:- The set of characters are represented as string literals in JAVA. Double quotes are used for string literals (" "). There are methods in Java to combine, Modify & Compare strings.

Eg:- "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. The rules for defining a variable as below:

i) They must not begin with a digit.
ii) Uppercase & lowercase are distinct.
iii) It should not be a Keyword.
iv) whitespace is not allowed.

Declaring a variable:- A variable should be defined before its used. The Syntax is

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

Eg:- int a, b, c;
float quot, div;

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

a) Initializing by Assignment statements.

b) Dynamic Initialization.

Initializing by assignment statements:

* one variable can be initialize using assignment statements.

The syntax is:

variable-name = value;

Eg:- int a=10, b, c=16;

double pi = 3.147;

Dynamic Initialization:- Java allows variables to be initialized dynamically, using expressions valed at the time variable is declared.

Eg:- 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 & lifetime of variables.

* Java allows variables to be declared within any block. A block is begin with an opening curly brace & 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 category

of scope:- Global & local. However these traditional scopes do not fit well with JAVA's strict object-oriented Model. As a general rule, variables defined & declared inside a scope are not visible (that is, accessible) to code that is defined outside the scope. Thus, when you declare a variable within a scope, you are localizing that variable & protecting it from unauthorized access &/or modification.

```
Class Scope
{
    public static void main (String args[])
{
    int x=10;
    if (x==10)
    {
        int y=20;
        System.out.println("x and y:"+x+" "+y);
        x=y*2;
    }
    y=100; // Error Y not known here
}}    System.out.println("x is "+x);
```

Note:- 1) There should not be two variables with the same name in different scope.

2) The variable at outer scope can be accessed in inner scope but vice versa is not possible.

Type Conversion & 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 called Type casting.

* Type casting can be done in two ways: → Widening (Implicit) (Automatic) → Narrowing (Explicit).

* Widening or automatic conversion takes place when:-
  i) The two types are compatible.
  ii) The distination type is larger than the source type.

Eg:- Int ← byte → short → int → long → float → double.
Widening casting (Implicit).

Note:- i) No automatic conversions from the numeric types to char or boolean is allowed. (char & bool are not compatible).

* Casting Incompatible types. (Narrowing conversion).
* When an int value has to be assigned to byte, then automatic conversion cannot be done i.e. byte is smaller than int. This is called Narrowing Conversion.
* the general form of explicit type conversion is:-

(target-type) value. → double → float → long → int → short → byte.

Eg:- int a;
byte b;
b = (byte) a; → is converted into byte from int.

Example program

Public class Test. {

```
public static void main (String [] args)
  {
     int i=100;
     long l = i;
     float f = l;
System.out.println("Int value"+i);
  System.out.println("Long value"+l);
  System.out.println("float value"+f);
```

O/P:- Int value 100
Long value 100
Floadt value 100.

Example for Narrow casting.

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

O/P:- Double value = 100.04
Long value = 100 Int value = 100.

Automatic Type promotion in Expressions.

* Type conversions also occurs in expressions.

Consider, byte a=40, b=50, c=100;

int d= a*b/c;

The result of the intermediate term a*b easily exceeds the range of either of its byte operands.

Thus, Java automatically promotes each byte, short, or Char operand to int when evaluating an expression.

ie. The sub-expression a*b is performed using integers & not bytes.

Thus result 50*40=2000 is valid.

Consider a case

byte b=50;

b=b*2; // Error! cannot assign an int to a byte!

The code is attempting to store 50*2, a perfectly valid byte value, back into a byte variable.

*However, because of the operands were automatically promoted 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. So,

byte b=50;

b=(byte)(b*2); & yields the value 100.

The type promotion Rules.

* Java defines several type promotion rules that apply to expressions.

They are as follows:-

1) First, all byte, short, & char values are presented to int.
2) If one operand is long, the whole expression is promoted to long.
3) If one operand is a float, the entire expression is promoted to float.
4) If any of the operands is double, the result is double.

Arrays

* An array is a group of like-typed variables that are referred to by a common name.
* A specific element in an array is accessed by its index.

One-dimensional Arrays.

* A one-dimensional array is essentially a list of like-typed variables.

The general form of 1D array is

type var-name [ ]; [No memory is allocated]

↑ data type of elements of array.

Eg:- int month_days[ ]; → Null

* "new" is a special operator that allocates memory.
* General form of new for 1D arrays are:-

array-var = new type[size];

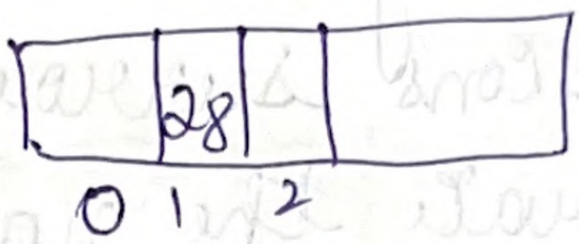
month_days = new int[12]; // Array of 12 integers

* Further all elements in the array will be initialized to zero.

Note:- In Java, all arrays are dynamically allocated.

* once an array has been allocated, a specific element in the array is accessed by specifying its index within square Brackets.
* All array indexes start at zero.

Eg:- month_days[1] = 28;



System.out.println(month_days[1]) will display "28" as o/p: Eg:- program for 1D array.

```
class Array {
public static void main (String args[]) {
    int month_days[];
    month_days = new int[12];
    month_days[0] = 31;
    month_days[1] = 28;
    month_days[2] = 31;
    month_days[3] = 30;
    month_days[4] = 31;
    month_days[5] = 30;
    month_days[6] = 31;
    month_days[7] = 31;
    month_days[8] = 30;
    month_days[9] = 31;
    month_days[10] = 30;
    month_days[11] = 31;
}
}
```

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

System.out.println("April has "+ month_days[3]
+ "days.");

}
}

O/P:- April has 30 days.

* Arrays can be initialized when they are declared.
* An array initializer is a list of comma-separated expressions surrounded by curly braces. The commas separate the values of the array elements.
* "It is not mandatory to use 'new' operator".

Eg:- int month_days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

Multidimensional arrays.

* In Java, multidimensional arrays are actually arrays of arrays. The general syntax for 2D array is:

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

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

0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4
3,0	3,1	3,2	3,3	3,4

Eg:- class twoDarray {
public static void main(String args[])
{ int twoD[][] = new int[4][5];

```
int i, j, k = 0;
    for (i=0; i<4; i++)
        for (j=0; j<5; j++){
            two D[i][j] = k;
            k++;
        }
    for (i=0; i<4; i++){
        for (j=0; j<5; j++)
            System.out.println (two D[i][j] + " ");
        }
    }
}
```

This program generates the following output:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

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.

Eg:-
```
String name1 = "Akash", name2;
name2 = name1;
System.out.println(name2) // String passed as parameter.
```

Applications of Java:-

1) Some of the applications of Java is that Internet users can use Java to create applet programs & run them using a web-browser.

2) There are two types of Java application. They are:-

1) Standalone Java application.
2) Web applets.

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

a) Compiling source code into byte code using javac compiler.
b) Executing the Bytecoded 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 & execute on local computer.

———— X ————