CHAPTER - 1 Marks = 12.

-: INTRODUCTION OF JAVA :-

JAVA HISTORY :-

Java is a general-purpose object-oriented Programming language developed by Sun Microsystems of USA in 1991, originally Java was called as "Oak" by James Gosling.

The goal of developing Java is to make language simple, portable and highly reliable.

The most important feature of Java is that - it is Platform independent language. Program developed in Java can be executed anywhere on the any system.

v.v.v. Imp

JAVA FEATURES :-

The following are the features of Java.

1. compiled and interpreted :-

Java Programs are compiled & Interpreted. First the Java compiler translates source code into bytecode instructions. In second stage, Java

"interPreter generates machine code that can be directly executed by the machine

2. Platform Independent and Portable:-
Java Program can be executed on any System and at any time.
"Write Once Run Anywhere"

3. Object-oriented:-
Java is true object oriented language. Almost everything in java is an object. All Program code & data reside within objects and classes.

4. Robust and Secure:-
Java is a robust language. It Provides many safeguards to ensure reliable code. Java Provides exception handling which captures series errors & eliminates risk of crashing the System.

Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet (small internet Program).

5) Simple, Small and familiar :-

Java is simple & small language. Many features of C & C++ are supported by Java, on other hand some of features are not supported by Java. Ex:- Java does not support pointers, goto statement, operator overloading, & multiple inheritance.

6) Multi-threaded & Interactive :-

Multi threaded means handling multiple tasks simultaneously. Eg:- we listen to an audio clip while scrolling a page or taking Print out etc.

7) High Performance :-

compared to C & C++, Java provides high performance.

etc.

8) Dynamic :-

Capability of dynamically linking a new class libraries, methods & Objects

How Java Differs from C and C++.

1)

* Java	* C
i) Java does not support keywords like "sizeof" and "typedef"	(i) C support keywords like "sizeof" and "typedef"
ii) Java does not contain data types like struct and union	(ii) C contain datatypes like struct and union
iii) Java does not contain keywords like auto, extern, register, signed & unsigned	(iii) C contain type modifiers keywords like auto, extern, register, signed & unsigned.
iv) Java does not support Pointers	(iv) C support Pointers.
v) Java does not have PreProcessor (Eg:- #include, #define etc)	(v) C support PreProcessor features
vi) Java requires that functions with no arguments must be declared with empty Parenthesis	(vi) C uses void keyword to indicate empty or no argument.

* continued

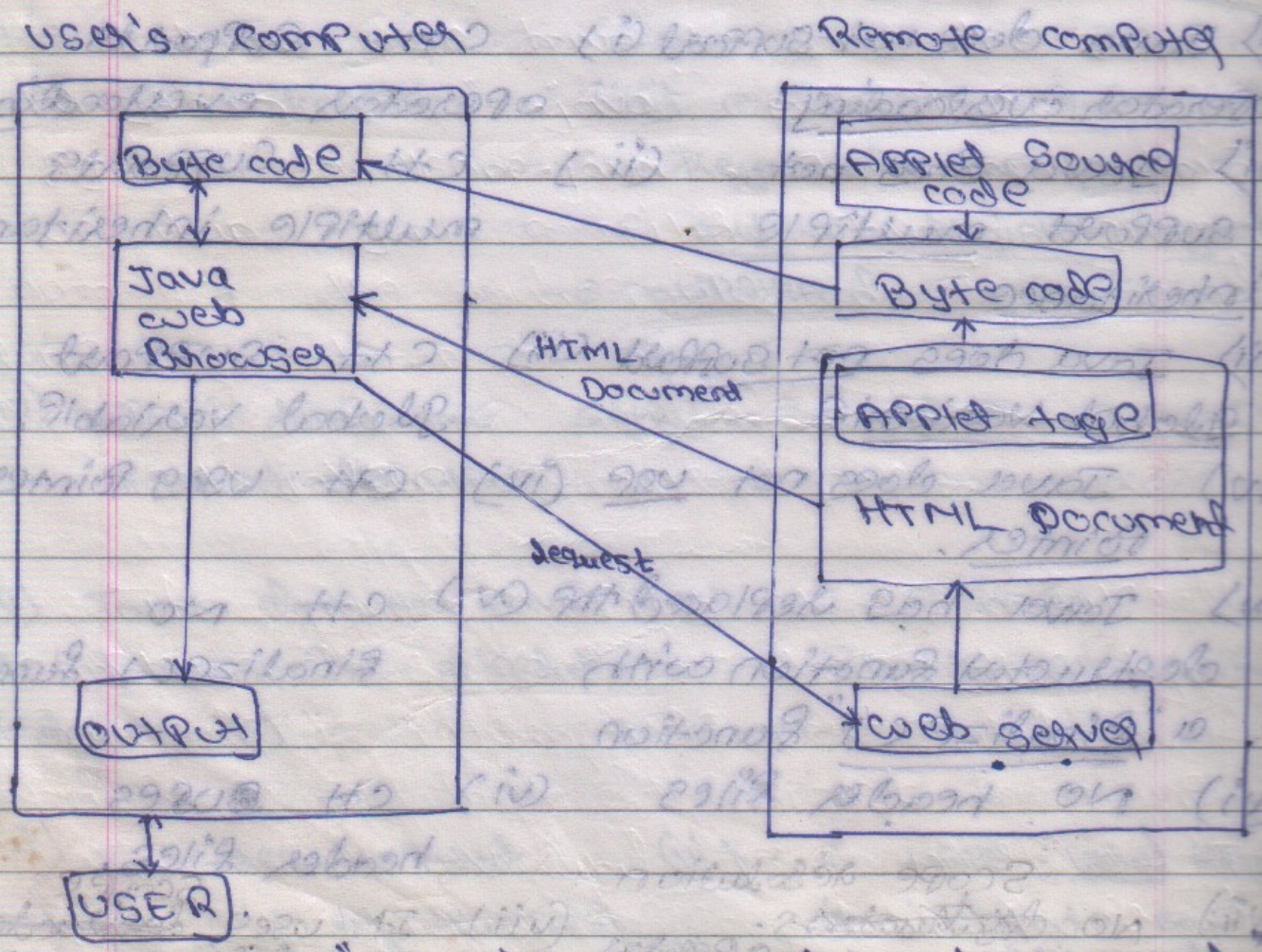
	Java	C++
(vii)	Java adds new operators such as instanceof and >>>	(vii) C++ does not support
II:	Java	C++
i)	Java does not support operator overloading	(i) C++ supports operator overloading
ii)	Java does not support multiple inheritance	(ii) C++ supports multiple inheritance
iii)	Java does not support global variable	(iii) C++ support global variable.
iv)	Java does not use pointer.	(iv) C++ uses pointers
v)	Java has replaced the destructor function with a "finalize()" function	(v) C++ no finalize() function
vi)	No header files	(vi) C++ uses header files.
vii)	No ~~destructors~~.	(vii) It uses scope resolution operator

Java and World Wide Web :-

Java is designed to be used on distributed environments such as Internet.

With the use of Java into web Pages made to display animation, graphics, games and wide range of special effects.

Java communicated with a web page through a special tag called <APPLET>. The following figure illustrates this process



"Java's Interaction with the web"

The figure shows the following communication steps:

1) The user sends a request for an HTML document to the remote computer's web server. The web server is a program that accepts a request, processes the request, & sends the required document.

2) The HTML document sent to user, containing the applet tag, which identifies the applet.

3) The corresponding applet bytecode is transferred to the user's computer.

4) The Java enabled browser on the user's computer interprets the bytecode & provides i/p

5) The user may further interact with applet but with no further downloading from the web server. This is because bytecode contains all the information necessary to interpret the applet.

* Simple Java Program :-

```
- class SampleOne {
    - Public static void main (String args[])
      {
        - System.out.println (" WELCOME TO JAVA ");
      }
}
```

- we have to save the Java Program ".java" extension.
Eg. "SampleOne.java". To compile this we run Java compiler, "Javac" giving name of the source file on the command line as shown below

```
c:\> javac SampleOne.java
```

The javac compiler creates a file called SampleOne.class that containing the byte code. Byte code is intermediate representation of a Program that contains m/c language for Java virtual m/c interpreter. To run this Program, we have to provide

class name as argument,

```
C:\> java SampleOne
WELCOME TO JAVA.
```

STEP BY STEP ANALYSIS OF PROGRAM:

Line 1:-

```
class SampleOne {
```

This line uses reserved word "class" to declare new class. "SampleOne" is a valid identifier that is used to identify the entire class. The entire class definition including all code & data, will be between the opening curly-brace "{" and the matching close curly-brace "}".

Line 2:-

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

① **Public** - "Public" is keyword, it is an **access specifier** which allows the

Programmer to control the visibility of each variable or method. In this case Public indicates that any class can see the main method.

(ii) Static :—

static is a keyword that allows method to be called without having to instantiate a particular instance of class. In this case of main, it is necessary that, to be declared static, since it is called by the interpreter before any instances are made.

(iii) void :-

If the method is not returning the value the keyword void is used.

(iv) main :—

main is method from where execution begins. The Java interpreter look for a method with this name when told to interpret a class.

NOTE :- The Javac compiler, compiles programs that do not have a main method. But the Java interpreter has no way to run class without main method.

(v) String args[] :- declares a parameter named args, which is an array of instances of class String

Line 4 :-

System.out.println("WELCOME TO JAVA")

class obj method

This line executes the println method in out, which is instance of outputStream that is statically initialized in the System class.

More of Java :-

A Java Program with multiple statements.

(P.T.O)

NOTE :- PACKAGE :→ is collection of classes.

```
/* This code computes
* the square root of a
* given number
*/
import java.lang.Math;
class SquareRoot
{
  Public Static void main (String args[])
   {
     double x=5; //Declaration & initialization
     double y; // simple declaration
     y= Math.sqrt(x);
     System.out.println("y=" +y);
   }
}
```

o/p → c:\> javac SquareRoot.java
→ c:\> java SquareRoot
y = 2.23607.

The first statement of the program is
import java.lang.Math;
the purpose of this statement is to instruct the interpreter to load the "Math class"

the Package "lang". This statement is similar to #include statement in C. Math class contains the "sqrt" method required in the program.

An Application with two classes:-

A program with multiple classes.

```
class Room.
{
    float length;
    float breadth;
    void getdata (float a, float b)
    {
        length = a;
        breadth = b;
    }
}
class RoomArea
{
    public static void main (String args[]) {
        float area;
→       Room r1 = new Room();
        r1.getdata (14, 10);
```

```
        area = r1.length * r1.breadth;
        System.out.println("Area =" +area);
    }
}
```

The Program contains two classes "Room" & "RoomArea". The 'Room' class defines two variables and one method to assign values to these variables. The class RoomArea contains the main method that initiates the execution.

The main method declares a local variable area and Room type object 'r1' & then assigns values to the data members of Room class by using the getdata method Finally it calculates the area & prints the results. The dot (.) operator is used to access the variables and methods of Room class.

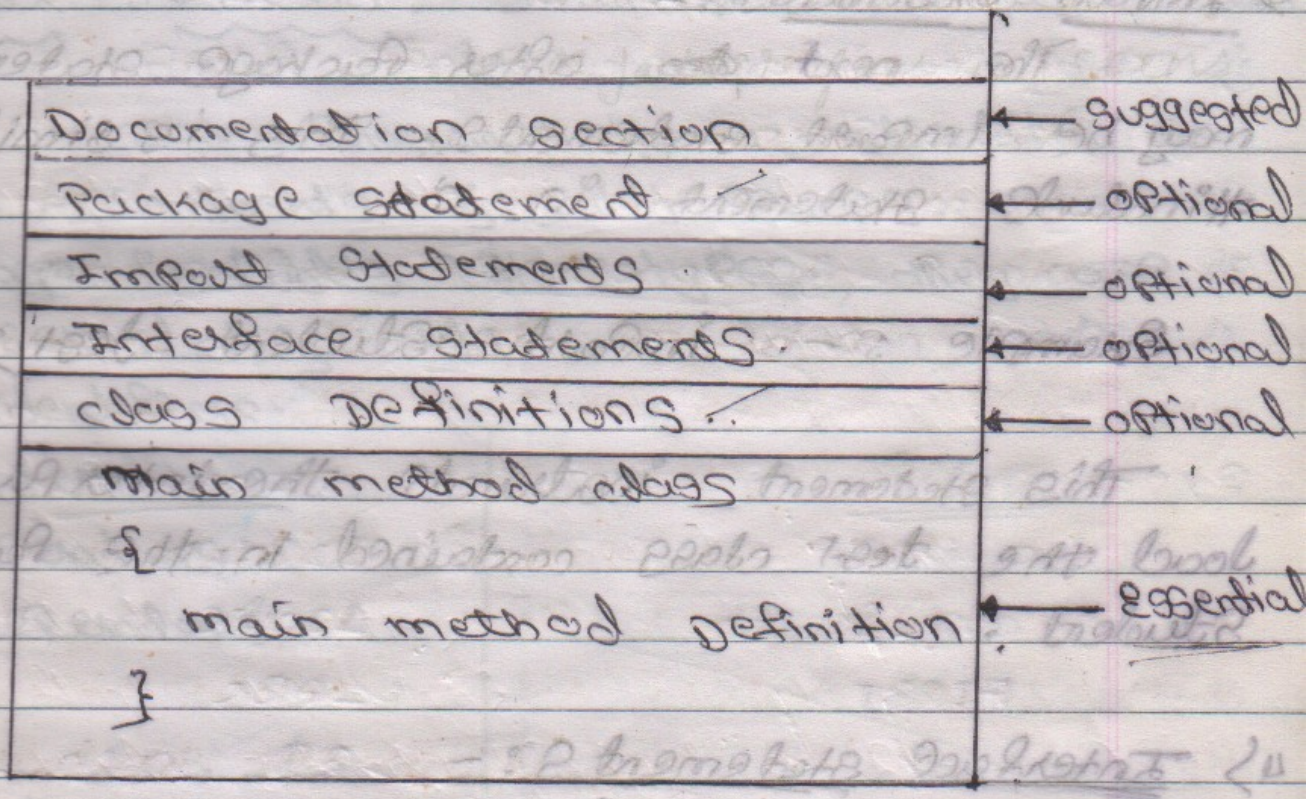
```
Room r1 = new Room();
```

The above statement is used to create the object. "new" is keyword used to create objects of class.

