02/01/20 Assignment no : 1

1) Explain paradigms of Programming languages.

Paradigms of Programming give the model for the programmer to write programs. It also provides the view of the program during execution. The different paradigms of Programming languages are

- * un-structured programming.
- * Procedure programming.
- * structured programming
- * object oriented programming (oops)

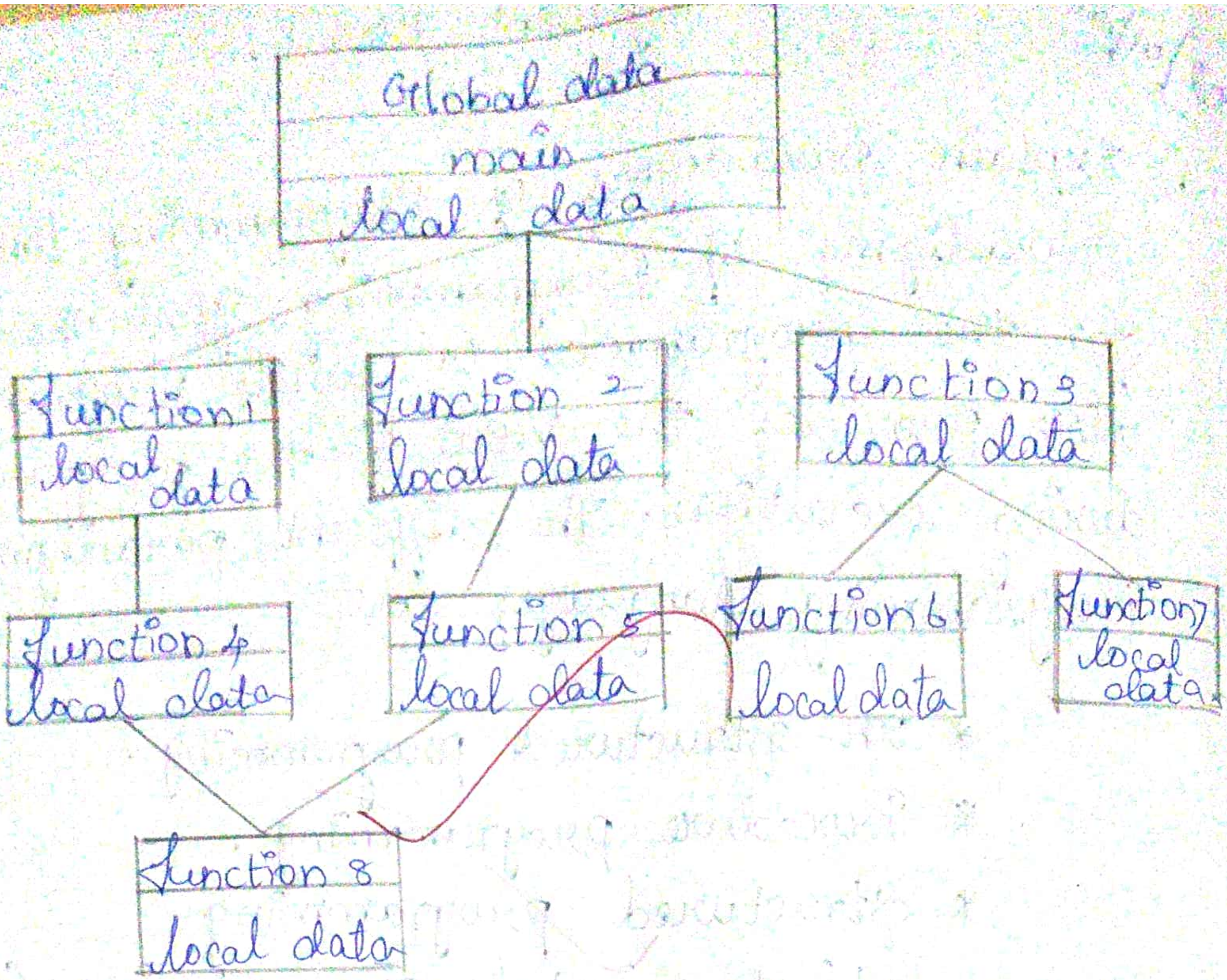
i) un-structured programming :

* Program codes are written in a single block. So, it is very difficult to follow and correct errors.

* This type of programming uses only global data and number of go to statement

Eg : BASIC

ii) Procedure Programming :



* Large Problems are divided into smaller Problems known as functions or procedures.

* It uses top-down programming technique.

* Data moves freely from one function to another.

* Data hiding is not possible.