Here "r1" is a object of Room

Java Program Structure :-

The General Structure of java Program is as shown below.



Documentation Section :-

It contains set of comment lines giving the name of the Program & other details.

Package statement :-

The first statement allowed in a java file is a Package statement. This statement declares a package name & informing the compiler that the classes defined here belong

to this example.

Package Studen;

3) Import Statements :-

The next after Package statement may be import statements. This is similar to #include statement in C

Example :- import Studen.Test;

This statement instructs the interpreter to load the Test class contained in the package Studen.

4) Interface Statements :-

It is used only when we wish to implement the multiple inheritance feature

5) class Definitions :-

The java Program may contain single or multiple classes.

6) main method class :-

All Programs execution begins from main method. So this class is essential

of a Java Program.

Java Tokens:- Smallest individual units in a java program is Tokens

Java Program is a collection of tokens, comments & white spaces. Java language includes five types, they are

1) Reserved keywords
2) Identifiers
3) Literals
4) operators
5) seperators

Reserved keywords:-

keywords are special identifiers that the java language has reserved to control how program is defined. Java language has 50 words as keywords. These keywords are combined with operators & seperators according to a syntax form definition of the Java language.

Ex:- boolean, try, catch, throw, this, final, finally, super, ...etc.

PAGE NO.:
DATE: / /

Identifiers:-

Identifiers are used for naming classes, methods, variables, objects, interfaces, Packages in a Program. The following are rules for naming Java Identifiers

1) They can have alphabets, digits, & underscore & dollar sign
2) They must not begin with a digit
3) uppercase & lowercase letters are distinct
4) They can be of any length.

Ex:- Valid	Invalid
average	1234
avg_length	12abc
FIRST	-ABC
	3$, a:b , @2.

Literals:-

A constant value in java is created by using a literal representation of it.

Java has five types of literals. They are:

1) Integer literals (ex→ int)
2) Floating Point literals (ex→ float, double)
3) character literals (ex→ char)
4) String literals (ex→ String)
5) Boolean literals (ex→ boolean)

Operators :-

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

Ex :- Arithmetic, logical, relational

Seperators :-

Seperators are special symbols that can appear in Program.

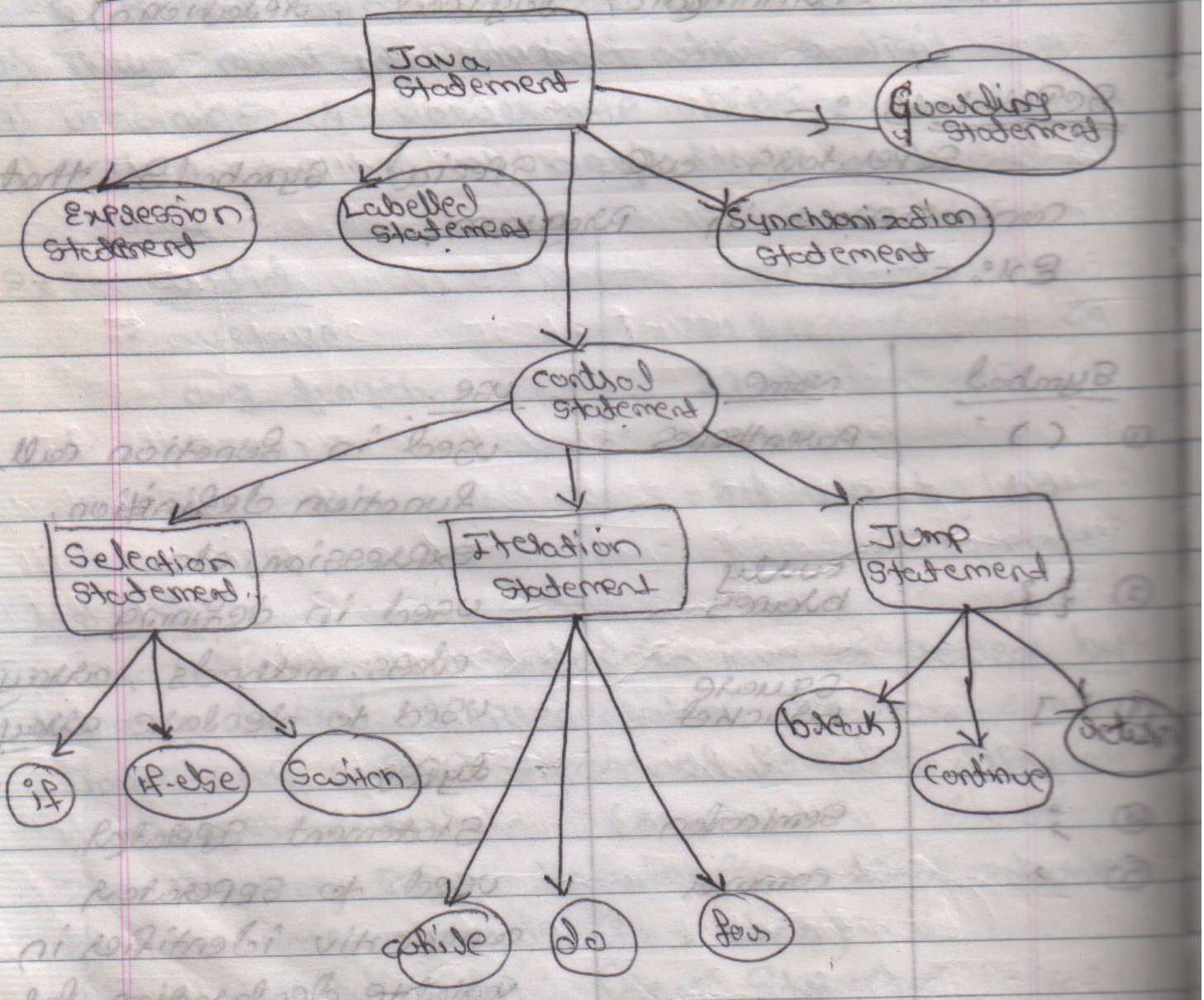
Ex :-

Symbol	Name	use
① ( )	Parentheses	used in function call, function definition, expression etc.
② { }	curly braces	used in defining class, methods, arrays
③ [ ]	square bracket	used to declare array types.
④ ;	Semicolon	Statement seperator
⑤ ,	comma	used to seperate consecutiv identifier in variable declaration, for statements, etc.
⑥ .	Period	used to seperate Package names from sub-Packages & classes.

Java Statements :—

Statement is an executable combination of tokens ending with a semicolon (;) mark.

Classification of Java statements :-



PAGE NO.:
DATE : / /

Implementing JAVA Program:-

Implementation of a java program involves following steps,

1) creating the Program
2) compiling the Program
3) Running the Program.

Before we run java Program, the JDK (Java Development Kit) Software must be installed.

Creating the Program:-

we can create Program in an editor.

ex:-

```
class Test
{
    Public static void main (String args[])
    {
        System.out.Println ("Hello!");
        System.out.Println("welcom to Java");
    }
}
```

we must save this Program with file name as : Test.java.

Note that all Java source files will have the extension .java. If the Program contains multiple classes, the file name must be the classname of the class containing the main method.

Compiling the Program :-

To compile the Program, we must run the Java compiler Javac, as

→ `javac Test.java`

If there are no errors, then javac compiler creates file called "Test.class" containing bytecode. The compiler automatically generates bytecode file as

`<classname>.class`

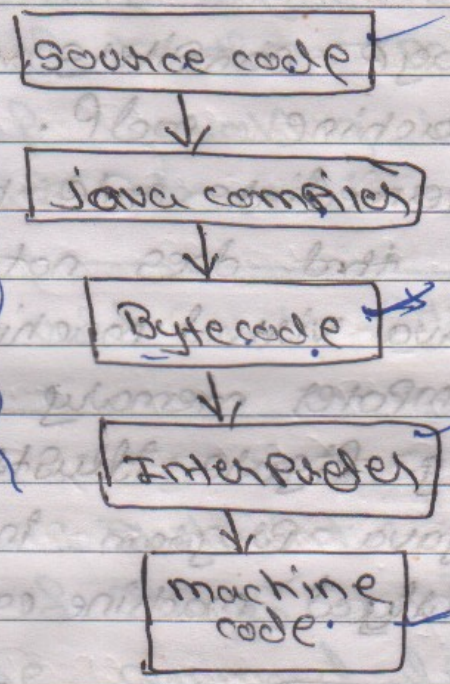
Eg:- Test.class.

(Continued)

javac :

PAGE NO.:
DATE: / /

Implementation of Java Program



Running the Program :—

We need java interpreter to run a program. At the command prompt, type

```
java Test
```

Now, the interpreter searches for the main method in the program & begins the execution. O/P will be displayed as.

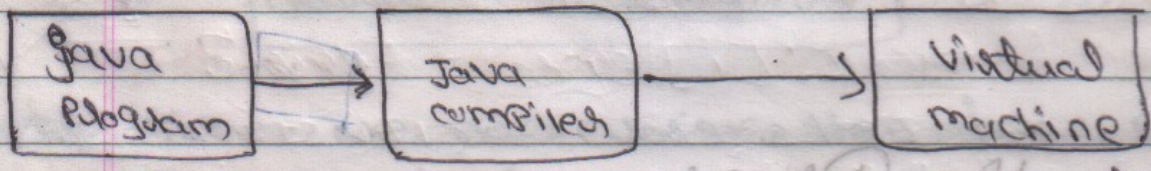
```
Hello
Welcome to Java
```

Imp.

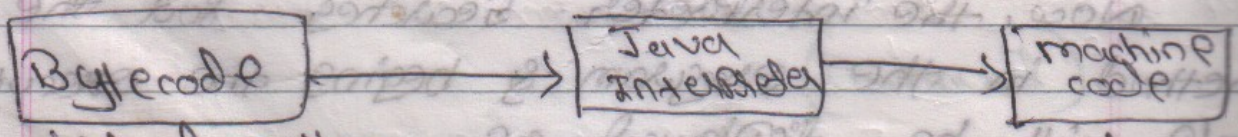
Java Virtual Machine :-

All language compilers translate source code into machine code. In java, compiler produces an intermediate code known as bytecode for a machine that does not exist. This machine is called the Java virtual machine and it exists only inside the computer memory.

The following figure illustrates the process of compiling a java program into bytecode which is referred as virtual machine code.



The virtual machine code is not machine specific. The machine specific code is generated by Java interpreter that act as intermediate between virtual machine and the real machine.



Command Line Arguments :-

command line arguments are Parameters that are supplied to the application Program at the time of invoking it for execution.
The Signature of main() method is.

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

The above statement "args" is declared as an array of Strings. Any arguments Provided in the command line are passed to the array "args" as its elements.