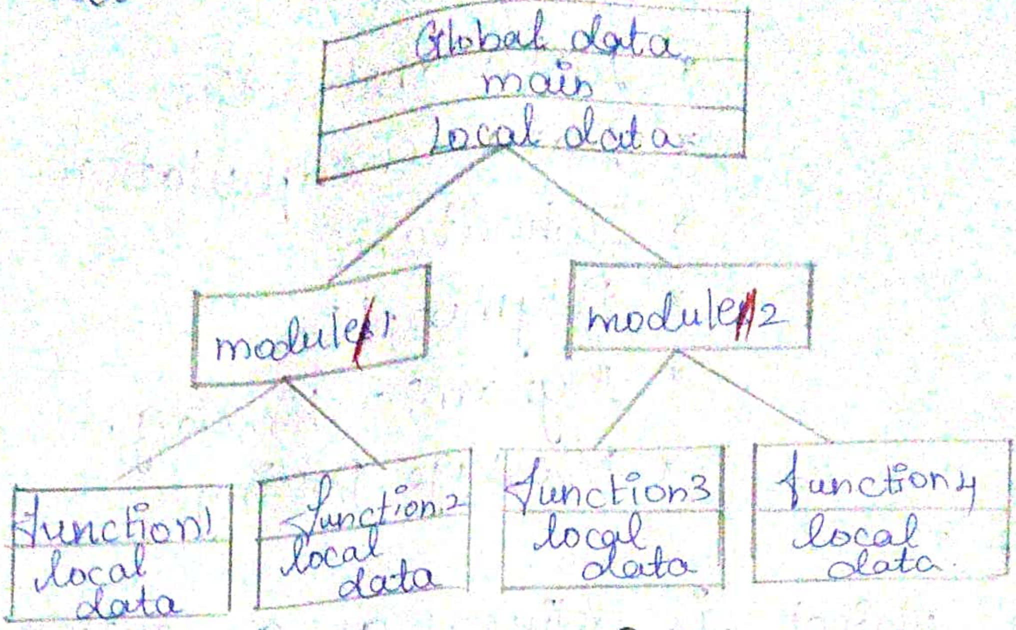
* It is difficult to add new functions and data structures.

iii) Structured Programming :

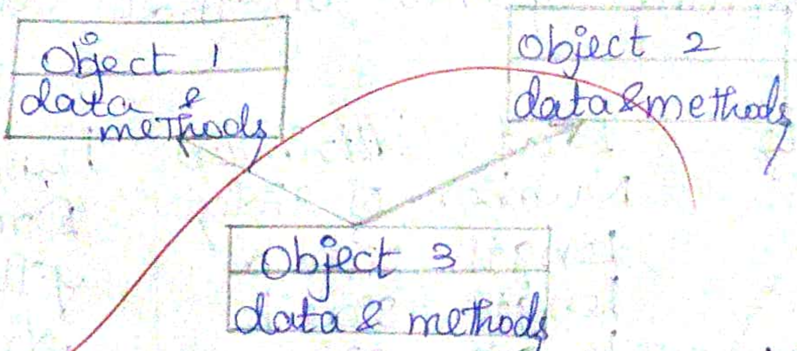
* Problem divided into modules.

* Structured Programming is a sub-set of Procedural programming. In this, the

usage of go to statement is completely removed.



iv) Object oriented programming:



* problems are divided into objects.
* It is not possible to access data freely
* Data hiding is possible.
* It uses bottom-up programming technique.
* It is easy to add new data and functions.

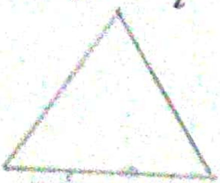
Q2) Explain about basic Concept Concept of oops:

- Object.
- Class.
- Data abstraction.
- Data encapsulation
- Inheritance
- Polymorphism.
- Dynamic binding.
- Message passing.

i) Object :

An object is defined as an entity that Contains data and its related functions. The functions operate on that data.

Objects	Data (Attributes)	Functions


Triangle
	Number of sides
	Length side a
	Length side b
	Length side c
	Border colour
	Fill colour
		Draw()
		fill colour()
		Area()
		Move()

ii) class :

A class is defined as a Collection of objects with Same type of data and functions. The functions of the class should be defined.

```
Class  Class_name
declaration of data
- - - - - - - - - - -
definition of functions
- - - - - - - - - - -
end class.
```

iii) Data abstraction:

* Abstraction is defined as a grouping of essential details and ignoring other details.

* Data abstraction is defined as a named Collection of data that describes a object in a class.

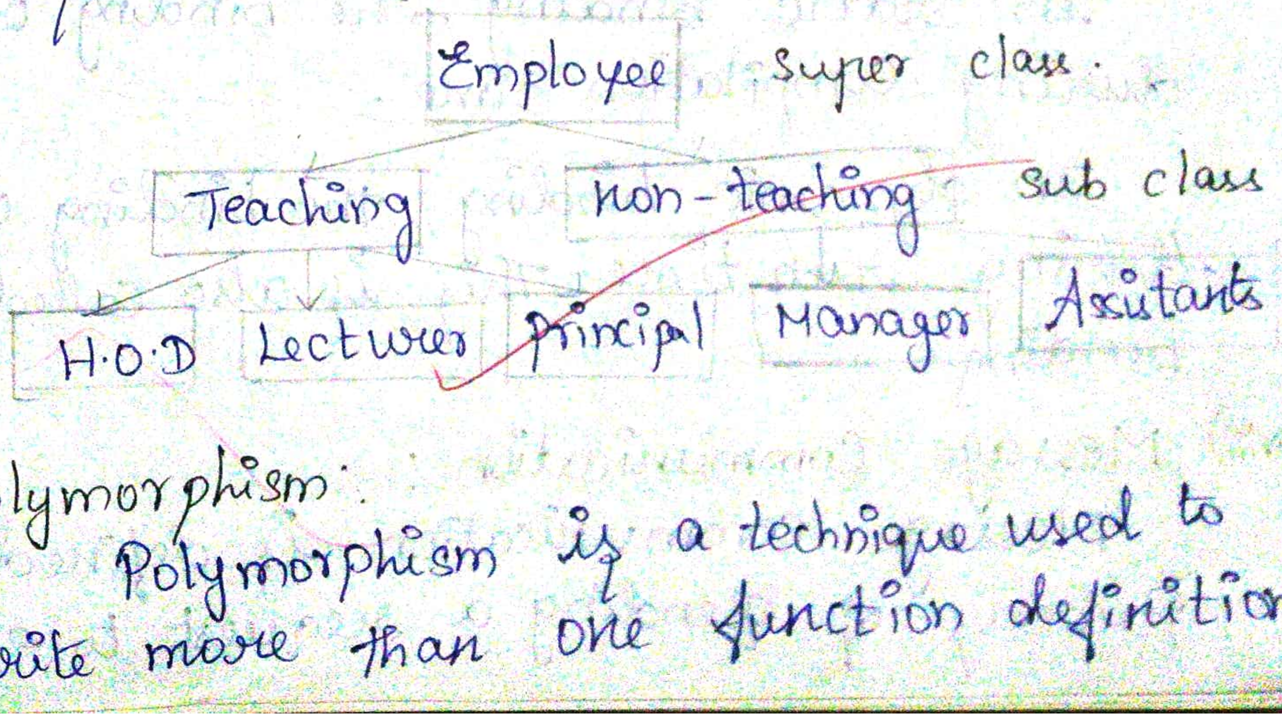
iv) Data encapsulation:

Encapsulation is a technique used to Protect the information in an object from other objects. This concept is called data hiding.

v) Inheritance:

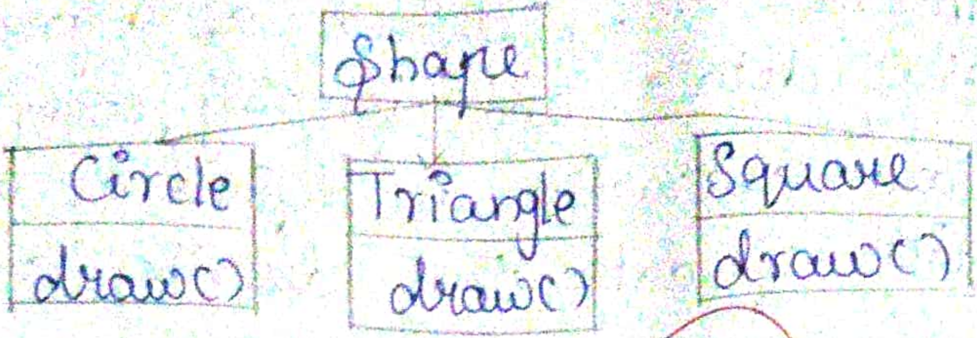
* Inheritance is a process of deriving new classes from existing classes.

* The existing classes are called base classes and the inherited classes are called derived classes.



vi) Polymorphism:

Polymorphism is a technique used to write more than one function definition with same function name



In the above classes, the function draw() is defined in all classes. But the operation of the function draw() different.

vii) Dynamic binding :

Binding is defined as the connection between the function call and its corresponding program code to be executed. There are two types of binding. They are

- * Static binding
- * Dynamic binding

In static binding, the binding occurs during compilation time.