Eg:- Java Test BASIC C++ JAVA

The command line containg three arguments are assigned to the array args as follows;

BASIC ⇒ args[0]
C++ ⇒ args[1]
JAVA ⇒ args[2]

Ex: Program to illustrate use of command line arguments.

```
class commandTest
{
Public static void main(String args[])
{
int count, i=0;
String string;
count = args.length;
System.out.Println("Number of arguments="+count);
while(i<count)
{
String = args[i];
i = i+1;
System.out.Println(i+":"+"Java is"+string+");
}
}
}
```

```
javac commandTest.java
java commandTest simple Secure Portable.
```

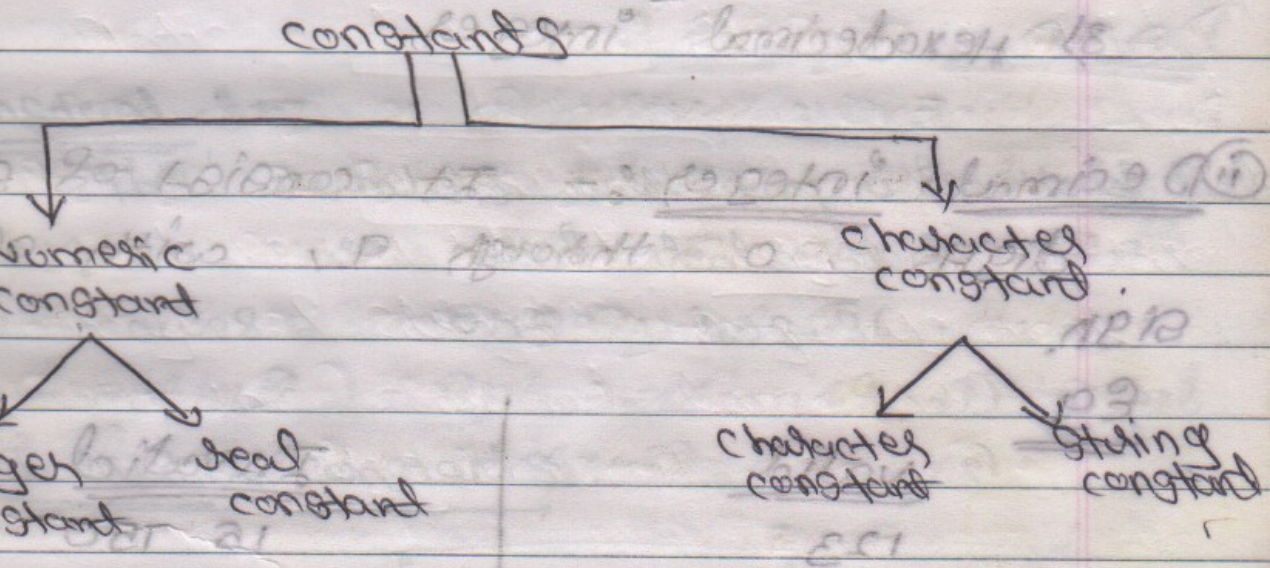
o/p

```
Number of arguments = 3
1: Java is simple
2: Java is Secure
3: Java is Portable.
```

Constants

constant is defined as a fixed value that do not change during the execution of a program.

classification of Java constants.



Integer constant :-

Integer constant is sequence of digits with optional minus sign

Ex:- 123, -321, 65432

(P.T.O)

Integer constants :-

An integer constant refers to a sequence of digits. There are 3 types of integer constants

1) Decimal integer
2) octal integer
3) Hexadecimal integer.

(1) **Decimal integer** :- It consist of set of digits, 0 through 9, with optional minus sign.

Eg

valid	Invalid
123	15 750
234	12.00
-4	$100
0	*111
23452	@231

(2) **octal integer** :- It consist of set of digits from '0' to '7'

Eg : 0, 037, 0451, 0551

(3) **Hexadecimal Integer** :- It consist of set of 16 digits, from '0' to '9' & '10 to 15' represented as 'A' to 'F'.

Hexa decimal numbers are preceded by "0X"

Eg:- 0X2
0X9F
0x bcd
0x1

Real constant :-

The numbers containing a fractional parts like 123.45 are called real constants.

Eg:- 0.008 -0.75 453.36

Single character constants :-

character constant contains a single character enclosed within a pair of single quote mark

Eg:- '5' 'x' 'A' ' '

String constants :-

String constant contains a sequence of characters enclosed between double quotes. The characters may be alphabets, digits, special characters & blank space.

Eg:- "Java" "1997" "+-/" "A" etc.

75-80

PAGE NO.:
DATE: / /

VARIABLES

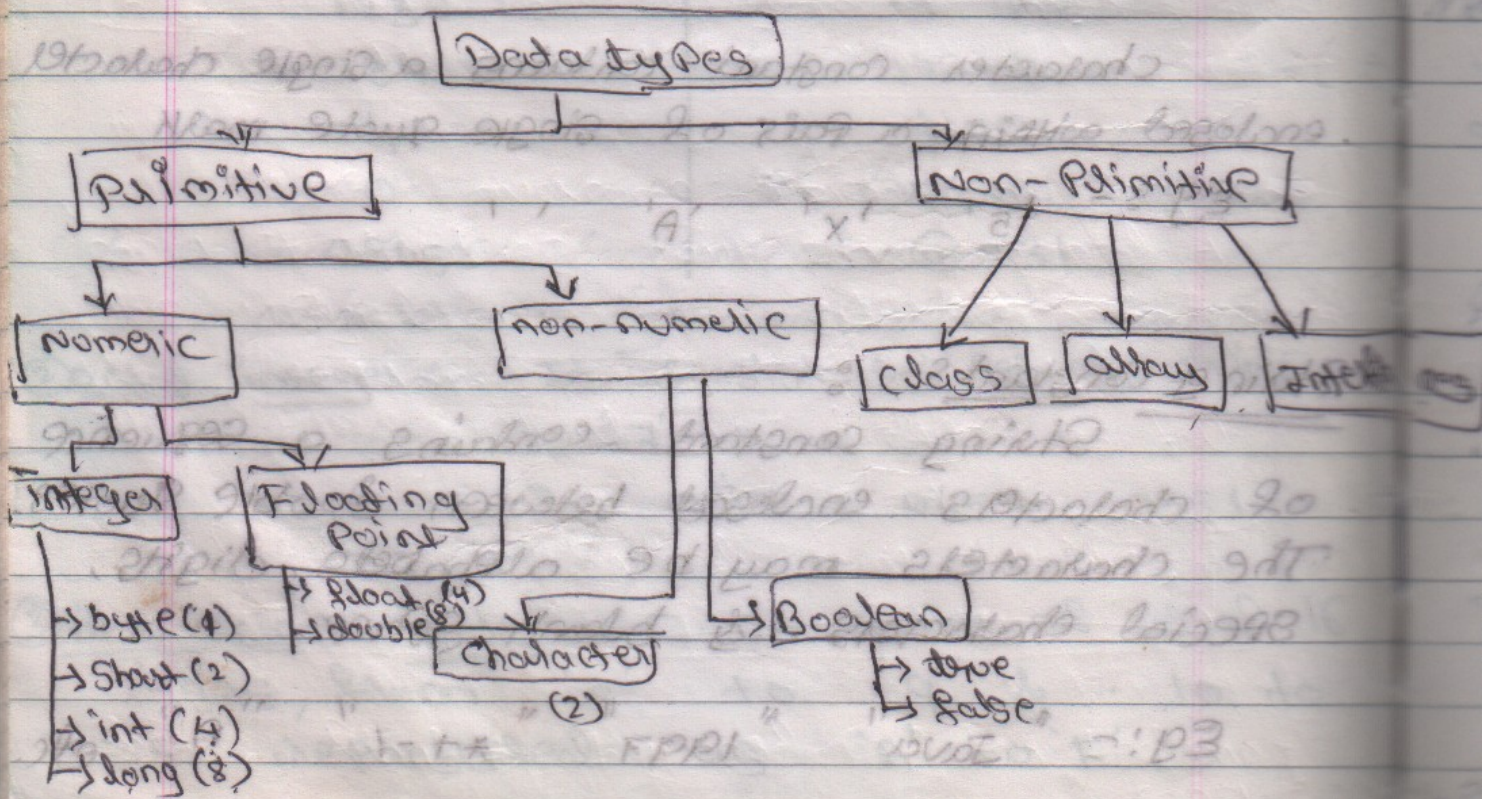
variable is a value that may take different value during execution of the Program.

eg: average, height, class strength.

DATA TYPES :-

Every variable in Java has data type. Data type specify the size & type of values that can be stored.

Datatypes can be classified into 2 types ① Primitive types (Builtin types)
② Non Primtive (Derived types)



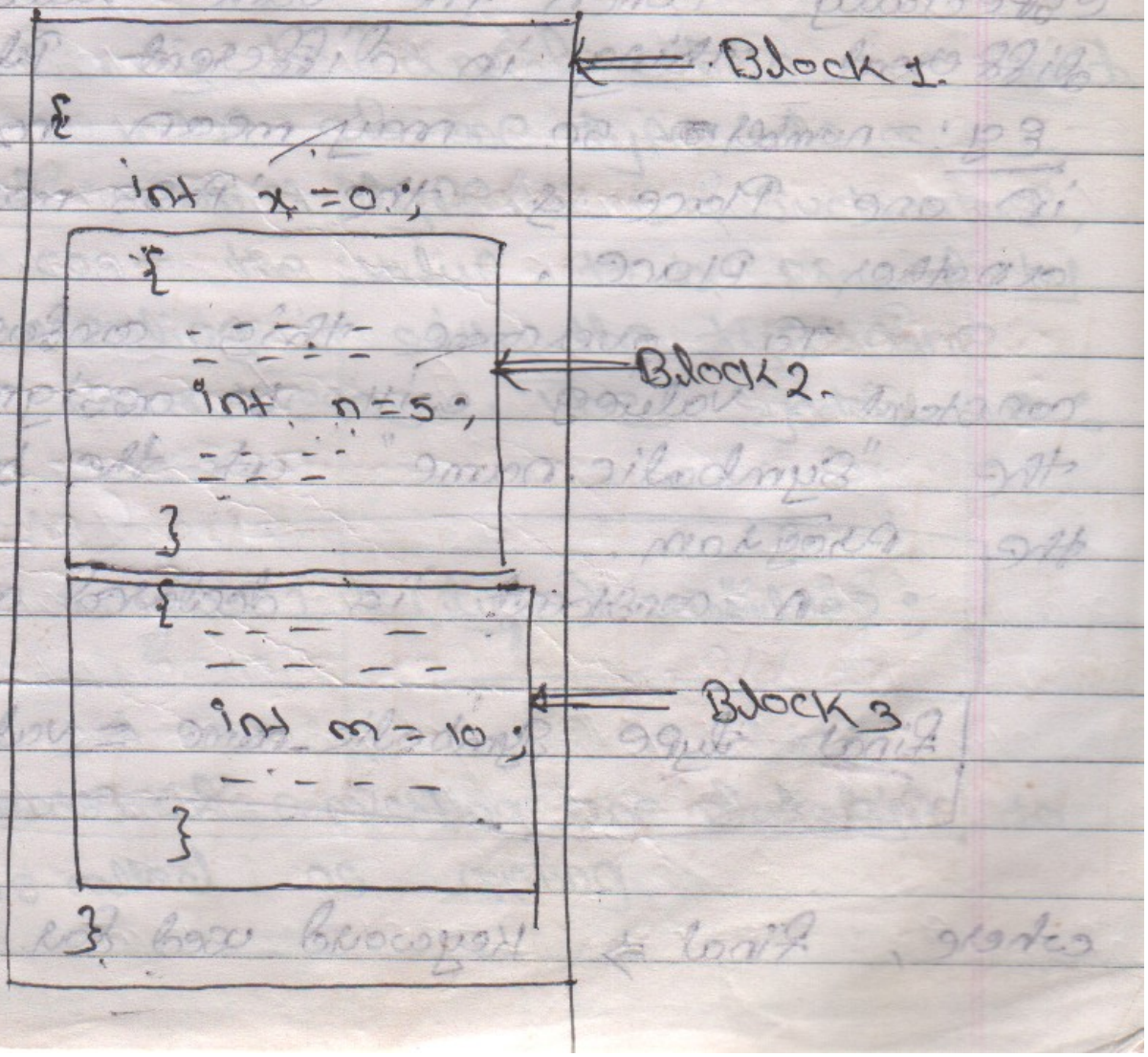
PAGE NO.:
DATE: / /

Scope of variables :-

The compound statement blocks in java are demarked with two curly braces {}. Java variables are valid only from the point where they are declared till the end of ~~block~~ enclosing compound statement.

These compound statement can be nested, & each containing its own set of local variable declaration.

The area of the program where the variable is accessible is called scope



The variable 'x' declared in Block 1 is accessible in all the 3 blocks, where as the variable 'n' declared in Block 2 is available only in Block 2, & the variable 'm' declared in Block 3 is available in Block 3 only.

Symbolic constants :-

When a numeric value appears in a Program, its use is not always clear, especially when the same value means different thing in different places.

Eg :- number 50 may mean no. of students in one Place & the 'Pass marks' at another Place.

To overcome this confusing the constant values will be assigned to the "Symbolic name" at the beginning of the Program.

A constant is declared as follows:

```
final type Symbolic_name = value;
```

where, final ⇒ keyword used for declaring constants

type => Data type

Symbolic name => is a variable name

eg => `final int STRENGTH = 100;`

NOTE :-

① After declaration of Symbolic constant they should not assigned to any other value with in the Program.

Type casting :-

There may be a need where we want to store a value of one type into a variable of another type. In such case, we must cast the value to be stored by preceeding it with the type name in the Parentheses. The syntax is :

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

The process of converting one datatype to another is called as casting.

int = 4
byte = 1
long = 8

Ex:-

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

The Process of converting Smaller type to larger type is known as widening

The Process of converting larger type to Smaller type is known as narrowing

Standard Default values:-

Types of variable	Default value
Byte	zero
short	zero
int	zero
long	zero (OL)
float	0.0f
double	0.0d
char	null character
boolean	false

Jmp

Special operators :-

They are two types of special operators
1) instanceof operator
2) Dot operator.

Instanceof operator :-

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

Example :- obj class

Person instanceof student

is true if the object Person belongs to the class student, otherwise it is false.

Dot operator :-

The dot (.) is used to access the instance variables & methods of class objects.

Ex :-

Person1.age // reference to the variable age

Person1.Salary() // reference to method Salary()

Mathematical Functions:-

The basic mathematical functions such as sqrt, Pow, Sin, cos, tan etc are used is some Programs. Java support these basic math functions through "Math" class defined in the java.lang Package.