In dynamic binding, the binding occurs during run time. This is also called late binding.

viii) Message Communication :

Message communication is defined as a process of sending a request to execute a function for an object.

object_name . message (information) ;

Eg : S1. read ( ) ;

Q. Explain about java features.

* Simple, Small and familiar.
* object oriented.
* Distributed
* Robust
* Secure
* Architecture neutral or platform independent
* Portable
* compiled and interpreted
* High performance

i) Simple, small, familiar :

Java is a simple small language. So it is very easy to learn. But programming in java is easier than C++ because it does not use header files, Pointers, operator over loading and virtual base classes.

ii) object oriented :

Java is a pure object oriented language. Every thing in java is an object. All programs and data reside inside objects and classes.

iii) Distributed :

This facility helps the users from different places to work together on a single application.

iv) Robust :

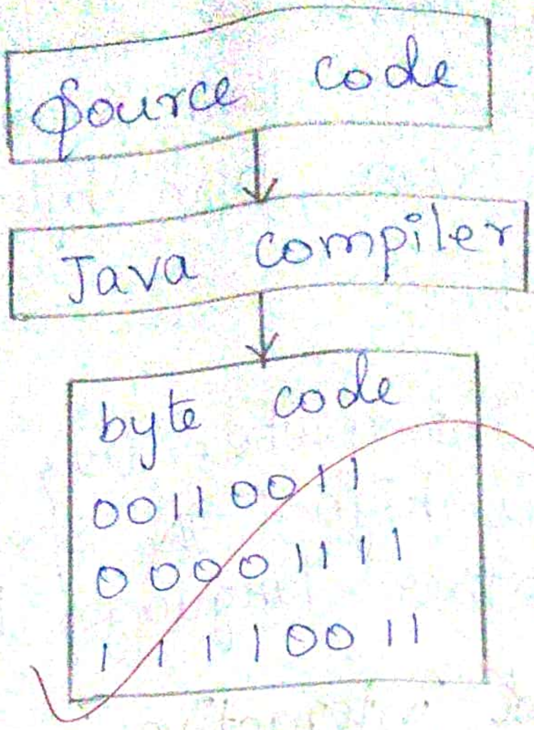
* Garbage collection is used to free the objects which are not in use.
* Exception handling technique is used to to avoid abnormal situation.

v) Secure :

Java is used for programming on internet, security becomes an important issue. Before a java code from internet is interpreted, a security check is applied on it. This ensures that the java codes does not contain any unwanted elements like viruses. digitally signing method is used to secure the code.

vi) Architecture neutral or platform independent.

Java compiler generates an architecture neutral or platform independent code called byte code. Java code can be run in any type of system.

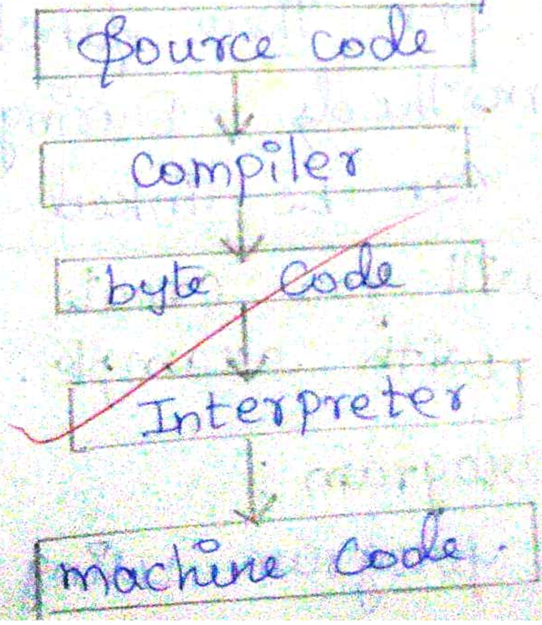


vii) Portable :

Java compiler generates a code called bytecode and this code can be used by any machine. So java is a portable language.

viii) compiled and interpreted :

Generally computer languages are either compiled or interpreted. But java combines both compiler and interpreter



Java compiler generates a machine independent code called byte code. Java interpreter generates machine code from bytecode.

ix) High performance:

Since java interpreter uses bytecodes, the performance is high. The speed is also comparable to other languages like C, C++.

x) Multithreaded and interactive:

Multithreaded means handling more than one job at a time. Java supports multithreading. Java also supports constructing interactive programs.

xi) Dynamic and extensible:

Java is a dynamic language. Java also supports functions written in other languages such as C and C++. These functions are called native methods. During run time native methods can be linked dynamically.

4) Explain how will you create and execute a java program with example.

i) Creating the program:

Create the program using any text

editor such as edit in Dos or notepad or wordpad etc, and save it in the java directory.

```
Class name . java
```

ii) Compiling the program:

Compile the created program using java compiler.

```
javac classname . java
```

java compiler creates a file called class file which contains bytecodes.

iii) Running the program:

Run the compiled program using java interpreter.

```
java classname
```

Java interpreter produces machine code from the byte code.

Eg.
```
class Sum
{
Public static void main (String args[])
{
int a = 40;
int b = 30;
int c = a+b;
System. out. println ("sum = "+c);
}
}
```
Save - Sum. java
compile - javac Sum. java
run - java Sum.