Syntax:-

```
Math.function_name();
```

Ex:- `double y = Math.sqrt(x);`

Eg: Prog to illustrate use of math function

```
import java.lang.*;
class mathfun
{
  Public Static void main (String args[])
  {
    double x;
    x = Math.max(22, 2);
    System.out.println("The max number is" + x);
    x = Math.sqrt(4);
    System.out.println("The square root is" + x);
    x = Math.log(4);
```

23 31/21

PAGE NO.:
DATE: / /

```
        System.out.println("The log of 4 is"+x);
    }
}
```

Labeled Loops (Break & continue)

break:

The term break refers to the act of breaking out of a block of code. It tells the runtime to pick up execution just past the end of the named block.

Labeled

Prog. to illustrate break statement:—

```
class Break {
    Public static void main (String args[]) {
        boolean t = true;
a:      {
b:          {
c:              {  System.out.println("Before the break");
                   if(t)
                       break b;
```

```
        System.out.println("This wont Execute");
    }
    System.out.println("This wont execute");
}
System.out.println("This is after b");
}
}
```

o/p :-

Before the break

This is after b.

continue :-

Operators & Expressions:-

The java operators can be classified as:

1) Arithmetic operators
2) Relational operators
3) Logical operators
4) Assignment operators
5) Increment & Decrement operators
6) Conditional operators
7) Bitwise operators
8)

1) Arithmetic operators:-

Arithmetic operators are used to construct mathematical expressions as in algebra.

Operator	Meaning
+	Addition / unary Plus
-	Subtraction / unary minus
*	multiplication
/	Division
%	modulo (Remainder)

Arithmetic operators can be used as

a+b	a%b
a-b	-a+b
a*b	-a*b ... etc.
a/b	

2) Relational operators :-

The comparision can be done with the help of relational operators.

operators	meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	is equal to
!=	not equal to

Eg :-

Expression	meaning
4.5 <= 10	True
4.5 < -10	False
-35 >= 0	False
10 < 7+5	True

3) Logical operators :-

Logical operator is one which is used to combine two or more relational expressi

operators	meaning
&&	logical AND
||	logical OR
!	logical NOT

Ex:- ① if (age > 55 && Salary < 1000)

② if (num < 0 || num > 100)

④ Assignment operators :-

Assignment operator are used to assign the value of an expression to a variable.

Syntax :-

~~var = expression;~~

var op = expression ;

where var => is variable

op => is binary operator

expression => _ _ _

= => is 'assignment operator'

Ex :-

① a = a+1

② b = a % 10

③ c = c * a

④ b = a / (n+1)

⑤ a += 1 is equivalent to a = a+1

⑥ b -= 1 " " " " b = b-1

⑦ a = a / (n+1) " " " " a /= n+1

etc :-

PAGE NO.:
DATE: / /

⑤ Increment & Decrement operators :-

The increment & Decrement operators are "++" and "--" respectively.

The operator ++ adds 1 to the operand while -- subtract 1.

⑥ conditional operator :-

The character pair ?: is a ternary operator. It is used to form the conditional expression as

exp1 ? exp2 : exp3.

the ternary works as follows : exp1 is evaluated first. If it is nonzero, then the expression exp2 is evaluated. ~~& becomes the value of~~ If the exp1 is false, then exp3 is evaluated.

Ex :- a = 10;
b = 15;
x = (a > b) ? a : b;

conditional operator is equivalent to if --- else --- statement.

(8) Bitwise operators :-

Bitwise operators are used to manipulate data at the values of bit level.

operators	meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
~	one's complement
<<	Shift left.
>>	Shift Right.
>>>	shift Right with zero fill

Ex :-

x = 1101 → (13, Decimal)

y = 1001 → (9, ")

(1) x & y will be 9

(2) x | y will be 13

(3) x ^ y will be 6

(4) ~x will 2

Eg :-

(*)
```
 1101
&1001
------
 1001
```
↓ 9

(*)
```
 1101
|1001
------
 1101
```
↓ 13

(*)
```
 1101
^1001
------
 0110
```
↓ 6

Decision Making, Branching & looping:

The following are decision making statements

1) if Statement.
2) if-else Statement.
3) Switch Statement,
4) conditional operator statement.

if - Statement :-

It is two-way decision statement & is used with expression. It takes the following form :-

Syntax:

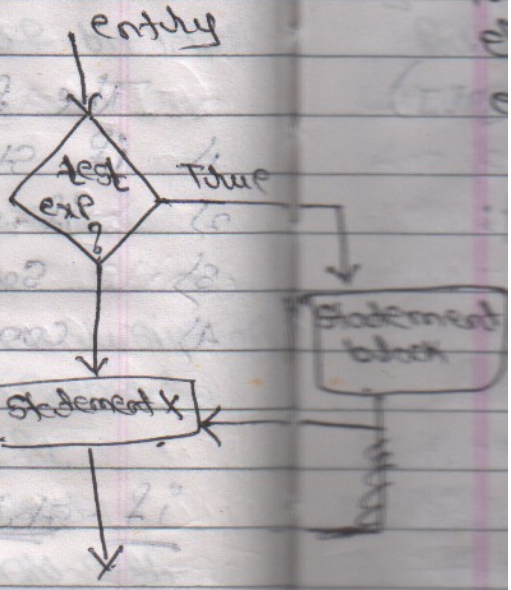
```
if (test_expression)
```

The if statement can be implemented in different forms

i) Simple if statement
ii) if-else statement
iii) nested if.... else statement
iv) else if ladder

The general form of simple "if" statement is

```
if (test_expression)
{
    ---------
    Statement block;
    - - - - -
    - - - -
}
- - - - - -
- - - - - -
Statement X;
```



If the "test_expression" is true "statement block" will be executed. else otherwise the execution will jump to Statement X.

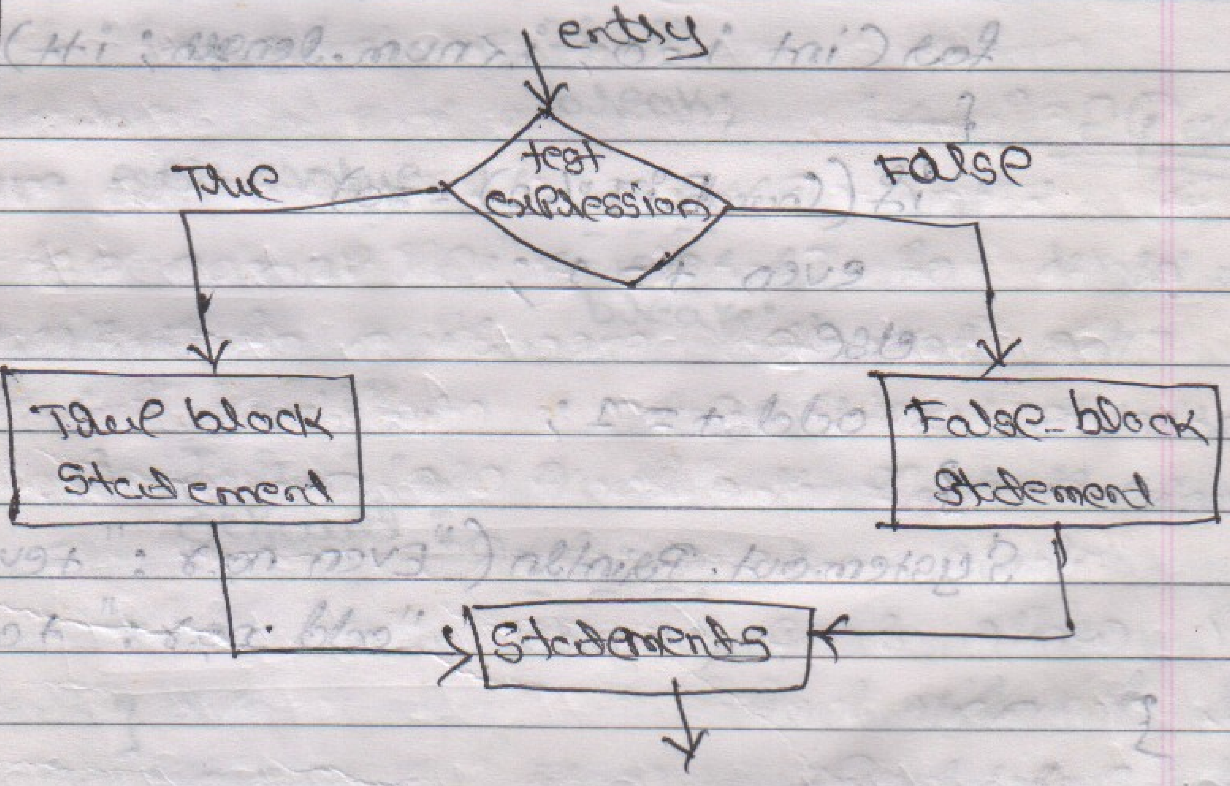
if ... else statement :-

Syntax :-

```
if (test_exp)
{
    True block statement;
}
else
{
    False block statement;
}
```

if the "test_exp" is true then True-block statement will be executed else False-block statement will be executed.

flow chart



e switch statement,

Prog to illustrate if-else statement :-

```
class Test
{
    Public Static void main (String args[])
    {
     int num[] = { 50, 65, 56, 71, 81};
     int  even=0, odd=0;
     for(int i=0; i<num.length; i++)
     {
      if((num[i]%2) ==0)
        even += 1;
      else
        odd += 1;
     }
    System.out.Println("Even no.s :" +even+
                       "odd no.s :" +odd);
}
```

The Switch Statement :-

when multiway decision has to be done then switch statement is used. Switch statement tests the value of a given variable against list of case values and when a match is found, a block of statements

associated with that case is executed. The general form of the switch statement is as follows;

```
switch (expression)
{
  case value_1 :
              ----
              break;
  case value_2 : ----
              ----
              break;
  ----------
  default :  ----
              break;
}
```

The expression is an integer expression or characters. value_1, value_2 .... are known as case Labels.

When switch statement is executed, the value of expression is compared against the case Labels. If the case is found whose value matches with the value of the expression, the block of statements that follows the case are executed.

The break Statement at the end of the block signals the end of a particular case & causes an exit from the switch statement.

The default block will be executed only if the value of expression does not match with any of case Labels.

Loops :—

The Process of repeatedly executing a block of Statement is known as Looping.

The repeated execution of statement continues until some conditions for the termination of the loop are satisfied.

There are 3 loops

1) while loop.
2) do-while loop.
3) For loop.

while Loop :—

The while is an entry controlled loop Statement. The test condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test condition is once again

evaluated and if it is true, the body is executed once again. This process continues until the condition becomes false.

The syntax of while loop is :-

```
while (test_condition)
{
    Body of loop
    - - - - -
    - - - - -
}
```

2) do-while loop :-

The do-while loop is exist controlled loop statement. without checking any test condition. the program proceed to evaluate the body of the loop. At the end of the loop, the test condition in the while statement is executed. If the condition is true program continues to evaluate the body of the loop. The syntax of do-while loop is :-

```
do
{
    - - - - -
    - - - - -
    - - - - -
} while (test_condition);
```

3) For loop :-

The execution of for loop begins as follows :

i) Initialization of the control variables is done first

ii) Then the value of control variable is tested using test condition. If the condition is true, the body of loop is executed; else the loop is terminated.

iii) when the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop. Now the control variable is increment. & new control variable is again tested. If the condition is satisfied, the body of the loop again executed. This continues until the control variable fails to satisfy the test condition.

The Syntax of for loop is :

```
for(initialization; test condition; increment)
{
  Body of loop
}
```

MQP.

Program to find the number of & sum of all integers greater than 100 & less than 200 that are divisible by 7

```
class Division
{
  Public static void main (String args[])
  {
    int  i=0, Sum=0, n=0;
    for (i= 100 , i<200 ; i++)
    {
      if (i%7 == 0)
      {
        n++;
        Sum = Sum +i;
      }
    }
  System.out.Println("Sum of integers greater than
                      100 & less than 200 that are
                      divisible by 7 is:\n" +Sum);

System.out.Println("Total number of integers that
                    are divisible by 7
                    (between 100 to 200) are:\n"
                    + n);
}
}
```

PAGE NO.:
DATE: / /

O/P:-

Sum of all integers greater than 100 & less than 200 that are divisible by 7 is: 196

Total number of integers that are divisible by 7 (between 100 to 200) are: 14.

Chapter - 2. Marks = 25

Classes, Objects and Methods.

PAGE NO.:
DATE: / /

Basic Defination :-

class :-

classes are user-defined data type and behave like the built in types of Programming language.

object :-

object can be defined as real world entity. They may represent a person, Place, bank A/c, ... etc.

A class may be thought as datatype & object may be thought as variable of that datatype.

Defining a class :-

class is a user-defined datatype that defines the data fields & methods of the object. we can create variable of class type using declaration that are similar to the basic type declaration. In java these variables are termed as instances of classes

The basic form of class definition is

```
class classname [extends superclassname]
{
    [fields declaration;]
    [methods declaration;]
}
```

⇒ Everything inside the square bracket is optional.

⇒ "classname" & "superclassname" are any valid identifiers.

⇒ The key word extends indicates that the properties of the "superclassname" are extended to "classname" class. This concept is known as Inheritance.

Fields Declaration :-

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated.

Syntax :- (Example)

```
class Rectangle
{
    int length;
    int width;
}
```

→ Instance variables / members variables.

Method Declaration :-

Class contains two things : instance variables & methods.