Q) Explain about java token:

A token is an individual element in java. More than one token can appear in single line. Each token must be separated by white spaces. white space may be blank, carriage return or tab. The various java tokens are

- * Keywords
- * Identifiers
- * Constants or literals
- * operators
- * Separators

Keywords:

Keywords are words which belong to java language. They hava standard predefined meaning. These words should be used only for their intended purpose. The users have no right to change its meaning. Keywords Should be written in lowercase.

Eg :- for, if, int, switch

Identifiers:

Identifiers are names given to classes methods, variables, objects, arrays, Packages' and interfaces in a program. These are user defined names.

Rules :

1) Identifiers are formed with alphabet, digits, underscore and doller sign characters.

2) The first character must be an alphabet

3) They can be of any length.

4) They are case sensitive.

Eg : temp

Java literals :

Literals are names that are used to represent constants. There are five types of literals. They are,

* integer literals

* floating point literals

* Character literals

* string literals

* boolean literals.

Separator :

Separators are special symbols which belongs to java language. These are used to indicate where the group of java codes are divided and arranged.

Eg: (), {}, [], ;, , .

6) Explain command line argument.

Command line arguments are parameters that are passed to the program from the command line. These are passed at the time of running the program. The general form is

```
C > java classname arguments to be
   Passed
```

The structure of main method is

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

String args [] is an array of string objects and is empty. Any argument passed through command line is stored in the array. We can access the array elements and use them in the program.

Eg:

```
Class Command 1
{
Public Static Void main (String args [])
{
System.out.println ("Total number of arguments:"
                     + args.length);
for (int i = 0 ; i < args.length ; i++)
System.out.println (args [i]);
}
}
```

Output :-

```
D:\jdk1.7\bin>java Command1 name
atchaya desi bhuvana dhanam
total number of arguments : 5
name
atchaya
desi
bhuvana
dhanam
```

20/20

K. Dhl

2/11/2020

10/01/20

Assignment no : 2

1) Explain about java operator.

An operator is a symbol which represents some operation that can be Performed on data.

- * Arithmetic operators.
- * Relational operators.
- * Logical operators.
- * Short hand assignment operators.
- * Increment and decrement operators.
- * Conditional operators.
- * Bitwise operators.
- * Special operators.

) Arithmetic operators :

Arithmetic operators are used to do arithmetic Calculations. There are two types.

- * Binary operators.
- * Unary operators.

Binary operators:

Binary operators need two operands for operations.

Operand 1	BO	Operand 2

Operator	Operation
+	addition
−	subtraction
*	Multiplication
/	division
% (modulo operator)	reminder after integer division.

Unary operators:

unary operators need only one operand for operation.

Uo operand

operator	operation
—	unary minus
++	increment
−−	decrement

ii) Relational operators:

Relational operators are used to find out the relationship between two Operands.

Operand 1 Ro operand 2

operator	operation
>	greater than
<	less than
>=	greater than equal to
<=	less than equal to
==	equal to
!=	not equal to

iii) Logical operators :

Logical operators are used to find out the relationship between relational expressions.

Operand 1 Lo operand 2

Operator	meaning
&&	AND
||	OR
!	NOT

Logical operators return results as indicated in the following table.

x	y	x&&y	x||y	!x
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

iv) Increment and decrement operators:

1) Increment operator:

++ is the increment operator. This adds 1 to the value contained in the variable.

Variable ++ or ++ Variable

Eg: a++ means a = a+1

2) Decrement operator:

-- is the decrement operator. This subtracts 1 from the value contained in the variable.

Variable -- or -- Variable

Eg: --a means a = a - 1.

v) Short hand assignment operators:

Short-hand assignment operators are operators which are used to simplify the coding of certain type of assignment statement

Variable operator = Expression

operator - +=, -=, *=, /=, %=

Conditional operators :

The Conditional operators ? and : are used to build simple conditional expressio
It has three operands. so it is called ternary operator.

Expression 1 ? Expression 2 : Expression 3;

Expression 1 is evaluated first. If it is True expression 2 is evaluated. If it is false expression 3 is evaluated.

Eg: big = a>b ? a : b;

Bitwise operators :

Operator	meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR
>>	Bitwise right shift.
<<	Bitwise left shift.
~	Bitwise complement.

Special operator :

There are two important special operato
They are.

i) Instance of operator
ii) dot operator (.)

instance of operator is used to find out whether the given object belongs to a particular class or not. The general form is

objectname instance of classname

It gives a true value if the object belongs to the class else false.

Eg: ramu instance of sport

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

Objectname . Variable or method

Eg: ramu. add ()

2) Explain branching and decision making statements

Decision making statements are used to skip or to execute a group of statements based on the result of some condition.