The general form of method is

```
returntype name (Parameter_list)
{
    ------ // Body of method.
    ------
}
```

Method declaration have 4 basic parts :

1) name of method
2) return type of method (int, void, float, etc)
3) A list of parameters
4) The body of the method

A Prog to compute area of rectangle.

```
class Rectangle
{
 int length, width ;
  void getData (int x , int y)
   {
      length = x ;
      width = y ;
   }
  int rectarea ()
   {
      int area = length * width
      return (area) ;
   }
}

class cal
 {  Public static void main (String args[]) {
      Rectangle R1 = new Rectangle ();  // creati
       R1 . getData ( 14 , 20);  // Accessing me
       int area = R1. rectarea ( ) ;
    System.out.Println (" Area:" + area);
      }
}
```

Creating objects :-

Objects in java are created using "new" operator. The new operator creates an object of the specified class & returns a reference to that object.

Eg :- Rectangle rect1; // declare the object
rect1 = new Rectangle(); // instantiate the object.

The first statement declares a variable to hold the object reference & the second one actually assigns the object reference to the variable.

Both statements can be combined into one as shown below

```
Rectangle rect1 = new Rectangle();
```

Accessing class Members :-

once class is declared it is having its own set of instance variables & instance methods. (data members & member functions). we cannot access directly these.

To do this, we must use the concerned object & the dot operator as shown below:

```
objectname.variablename = value;
objectname.methodname(Parameter-list);
```

"objectname" is the name of the object, "variable name" is the name of instance variable inside the class that we wish to access.

"Methodname" is the method that we wish to call. & "Parameter-list" is a comma separated list of "actual values".

Ex:-

1. rect1.getdata(15,10);
2. rect1.length = 15;
3. rect1.width = 10; etc.

V.V.V
MP

MCSP

PAGE NO.:
DATE: / /

Constructors :-

Constructor is a special type of method that enables an object to initialize itself when it is created".

Special Properties of constructors.

1) constructors have the same name as the class name.
2) They do not any return type, not even as void.
3) A constructor cannot be static, final, Public.
4) constructors cannot be inherited.

Types of constructors :-

1) Default constructors
2) Parameterized constructors
3) Copy constructors

Default constructor :- (constructors with no arguments)

Default constructor automatically initialize the object variables with some default values.

Ex:- Prog to illustrate the use of default parameterized constructors.

```
class Perimeter
{
    int length;
    int breadth;
  Perimeter()
    { length=0;
      breadth=0;
    }
    void disP()
    {
      System.out.println("length =" + length +
                         "breadth =" + breadth);
    }
}
class defaultdemo
 {
    Public Static void main (String args[])
    {
      Perimeter P1 = new Perimeter();
      P1.disP();
    }
}
```

(2) Parameterized constructors:-

Parameterized constructors is one, which take Parameter values as argument.

Ex:- Prog to illustrate the use of Parameterized constructor.

```
class Perimeter
{
  int length;
  int breadth;
  Perimeter (int x, int y)
  {
    length = x;
    breadth = y;
  }
  void disP()
  {
    System.out.Println ("length=" + length +
                        "breadth=" + breadth);
  }
}
class Parameterdemo
{
  Public static void main (String args[])
  {
    Parameterdemo P1 = new Parameter (10,14);
    P1.disP();
  }
}
```

③ Copy constructor:-

Copy constructor is one, which accepts argument as a object itself.

Ex: Prog to illustrate the use copy constructor

```
class copy
 {
    int a;
    int b;
    copy(int x, int y)
    {
       a=x;
       b=y;
    }
    copy (copy c)
    {
       a = c.a;
       b = c.b;
    }
    void disp()
    {
       System.out.println("a="+a+"b="+b);
    }
 }
```

```
class copydemo
{
    Public static void main (String args[])
    {
        copy c = new copy (10, 14);
        copy c1 = new copy (c);
        c.disP();
        c1.disP();
    }
}
```

Method overloading :-

It is possible to create methods that have the same name, but different parameter lists & different definitions. This is called method overloading. When we call a method in an object, java matches up the method name first & then the number & type of parameters to decide which one of the definitions to execute. This Process is known as Polymorphism.

To create an overloaded method, all we have to do is to provide several different method definitions in the class, all with the same name, but with different parameter

lists. The difference may either be in the number or type of arguments.

Ex:- Prog to illustrate the concept of method overloading.

```
class overload
{
    int length;
    int breadth;
    overload ( )
    { length = 0;
      breadth = 0;
    }
    overload (int x, int y)
    { length = x;
      breadth = y;
    }
    overload (overload o)
    { length = o.length;
      breadth = o.breadth;
    }
    void disp()
    {
        System.out.println("length = " + length + "breadth = " + breadth);
    }
```

```
}
class overloaddemo
{
  Public Static void main (String args[])
  {
    overload o1 = new overload ();
    overload "o2" = new overload (10, 14);
    overload o3 = new overload (o2);
    o1.disP();
    o2.disP();
    o3.disP();
  }
}
```

Static members :-

Assume that we want to define a member that is common to all the objects & accessed without using Particular object. That is the member belongs to the class as a whole rather than the objects created from the class. Such members can be defined as follows :

```
static int count;
Static int max (int x, int y);
```

The static variables & static methods are referred to as class variables & class method.

Jap

NOTE :- The static methods are called using class names.

Static methods have several restrictions:

1) They can only call other static methods
2) They can only access static data
3) They cannot refer to "this" or "super" in any way.

Program to illustrate the use of static members :-

```
class MathOperation
{
  static float mul (float x, float y)
  {
    return (x*y);
  }
  static float divide (float x, float y)
  {
    return (x/y);
  }
}

class MathApplication
{
```

```
Public void static main (String args[])
{
    float a = MathOperation.mul(4.0, 5.0);
    float b = MathOperation.divide(a, 2.0);
    System.out.println(" b = " +b);
}
}
```

O/P :-

b = 10.0

Inheritance :- Extending a class

Def^n:- The mechanism of deriving a new class from the old one is called Inheritance.

The old class is known as a base class & derived class is known as subclass or child class.

The Inheritance allows subclasses to inherit all the variables & methods of their parent classes. Inheritance can be of different forms

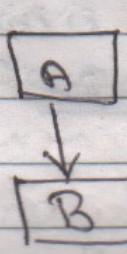
1) single Inheritance
2) multiple Inheritance

3) Hierarchial Inheritance

4) Multilevel Inheritance.

1) Single level Inheritance :- [only one super class]

In single level Inheritance only one base class / superclass will be there, using which subclass is derived.

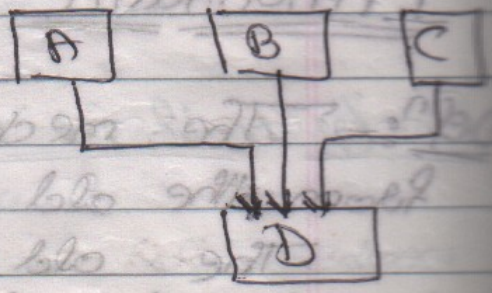


Superclass

Subclass

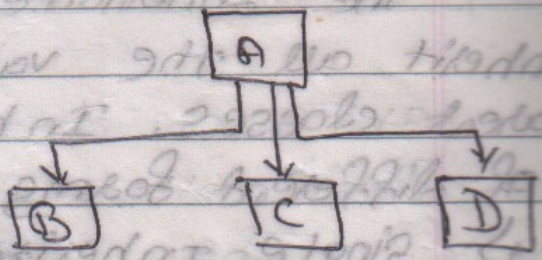
2) Multiple Inheritance :- [several superclass]

In multiple Inheritance a subclass is derived by using several/many super classes.



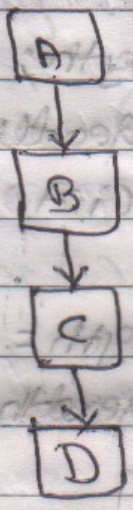
3) Hierarchial Inheritance :- [one super class many sub class]

In hierarchial Inheritance there will be one superclass, using which many sub classes are derived.



4) Multilevel Inheritance :- [Derived from derived class]

In Multilevel Inheritance at the first level subclass is 'B' derived from the Base class 'A', then ~~the~~ at second level class 'C' is derived from the derived class 'B', like this it continues to third level.



Syntax for Defining Subclass :-

```
class subclassname extends superclassname
{
    variable declaration;
    method declaration;
}
```

The keyword "extends" signifies that the properties of the superclassname are extends to the subclassname.

Lab

Program to illustrate concept of Single inheritance.

```
class Room
{
  int length;
  int breadth;
  Room (int x, int y)
  {
    length = x;
    breadth = y;
  }
  int area ()
  {
    return (length * breadth);
  }
}

class BedRoom extends Room
{
  int height;
  BedRoom (int x, int y, int z)
  {
    super (x, y);
    height = z;
  }
  int volume ()
  {
    return (length * breadth * height);
  }
}
```

c.

```
class InheritTest
{
    Public Static void main (String args[])
    {
        BedRoom room1 = new BedRoom (14,12,10);
        int area1 = room1.area();
        int volume1 = room1.volume();
        System.out.Println ("Area1 = " +area1);
        System.out.Println ("volume = "+volume);
    }
}
```

o/p :-

Area 1 = 168
volume 1 = 1680

Super :- (Subclass constructors)

The subclass constructor uses the keyword super to invoke the constructor method of the superclass. conditions for using the keyword super are as follows

1) super can be used within a subclass constructor method.
2) The call to super class constructor must appear as the first statement with in the subclass constructor.

3) The Parameters in the super call must match the order & type of the instance variable declared in the superclass

OVERRIDING METHODS

There may be occasions when we want an object to respond to the same method but have different behaviour. When that method is called. That means, we should override the method defined in the superclass. This is possible by defining a method in the subclass that has the same name, same arguments & same return type as a method in the superclass. Then, when that method is called, the method defined in the subclass is invoked & executed instead of the one in the superclass. This is known as overriding

* program illustrates the concept of overri
The method display() is overrides

PAGE NO.:
DATE : / /

Illustration method overriding.

```
class super1
{
  int x;
  super1(int x)
{
   this.x = x;
}
  void display()
{
    System.out.println("super x = " + x);
}
  }
class sub extends super1
{
  int y;
 sub (int x, int y)
 {
  super(x);
   this.y = y;
 }
  void display()
{
 System.out.println("super x = " + x);
 System.out.println("sub y = " + y);
}
}
```

```
class overridingTest
{
   public static void main (String args[])
{
   Sub S1 = new Sub (100, 200);
   S1. display ();
}
}
```

Out put

Super X = 100
Sub y = 200

* FINAL VARIABLES & METHODS

All methods & variable can be overriden by deafault in subclasses. If we wish to prevent the subclasses from overriding the members of the superclass, we can declare them as final using the keyword final as a modifier.

Example

```
final int size = 100;
final void showstatus () {.....}
```

Making a method final ensures that the functionality defined in this method

will never be altered in any way.

* FINAL CLASSES

Sometimes we may like to prevent a class being further subclasses for security reasons. A class that cannot be subclassed is called a final class. This is achieved in Java using the keyword final as follows.

```
final class Aclass {.....}
final class Bclass extends Someclass{.....}
```

Any attempt to inherit these classes will cause an error & the compiler will not allow it.

* FINALIZER METHODS

Constructor method is used to initialize an object when it is declared. This process is known as initialization. similarly, Java supports a concept called finalization, which is just opposite to initialization.

The finalizer method is simply finalize() & can be added to any class.

Java calls that method whenever it is about to reclaim the space for that object.

* ABSTRACT METHODS & CLASSES

By making a method final we ensure that the method is not redefined in a subclass. That is, the method can never be subclassed. Java allows us to do something that is exactly opposite to this. That is we can indicate that a method must always be redefined in a subclass, thus making overriding compulsory. This is done using the modifier keyword abstract in the method definition.

Example

```
abstract class Shape
{
  .....
  abstract void draw();
  .....
}
```

While using abstract classes, we must satisfy the following conditions

- We cannot use abstract classes to instantiate object directly. for EX//
  Shape S = new Shape( )
  is illegal because Shape is an abstract class.
- The abstract methods of an abstract class must be defined in its subclass.
- We cannot declare abstract constructors or abstract static methods

Method with varargs

Chapter - 4 [10 marks]

Interfaces : Multiple Inheritance

* Introduction

Java does not support Multiple-inheritance. That is, classes in Java cannot have more than one superclass. For instance, a definition like.

```
class A extends B extends C
{
    ----
}
```

- is not permitted in Java. Java provides an alternate approach known as interfaces to support the concept of Multiple inheritance.

* Defining Interfaces.

Interface is basically a kind of class. Like classes, interfaces contains methods and variables but with a major difference. The differen-ce.

The difference is that interfaces define only abstract methods and final fields. This means that interfaces do not specify any code to implement these methods and data fields contain only constants. i.e., it is the responsi-bility of the class that implements an interface to define the code for implementation of these Methods.

The syntax for defining an interface is very similar to that for defining a class. The general form of an interface definition is:

Syntax:

```
interface InterfaceName
{
    Variable declaration;
    methods declaration;
}
```

Here, Interface is the key word and InterfaceName is any valid Java Variable (Just like class names). Variables are declared as follows.

Chapter - 4

```
Static final type VariableName = Value;
```

Methods declaration will contain only a list of methods without any body statements.