* simple if statement
* if else statement
* else if statement
* nested if ... else

* Switch statement

i) Simple if statement :

```
if (test condition)
{
 Statement block;
}
next statement;
```

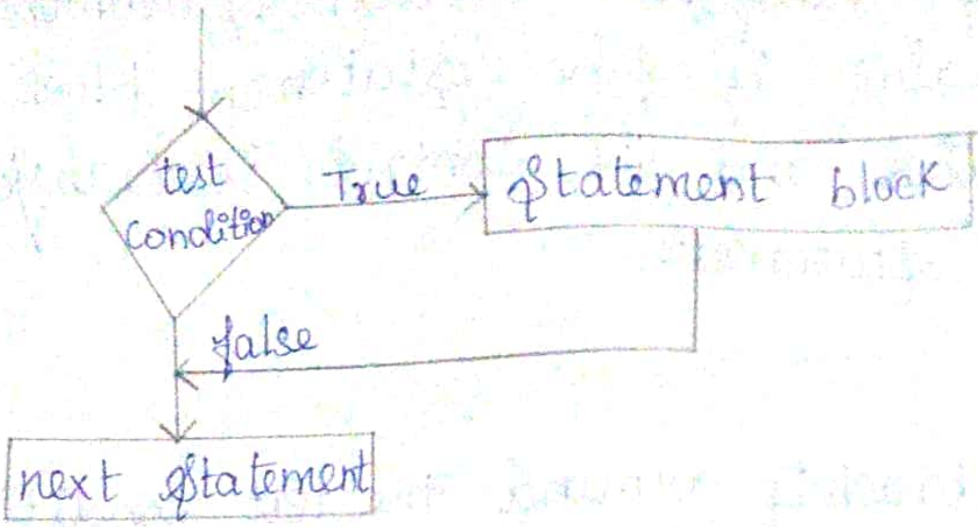
The Computer first evaluates the Value of the test condition. If the value is true, statement block and next statement are executed sequentially. If the Value is false statement block is skipped and Execution starts from next statement.

Rules :

i) The brackets around the test condition are must.

ii) The test condition must be relational or logical expression.

Flow diagram :



Eg1..

```
      m = 60;
   if ( m > = 40)
   {
     System. out. println ("PASS");
   }
```

if ... else statement :

```
if (test condition)
{
 statement block-1 ;
}
else
{
 statement block-2 ;
}
next statement ;
```

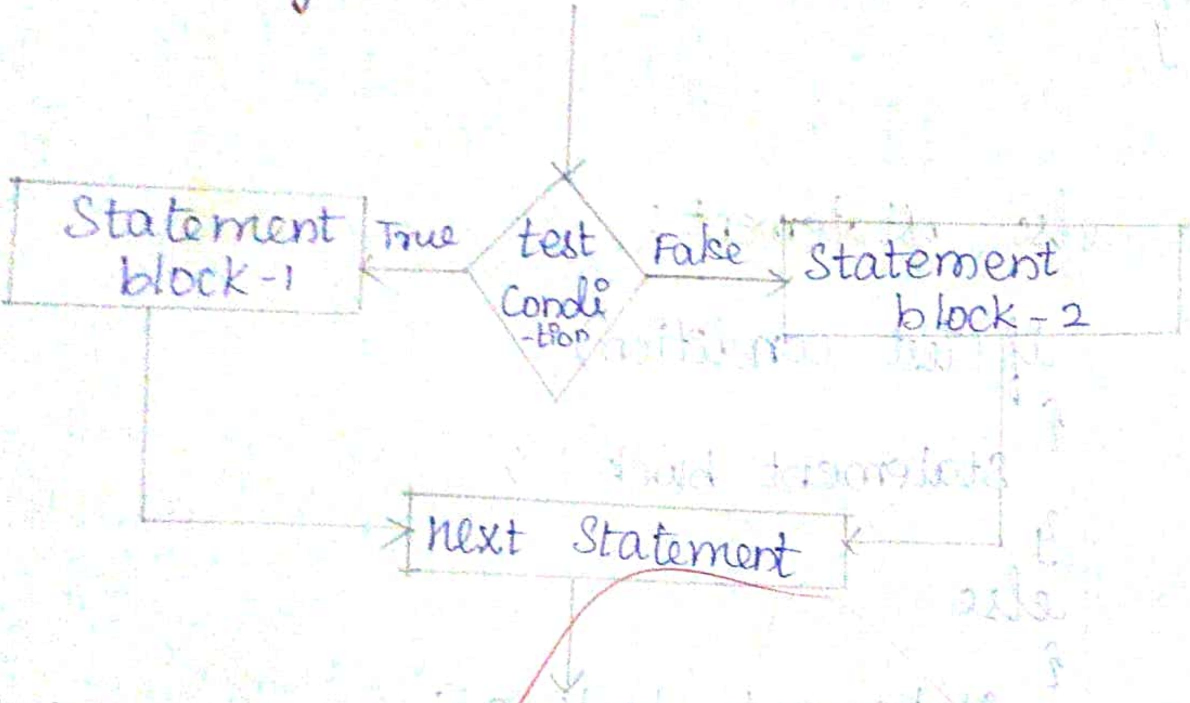
The computer first evaluates the value of th test condition. If the value is true, statement block-1 is executed and the control is transferred to next statement. If the value is false, statement block-2 is executed and the control is transfered to next statement.