Ex:

```
return_type methodName1(parameter_list);
```

An example of an interface definition that contains two variables and one method

```
interface Item
{
    static final int code = 1001;
    static final String name = "Fan";
    void display();
}
```

The code for the method is not included in the interface and the method declaration simply ends

with a semicolon. The class that implements this interface must define the code for the method.

① Difference between class and Interface

Class	Interface
* The member of a class can be constant or variables.	* The member of an interface are always declared as constant, i.e their value are final.
* The class definition can contain the code for each of its methods. That is, the methods can be abstract or non-abstract.	* The methods in an interface are abstract in nature, i.e there is no code associate with them. It is later defined by the class that implement the interface.
* It can be instantiated by declaring objects.	* It cannot be used to declare objects. It can only be inherited by a class.

Class	Interface
* It can use various access specifiers like public, private, or protected	* It can only use the public access specifiers.

* Extending Interfaces:

Like classes, interfaces can also be extended. That is, an interface can be subinterfaced from other interfaces. The new subinterfaces will inherit all the members of the superinterface in the manner similar to subclasses. This is achieved using the keyword extends as shown below.

```
interface name2 extends name1
{
    body of name2
}
```

For example, we can put all the constants in one interface and the methods in the other. This will enables us to use the constants in classes where the Methods are not required.

Eg:

```
Interface ItemConstants
{
    int code = 1001;
    String name = "Fan";
}
Interface Item extends ItemConstants
{
    void display();
}
```

The Interface Item would inherit both the constants code and name into it. Note that the variables name and code are declared like simple variables. It is allowed because all the variables in an interface are treated as constants although the

keywords final and static are not present.

Imp. ** We can also combine several interfaces together into a single interface. following declaration are valid.

```
interface ItemConstants
{
    int code = 1001;
    String name = "Fan";
}
interface Itemmethods
{
    void display();
}
```

→ interface Item extends ItemConstants, Itemmethods
{
}

While interfaces are allowed to extend to other interfaces, subinterfaces cannot define the methods declared in the superinterfaces. After all, subinterfaces are still interfaces, not classes. Instead it is the responsibility of any class that implements the derived interface to define all the methods. Note that when an interface extends two or more interfaces, they are separated by commas.

* Implementing Interfaces.

Interfaces are used as "super classes" whose properties are inherited by classes. It is therefore necessary to create a class that inherits the given interfaces. This is done as follows:

```
class classname implements interfacename
{
    body of classname
```

Here the class classname "implements" the interface interfacename. A more general form of implementation may look like this:

```
class classname extends superclass
    implements interface1, interface2, ...
{
    body of classname
}
```

This shows that a class can extend another class while implementing interfaces.

13, 15, 16

14, 12,

12 (a)

PAGE NO.:
DATE: / /

b) AP to illustrate implementing interfaces.

```
interface Area
{
final static float pi = 3.142F;
float compute (float x, float y);
}
class Rectangle implements Area
{
    public float compute (float x, float y)
    {
        return (x*y);
    }
}
class circle implements Area
{
  public float compute (float x, float y)
  {
      return (pi * x * x);
  }
}
class Interface Test
{
public static void main (String args[])
```

```
        Rectangle rect = new Rectangle();
    Circle cir = new Circle();
    //Area area;
        area = rect;
System.out.println(" Area of Rectangle
                + area.compute(10,20));

    area = cir;
System.out.println(" Area of Circle = "
                + area.compute(10,0));
        }
    }
```

O/P

Area of Rectangle = 200
Area of Circle = 314.

ACCESSING INTERFACE VARIABLES. *

Interfaces can be used to declare a set of constants that can be used in different classes. Since such interfaces do not contain methods. The values can be used in any method, as part of any variable declaration, or anywhere where we can use a final value.

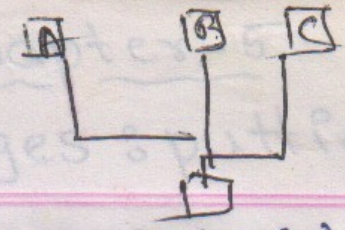
Ex

```
interface A
{
    int m = 10;
    int n = 50;
}
class B implements A
{
    int x = m;
    void methodB(int size)
    {
        if(size < m)
        {
        }
    }
}
```

* WAP to illustrates the implementation of the concept of multiple inheritance using interfaces.

```
class Student
{
    int rollNumber;
    void getNumber(int n)
    {
        rollNumber = n;
    }

    void putNumber()
    {
        System.out.println("Roll No:" + rollNumber);
    }
}
class Test extends Student
{
    float part1, part2;
    void getMarks(float m1, float m2)
    {
        part1 = m1;
        part2 = m2;
    }
```



585

PAGE NO.:
DATE: / /

```
Void putMarks()
  {
System.Out.println("marks obtained ");
System.out.println("part1=" +part1);
System.Out.println("part2=" +part2);

    }
  }
interface sports
  {
    float sportWt = 6.0F;
    void putWt();
  }
Class Results extends Test implements
                                Sports
    {
      float total;
 public void putWt()
    {
System.Out.println("Sports Wt = " +sportWt);

    }
 Void display()
   {
```

```
        total = part1 + part2 + sportWt;
        putNumber();
        putMarks();
        putWt();
        System.out.println("Total Score = "+total);
    }
}

class Hybrid
{
public static void main(String args[])
    {
    Results Student1 = new Results();
    Student1.getNumber(1234);
    Student1.getMarks(27.5F, 33.0F);
    Student1.display();
    }
}
```

O/p

Roll No : 1234

Marks obtained

Part 1 = 27.5

Part 2 = 33

SportWt = 6 , Total Score = 66.5

Chapter-5

15 Marks

Packages : putting classes together

Introduction :→

If we need to use classes from other programs without physically copying them into the program under development? This can be accomplished in Java by using what is known as packages, a concept similar to "class libraries" in other languages. Another way of achieving the reusability in Java, therefore, is to use packages.

Packages are Java's way of grouping a variety of classes and/or interfaces together. The grouping is usually done according to functionality. In fact, packages act as "containers" for classes.

By organizing our classes into packages we acheive the following benefits :

(*) The classes contained in the packages of other programs can be easily reused.

(*) In packages, classes can be unique compared with classes in other packages. i.e two classes in two different

Packages can have the same name. They may be referred by their fully qualified name, comprising the package name and the class name.

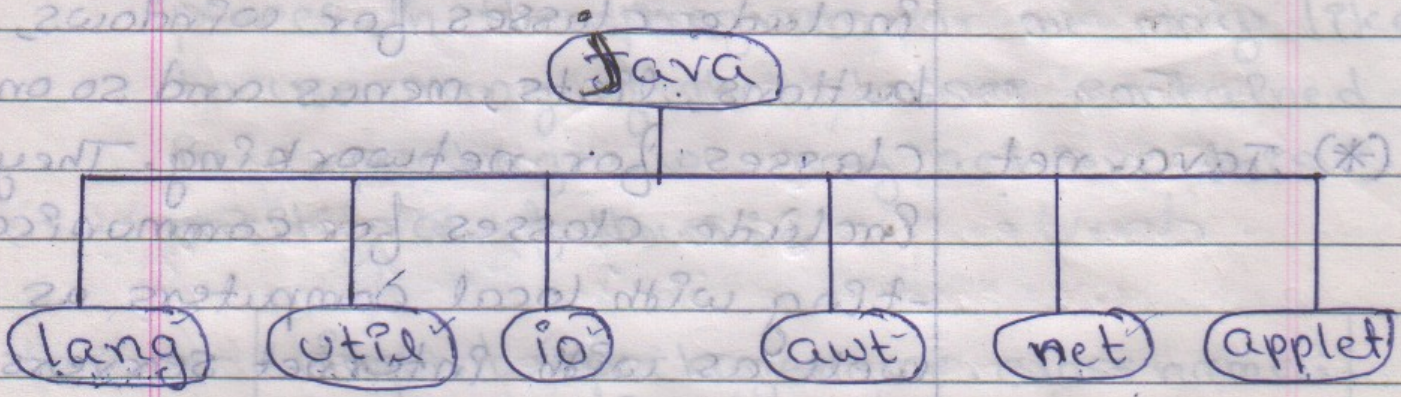
(*) Packages provide a way to "hide" classes thus preventing other programs or packages from accessing classes that are meant for internal use only.

(*) Packages also provide a way for separating "design" from "coding". First we can design classes and decide their relationships, and then we can implement the Java code needed for the methods. It is possible to change the implementation of any method without affecting the rest of the design.

JAVA API PACKAGES

Java API provides a large number of classes grouped into different packages according to functionality. Most of the time we use the packages available with the Java API.

Frequently used API packages.



Java System packages and their classes.

Package Name	Contents.
(*) Java.lang	language support classes. These are classes that Java compiler itself uses and therefore they are automatically imported. They include classes for primitive types, Strings, Math functions, threads and exceptions.
(*) Java.util	Language utility classes such as vectors, hash tables, random numbers, date etc.).
(*) Java.io	Input/output support classes. They provide facilities for the input and output of data.

Continue...

(*) Java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on..
(*) Java.net	Classes for networking. They include classes for communica-ting with local computers as well as with internet servers.
(*) Java.applet	classes for creating and implementing applets.

Using System packages.

There are two ways of accessing the classes stored in a package. The ~~package~~ first approach is to use the fully qualified class name of the class that we want to use. This is done by using the package name containing the class and then appending the class name to it using the dot operator.

For example, if we want to refere to the class 'color' in the 'awt' package, then we may do so as follows:

Java.awt.colour

But, in many situations, we might want to use a class in a number of places in the program or we may like to use many of the classes contained in a package. We may achieve this easily as follows:

```
import packagename.classname;
(or)
import packagename.*;
```

These are known as import statements and must appear at the top of the file, before any class declarations. import as a keyword.

Naming Conventions

Packages can be named using the standard Java naming rules. By convention, however, packages begin with lowercase letters. This makes it easy for users to distinguish package names from class names when looking at an explicit

reference to a class. we knows that all class names, again by convention, begin with an uppercase letter.

For example,

```
double Y = Java.lang.Math.sqrt(x);
```

package name — class name — method name

Creating Packages.

First declare the name of the package using the "package" keyword followed by a package name. This must be the first statement in a Java source file (except for comments and white spaces). Then we define a class, just as we normally define a class.

```
package firstpackage; // package declaration
public class firstclass // class definition.
{
    Body of class
}
```

PAGE NO.:
DATE: / /

Creating our own package involves the following steps.

1) Declare the package at the beginning of a file using the form.

```
Package packagename;
```

2) Define the class that is to be put in the package and declare it public.
3) Create a subdirectory under the directory where the main source files are stored.
4) Store the listing as the classname.java file in the subdirectory created.
5) Compile the file. This creates .class file in the subdirectory.

Remember that case is significant and therefore the subdirectory name must match the package name exactly.

A java package file can have more than one class definitions. In such cases, only one of the classes may be declared public and that class name with .java extension is the source file name. When a source file with more than one class

definition is compiled, Java creates independent .class files for those classes.

Accessing a package

We use the import statement when there are many references to a particular package or the package name.

The general form of import statement for searching a class is as follows:

```
import package1 [.package2] [.package3]
                                .classname;
```

Here package1 is the name of the top level package, package2 is the name of the package that is inside the package1, and so on. We can have any number of packages in a package hierarchy. Finally, the explicit classname is specified.

The statement must end with a semicolon(;). The import statement should appear before any class definitions in a source file. Multiple import statements are allowed.

The following is an example of importing a particular class:

```
import firstpackage.secondpackage.Mycloss;
```

We can also use another approach as follows:

```
import packagename.*;
```

Here, packagename may denote a single package or a hierarchy of packages as mentioned earlier. The star(*) indicates that the compiler should search this entire package hierarchy when it encounters a class name.

Using a package:

Consider simple programs that will use classes from other packages.

The listing below shows a package named package1 containing a single class, classA.

```
package package1;
public class classA
{
public void displayA()
{
System.out.println("class A");
}
}
```

This source file should be named classA.java and stored in the subdirectory package1 as stated earlier.

Now compile this java file. The resultant ~~classA.java~~ classA.class will be stored in the same subdirectory.

Now consider the listing shown below:

```
import package1.classA;
class packageTest1
{
public static void main(String args[])
{
classA objectA = new classA();
objectA.displayA();
}
}
```

This listing shows a simple program that imports the class, classA from the package package1. The source file should be saved as packageTest1.java and then compiled. The source file and the compiled file would be saved in the directory of which package1 was a subdirectory. Now we can run the program and obtain the results.

During the compilation of packageTest1.java the compiler checks for the file classA.class in the package1 directory for information it needs, but it does not actually include the code from classA.class in the file packageTest1.class. When the

packageTest1 program is run, Java looks for the file packageTest1.class and loads it using something called class loader. Now the interpreter knows that it also needs the code in the file classA.class and loads it as well.

Let us consider another package named package2 containing again a single class as shown below:

```
package package2;
public class classB
{
protected int m=10;
public void displayB()
{
   System.out.println("class B");
   System.out.println("m=" +m);
}
}
```

7000 - 345 Indiabulls HSG 20-9-14

342

PAGE NO.:
DATE: 9/9/14

Importing classes from other packages.

```
import package1.classA;
import package2.*;
class packageTest2
{
  public static void main(String args[])
  {
    classA objectA = new classA();
    classB objectB = new classB();
    objectA.displayA();
    objectB.displayB();
  }
}
```

This program may be saved as packageTest2.java, compiled and run to obtain the results.

OUTPUT

```
Class A
class B
m = 10.
```

IMP.

Subclassing an imported class.

```
// package Test3.java

import package2.classB;
class classC extends classB
{
   int n=20;
   void displayC()
   {
   System.out.println("class C");
   System.out.println(" m=" +m);
   System.out.println(" n=" +n);
   }
}
class packageTest3
{
   public static void main(String args[])
   {
   classC objectC = new classC();
   objectC.displayB();
   objectC.displayC();
   }
}
```

Adding a class to a package.

It is simple to add a class to an existing package.

Consider the following package:

```
package p1;
public class A
{
    // Body of A
}
```

The package P1 contains one public class by name A. Suppose we want to add another class B to this package. This can be done as follows:

1) Define the class and make it public.
2) Place the package statement.

```
package p1;
```

before the class definition as follows:

```
package p1;
public class B
{
    // Body of B
}
```

3) Store this as B.java file under the directory p1.

4) Compile B.java file. This will create a B.class file and place it in the directory p1.

Note that we can also add a non-public class to a package using the same procedure.

Now, the package p1 will contain both the classes A and B. A statement like

```
import p1.*;
```

will import both of them.

Since a java source file can have only one class declared as public, we cannot put two or more public classes together in a .java file.

This is because of the restriction that the file name should be same as the name of the public class with .java extension.

If we want to create a package with multiple public classes in it, we may follow the following steps:

1) Decide the name of the package.
2) Create a subdirectory with this name under the directory where main source files are stored.
3) Create classes that are to be placed in the package in separate source files and declare the package statement.

package packagename;

at the top of each source file.

4) Switch to the subdirectory created earlier and compile each source file. When completed, the package would contain .class files of all the source files.

Hiding classes

When we import a package using asterisk(*), all public classes are imported. However, we may prefer to 'not import' certain classes. i.e we may like to hide these classes from accessing from outside of the package. Such classes should be declared "not public".

Example:

```
package p1;
public class X //public class,
{                    available outside.
    //body of X
}
class Y              // not public, hidden
{
    //body of Y
}
```

Class Y is not Public, so this class can't be seen & used only by other classes in the same Package. & also we can't create object of class Y.

Chapter - 7

10 marks

Managing Errors & Exceptions.

Types of Errors

Errors may broadly classified into two categories.

* Compile-time errors
* Run-time errors.

Compile-time Errors.

All syntax errors will be detected and displayed by Java compiler and therefore these errors are known as compile-time errors. Whenever the compiler displays an error, it will not create the .class file. It is therefore necessary that we fix all the errors before we can successfully compile and run the program.

Illustrate compile-time errors

```
class Error1
{
  public static void main (String args[])
    {
System.out.println("Hello Java!") // missing ;
    }
}
```

The following message will be displayed in the screen:

```
Error1.java : 5 : ';' expected.
System.out.println("Hello java!")
                                ^
1 error.
```

We can now go to the appropriate line, correct the error, and recompile the program.

The compile-time errors are due to typing mistake. Typographical errors are hard to find. we may have to check the code word by word, or even-

-character by character. The most common problems are:

* Missing semicolons.
* Missing (or mismatch of) brackets in classes and Methods.
* Misspelling of identifiers and keywords.
* Missing double quates in strings
* Use of undeclared variables.
* Incompatible types in assignments/initialization
* Bad references to objects.
* And so on.,

Run-Time Errors [Symantic / logical Error]

Sometimes, a program may compile successfully creating the .class file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as Stack overflow.

Most common run-time errors are.

* Dividing an integer by zero
* Accessing an element that is out of the bounds of an array.
* Trying to store a value into an array of an incompatible class or type.

* Trying to cast an instance of a class to one of its subclasses.
* Passing a parameter that is not in a valid range or value for a method.
* Trying to illegally change the state of a thread
* Attempting to use a negative size of an array.
* Using a null object reference as a legitimate object reference to access a method or a variable.
* Converting invalid String to a number
* Accessing a character that is out of bounds of a string.
* And many more.

13.2 Illustration of Run-time errors

```
Class Error2
{
public static void main (String args[])
{
int a = 10;
int b = 5;
int c = 5;
```

continue.

10/0 -> ∞

PAGE NO.:
DATE: / /

int x = a / (b-c); // Division by zero
System.out.println("x=" + x);
int y = a / (b+c);
System.out.println("y=" + y);
}
}

While executing, it displays the following message and stops without executing further statements.

java.lang.ArithmeticException: / by zero
at Error2.main(Error2.java:8)

When java run-time tries to execute a division by zero, it generates an error condition, which causes the program to stop after displaying an appropriate message.

EXCEPTIONS

An exception is a condition that is caused by a run-time error in the program. When the java interpreter encounters an error such as dividing an

integer by zero, it creates an exception object and throws it (i.e informs us that an error has occurred).

If the exception object is not caught and handled properly, the interpreter will display an error message and will terminate the program. If we want the program to continue with the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective action. This task is known as "exception handling".

The purpose of exception handling mechanism is to provide a mean to detect and report an "exceptional circumstances" so that appropriate action can be taken. The mechanism suggest incorporation of a separate error handling code that performs the following tasks.

1) Find the problem (Hit the exception).
2) Inform that an error has occurred (throw the exception).
3) Receive the error information (catch the exception).
4) Take corrective actions (Handle the exception).

Exceptions in Java can be categorised into two types.

* Checked exceptions: These exceptions are explicitly handled in the code itself with the help of try, catch blocks. Checked exception are extended from the java.lang.Exception class.

* Unchecked exception: these exceptions are not essentially handled in the program code; instead the JVM handles such exceptions. Unchecked exceptions are extended from the java.lang.RuntimeException class.

Java uses a keyword try to preface a block of code that is to cause an error condition and "throw" an exception. A catch block defined by the keyword catch "catches" the exception "thrown" by the try block and handles it appropriately. The catch block is added immediately after the try block. The following example illustrates the use of simple try and catch statements.

```
try
{
    statement1; // generates an exception.
}
catch (Exception-type e)
{
    Statement2; // processes the exception.
}
```

The try block can have one or more statements that could generate an exception. If any one statement generates an exception jumps to the catch block that is placed next to the try block.

program to illustrate using try and catch for exception handling

```
class Error3
{
public static void main (String args[])
{
    int a = 10;
    int b = 5;
    int c = 5;
    try
    {
int x = a/(b-c);  // Exception here
    }
    catch (ArithmeticException e)
    {
System.out.println (" Division by zero");
    }
int y = a/(b+c);
System.out.println (" y = " + y);
}
}
```

o/p

Division by zero

y = 1.

* program for catching invalid Command line arguments.

```
class CLineInput
{
public static void main (String args[])
{
int invalid = 0;        // Number of invalid
int number, count = 0;     arguments.
for(int i=0; i<args.length; i++)
  {
  try
  {
  number = Integer.parseInt(args[i]);
  }
Catch (NumberFormatException e)
  {
  invalid = invalid + 1;  // caught an invalid
                             number.
System.out.println(" Invalid Number " +args[i]);
    continue;
}
```

```
        count = count + 1;
    }
System.out.println("Valid Numbers = " + count);
System.out.println("Invalid Numbers = " + invalid);
    }
}
```

When we run the program with the command line;

Java CLineInput 15 25.75 4.0 Java 10.5 6

it produce the following output.

```
Invalid Number : 25.75
Invalid Number : Java
Invalid Number : 10.5
Valid Number = 3
Invalid Number = 3
```

Multiple Catch Statements

It is possible to have more than one catch statement in the catch block as illustrated below;

```
---
try
{
    statement; // generates an exception
}
catch (Exception-Type-1 e)
{
    statement; // processes exception type1
}
catch (Exception-Type-2 e)
{
    statement; // processes exception
                  type2
}
.
.
.
catch (Exception-Type-N e)
{
    statement;
}
```

When an exception in a try block is generated, the java treats the multiple catch statement like cases in a switch statement. The first statement whose parameter matches with the exception object will be executed, and the remaining statements will be skipped.

```
Catch (Exception e);
```

The catch statement simply ends with a semicolon, which does nothing.

* program to illustrate the use of Multiple catch blocks.

```
Class Error4
{
public static void main (String args[])
  {
  int a[] = {5, 10};
  int b = 5;
  try
  {
int x = a[2] / b - a[1];
  }
```

```
Catch (ArithmeticException e)
{
System.out.println("Division by zero");
}
Catch (ArrayIndenOfBoundsException e)
{
System.out.println("Array inden error");
}
Catch (ArrayStoreException e)
{
System.out.println("wrong data type");
}
int y= a[1]/a[0];
System.out.println("y="+y);
}
}
```

O/P

Array Inden error

y= 2.

Using Finally Statement

Java supports another statement known as finally statement that can be used to handle an exception that is not caught by any of the previous catch statements, finally block can be used to handle any exception generated within try block. It may be added immediately after the try block or after the last catch block. as shown below.

(1)

```
try
{
  ---
}
finally
{
  ---
}
```

(2)

```
try
{
  ---
}
catch(---)
{
  ---
}
catch(---)
{
  ---
}
finally
{
  ---
}
```

Throwing our own Exceptions.

There may be times when we would like to throw our own exception. We can do this by using the keyword throw as follows:

```
throw new ThrowableSubclass;
```

.

```
throw new ArithmeticException();
throw new NumberFormatException();
```

program demonstrate the use of a user-defined subclass of Throwable class. Note that Exception is a subclass of Throwable and therefore MyException is a subclass of Throwable class. An object of a class that extends Throwable can be thrown and caught.

* program to illustrate the concept of Throwing our own exception.

```
import java.lang.Exception;
class MyException extends Exception
  {
   MyException (String message)
    {
      super(message);
    }
  }
Class TestMyException
  {
    public static void main (String args[])
      {
   int x=5; y=1000;
   try
   {
float z=(float)x/(float)y;
     if (z<0.01)
     {
       throw new MyException("Number is
                    -too small");
     }
}
```

```
Catch (MyException e)
    {
System.out.println("Caught my exception");
System.out.println(e.getMessage());
    }
finally
    {
      System.out.println("I am always here");
    }
}
```

o/p

Caught My exception.
Number is too small.
I am always here.

CHAPTER-06 MAX MARKS :- 15

MULTITHREADED PROGRAMMING

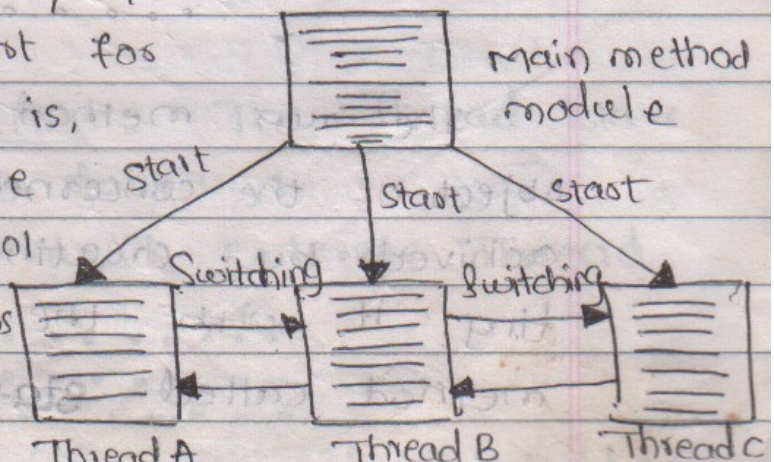
INTRODUCTION :

Multithreading is a conceptual programming paradigm (process) is devided into two or more subprograms (process), which can be implemented at the same time in parallel. For example one sub-program can display animation on the screen while another may build the next animation to be displayed. This is something similar to dividing a task ~~and assigni~~ into subtasks and assigning them to different people for execution independently and simultaneously

A thread is similar to a program that has a single flow of control. It has a beginning, a body, and an end, and executes command sequentially. In fact, all main program in our earlier examples can be called single thereaded programs. Every program will have atleast one thread as show in fig below

A unique property of java is it's support for multithreading. That is, java unables us to use multiple flows of control in developing programs

Each flow of control may be



A Multithreaded program

thought of as seperate tiny program (or module) known as thread that runs in parallel to others as shown in figure.

Since threads in java are sub-progra of main application program and share the same memory space, they are known as "lightweight threads" or "lightweight processes".

→ CREATING THREADS:- Threads are implimenting in the form of objects that contains a method called run(). The run() method is the heart and soul of any thread. It makes up the entire body of a thread and is the only method in which the threads behaviour can be implimented. A typical run() would apear as follows.

```
public void run()
{
    ..............
    .............. (statements for implime
    .............. nting thread)
}
```

The run method should be invoked by an object of the concerned thread. This can be achived by creating the thread and initiating it with the help of another thread method called start().

A new thread can be created in two ways.

① :- By creating a thread class :- Define a class that extends **Thread** class and override its **run()** method with the code required by the thread.

② :- By converting a class to thread :- Define a class that impliment **Runnable** interface. the **runnable** interface has only one method, **run()**, that is to be defined in the method with the code to be executed by the thread

→ EXTENDING THE THREAD CLASS :- It inludes following steps.