Rules:

i) The brackets around the test condition are must.

ii) The test condition must be relational or logical expression.

Flow diagram:



Eg:

```
m = 60;
if (m > = 40)
{
    System.out.println ("PASS");
}
```

```
else
{
System.out.println("FAIL");
}
```

Nested if- else statement:

```
if (test condition-1)
{
if (test condition-2)
{
Statement block-1;
}
else
{
Statement block-2;
}
}
else
{
Statement block-3;
}
next statement;
```

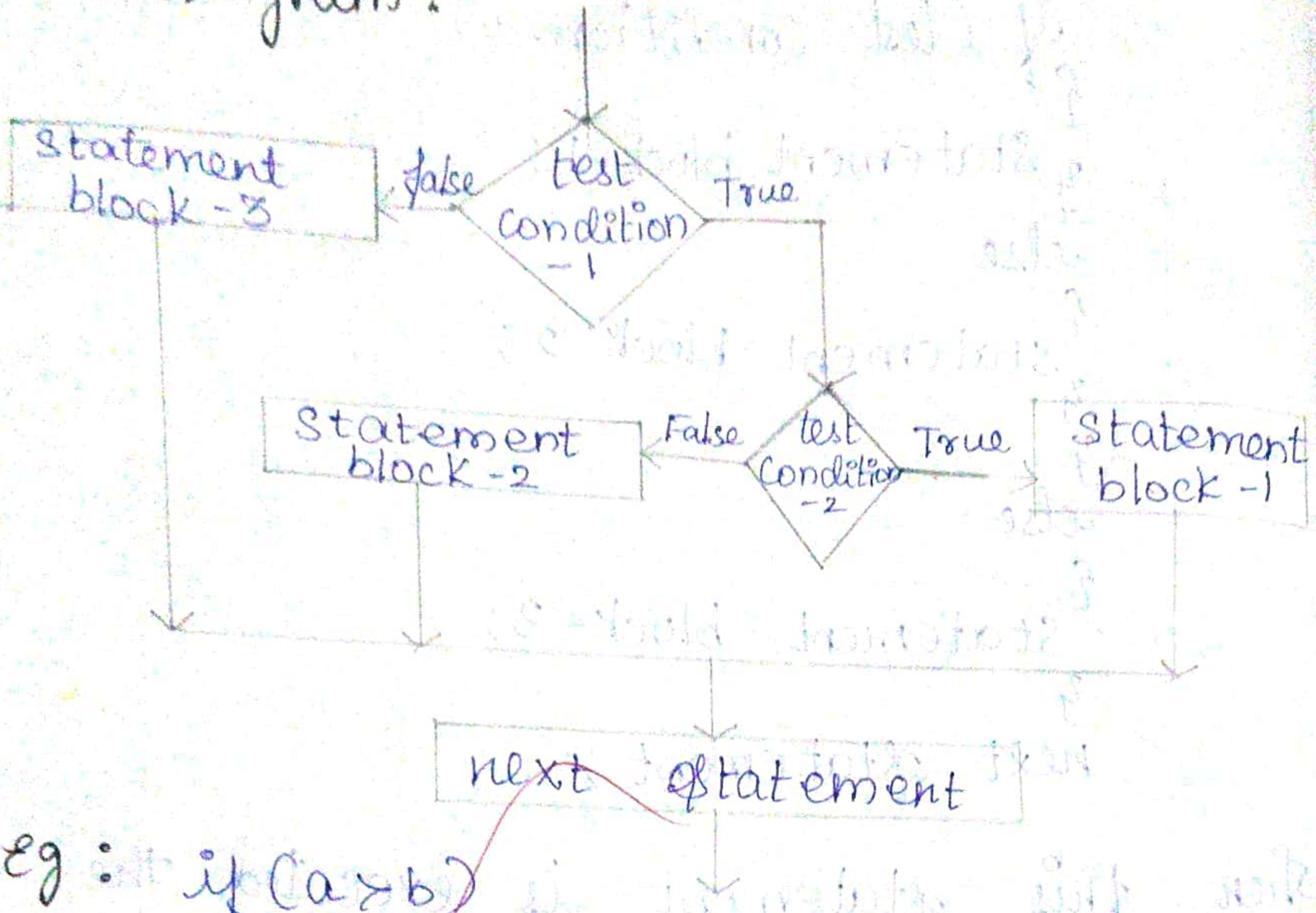
When this statement is executed the computer first evaluates the value of test Condition-1. If it is false control will be transferred to Statement block-3. If it is true test condition is evaluated. If it is true statement block-1 is executed and Control is transferred to next statement else statement block-2 is executed and control is transferred to next statement

Rules :

i) The brackets around the test condition are must.

ii) The test conditions must be relational or logical expression.