① :- Declaire the class as extending the **Thread** class

② :- Impliment the **run()** method that is responsible for executing the sequence of code that the thread will execute.

③ :- Create the thread object and call the **start()** method to initiate the thread execution.

→ DECLAIRING THE CLASS :- The **Thread** class can be extended as follows :

```
class Mythread extends Thread
{
  .....
  .....
  .....
}
```

Now we have a new type of thread My-thread

→ **Implementing the run() Method :-**

The **run()** method has been inherited by the class **MyThread**. We have to override this method in order to ~~override~~ implement the code to be executed by our thread. The basic implimentation of **run()** will be look like this:

```
public void run()
{
    ..........
    .......... // Thread code here
    ..........
}
```

When we start the new thread, java calls the thread's **run()** method, it is the **run()** where all the action takes place.

→ **AN EXAMPLE OF USING THE THREAD CLASS :-**

Program illustrates the use of **Thread** class for creating and running threads in an application. The program creates threee threads A, B, and C for undertaking threee different tasks. The **main** method in the **Thread Test** class also constitutes another thread which we may call the "main thread".

The main thread dies at the end of it's main method. However, before it dies, it creates and starts all the threeads A, B and C

Note the statements like:

```
new AC().start();
```

in the main thread. This is just a compact way of starting a thread. This is equivalent to:

```
A threadA = new AC();
threadA.start();
```

Immidialely after the thread A is started, there will be two threads running in the program: the main thread and the thread A. The start() method returns back to the main thread immidiately after invoking the run() method, thus allowing the main thread to start the thread B.

PROGRAM TO CREATING THREADS USING THE THREAD CLASS :-

→

```
class A extends Thread
{ public void run()
  { for (int i=1; i<=5; i++)
    { System.out.println("\t From ThreadA : i="+i);
    }
    System.out.println(" Exit from A");
  }
}
class B extends Thread
{ public void run()
  { for(int j=1; j<=5; j++)
    { System.out.println("\t From ThreadB: j="+j);
    }
```

```
        System.out.println("Exit from B");
    }
}
class C extends Thread
{   public void run()
    {   for (int k=1; k<=5; k++)
        {   System.out.println("\t From Thread C: k="+k);
        }   System.out.println("Exit from C");
    }
}
class ThreadTest
{   public static void main(String args[])
    {   new A().start();
        new B().start();
        new C().start();
    }
}
```

OUT-PUT OF PROGRAM WOULD BE

First run

```
From Thread A : i = 1
From Thread A : i = 2
From Thread B : j = 1
From Thread B : j = 2
From Thread C : k = 1
From Thread C : k = 2
From Thread A : i = 3
From Thread A : i = 4
From Thread B : j = 3
From Thread B : j = 4
```

From Thread C: K = 3
From Thread C: K = 4
From Thread A: i = 5
Exit from A
From Thread B: j = 5
Exit from B
From Thread C: K = 5
Exit from C

Similarly, it starts C thread. By the time the main thread has reached the end of it's **main** method, there are a total of four seperate threads running in parallel.

The out-put from the threads are not specially sequential. They do not follow any special (specific) order. They are running independently of one another and each executes whenever it has a chance.

→ STOPPING AND BLOCKING A THREAD

STOPPING A THREAD :- whenever we want to stop a thread from running further, we may do so by calling it's **stop()** method, like

```
aThread.stop();
```

This statement causes the thread to move to the **dead** state. A thread will also move to the dead state autometically when it reaches

the ends of it's method. The **stop()** method may be used when the premature death of a thread is desired.

→ **BLOCKING A THREAD :-** A Thread can also be temporarily suspended or blocked from entering into the runnable & subsequently running state by using either of the following thread methods:

```
sleep()     // blocked for a specified time
suspend()   // blocked untill further orders
wait()      // blocked untill certain condition occurs
```

These methods cause the thread to go into the **blocked (or not-runnable)** state. The thread will return to the runnable state when the specified time is elapsed in the case of **sleep()**, the **resume()** method is invoked in the case of **suspend()**, and the **notify()** method is called in the case of **wait()**.

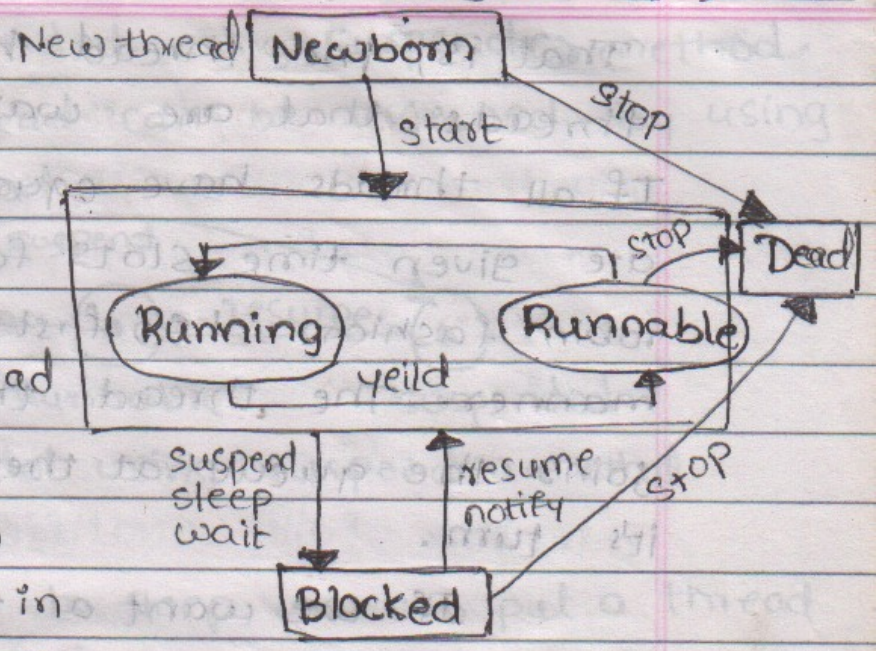
IMP → **LIFE CYCLE OF A THREAD :-** During the life time of a thread, there are many states it can enter. They include:

1. :- Newborn State
2. :- Runnable state (ready state)
3. :- Running state,

State transition daigram of thread

(4)& Blocked state

(5)& Dead state

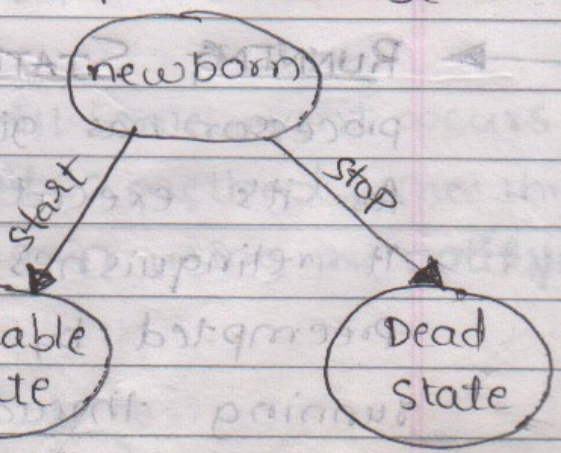


→ NEWBORN STATE :-

When we create a thread object, the thread is born and is said to be in new born state. The thread is not yet scheduled for running at this state, we can do only one of the following things with it:

- Schedule it for running using **start()** method.
- Kill it using **stop()** method.

If scheduled, it moves to the runnable state (Fig). If we attempting to use any other method at this stage an exception will be thrown



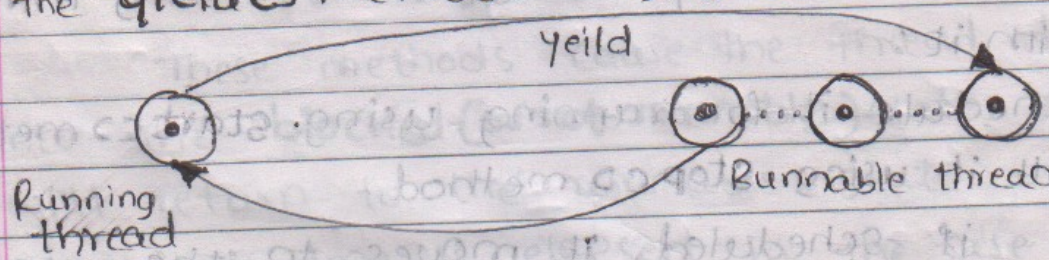
→ RUNNABLE STATE :-

Runnable state means that the thread is ready for the execution and is waiting for the availiblity of processor.

scheduling a new born thread

That is, the thread has joind the queue of threads that are waiting for execution. If all threads have equal priority, then they are given time slots for execution in round robin fashion i.e., first-come, first-serve manner. The thread that relinquishes control joins the queue at the end again waits for it's turn.

If we want a thread to relinquish control to another thread to equal priority before it's turn comes, we can do so by using the **yield()** method. (Fig) below

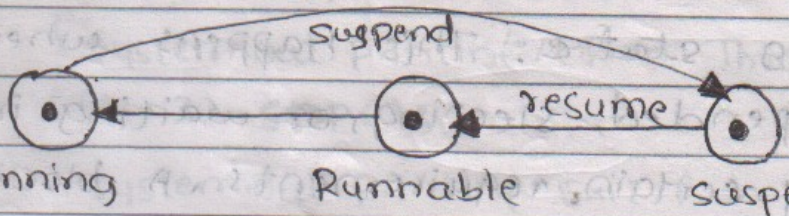


Relinquishing control using yeild() method

→ **RUNNING STATE** :- Running means that the processor has given it's time to the thread for it's execution. The threads runs untill it relinquishes control on it's own or it is preempted by a higher priority thread. A running thread may relinquish it's control in one of the following situations.

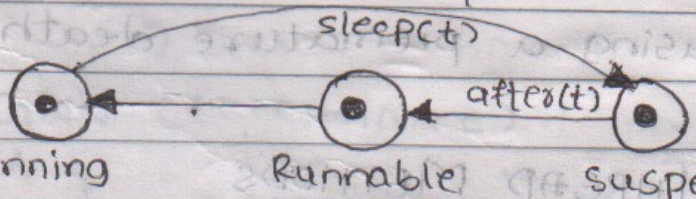
PAGE NO.:
DATE: / /

① :- It has been suspended using **suspend()** method. A suspended thread can be reviewed by using the **resum()** method.



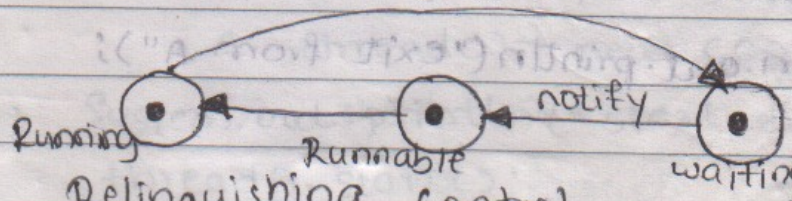
Relinquishing control using suspend() method

② :- It has been made to sleep, we can put a thread to sleep for specified time period using the method **sleep(time)** where time is in miliseconds. This means that the thread is out of the queue during this time period. The thread re-enters the runnable state as soon as this time period is elapsed.



Relinquishing control using sleep() method

③ :- It has been told to wait until some event occurs. This is done using the **wait()** method. The thread can be scheduled to run again using the **notify()** method.



Relinquishing control using wait() method.

PAGE NO.:
DATE: / /

→ **BLOCKED STATE :-** A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements. A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

→ **DEAD STATE :-** Every thread has a life cycle. A running thread ends it's life when it has completed executing it's run() method. It is natural death. However, we can kill it by sending the stop massage to it at any state thus causing a premature death to it.

MAP → **USING THREAD METHODS**

Use of yield(), stop(), and sleep() methods

```
class A extends Thread
{ public void run()
  { for (int i= 1; i<=5; i++)
    { if(i==1) yield();
      System.out.println("\t From Thread A: i= "+i);
    }
    System.out.println("exit from A");
  }
}
```

```
class B extends Thread
{   public void run()
    {  for(int j=1; j<=5; j++)
       { System.out.println("\t From Thread B: j=" +j);
      if(j==3) stop();
    }   System.out.println("Exit from B");
    }
} class C extends Thread
  {  public void run()
     {   for (int k=1; k<=5; k++)
     {  System.out.println("\t From Thread C: k=" +k);
      if (k==1)
      try {
            sleep(1000);
          }
     Catch (Exception e)
      { }
     }  System.out.println("Exit from C");
     }
  }   class ThreadMethods
{  public static void main( String args[])
   {    A  threadA = new A();
        B  threadB = new B();
        C  threadC = new C();
  System.out.println("start thread A");
  threadA.start();
```

```
System.out.println(" start thread B");
threadB.start();
System.out.println(" start thread C");
threadC.start();
System.out.println(" End of main thread");
}
}
```

Here is the out-put of the program

```
start thread A
start thread B
start thread C
    From Thread B: j=1
    From Thread B: j=2
    From Thread A: i=1
    From Thread A: i=2
End of main thread
    From Thread C: k=1
    From Thread B: j=3
    From Thread A: i=3
    From Thread A: i=4
    From Thread A: i=5
Exit from A
    From Thread C: k=2
    From Thread C: k=3
    From Thread C: k=4
    From Thread C: k=5
Exit from C
```