Flow diagram:



Eg :
```
if (a>b)
{
 if (a>c)
System.out.println("big="+a);
else
System.out.println("big="+c);
else
{
 if (b>c)
 System.out.println ("big="+b);
 else
 System.out.println ("big="+c);
}
}
```

iv) Else ... if ladder statement :

```
if (test condition-1)
{
 statement block-1;
}
else if (test condition-2)
{
 Statement block-2;
}
- - - - - - - - - -
- - - - - - - - - -
else if (test condition-n)
{
 statement block-n;
}
else
default statement;
next statement;
```

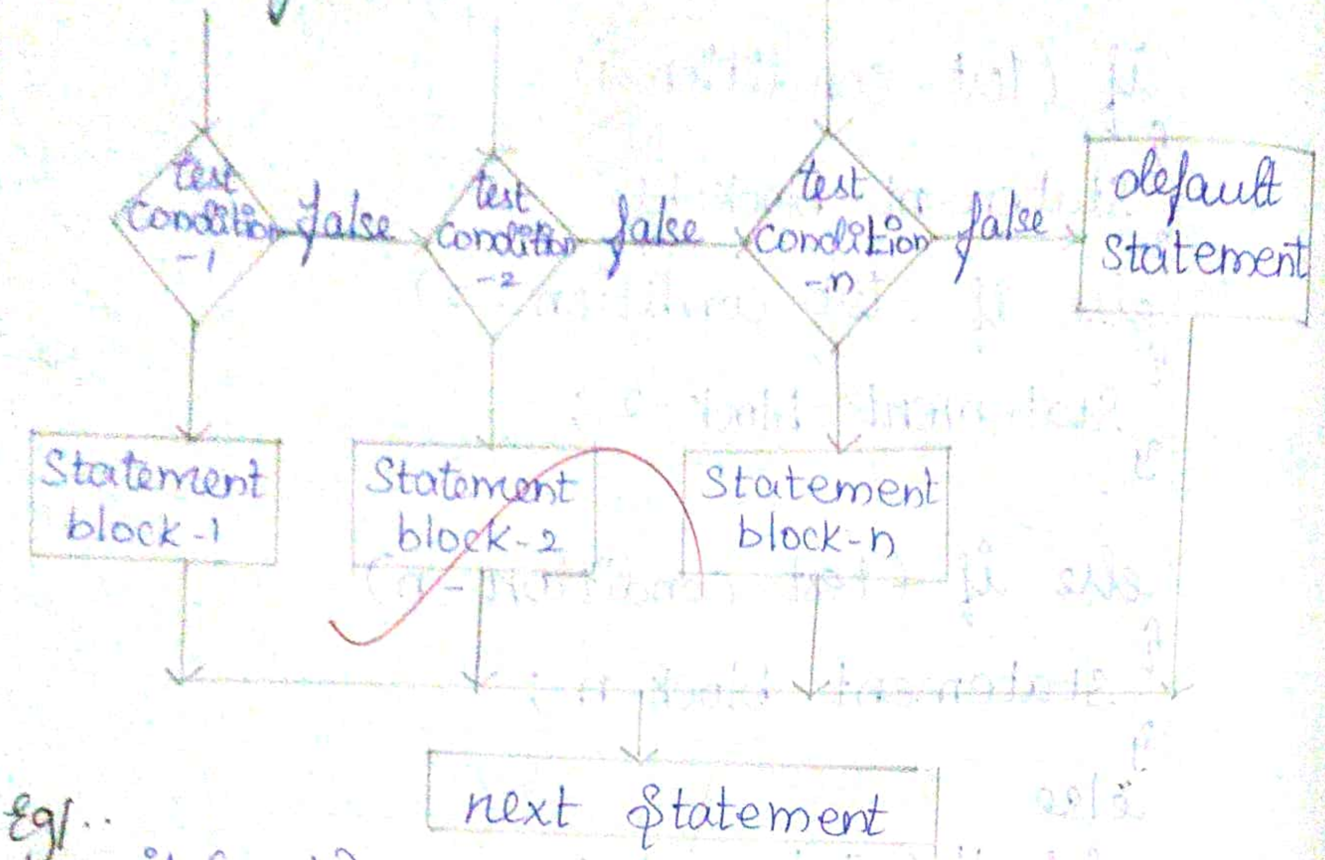
Computer executes this statement from top to bottom. If a true test Condition is found, the statement block associated with it is executed. when all the test Conditions are false, then the final else Containing, the default statement will be executed.

Rules :

i) The brackets around the test Conditio are must.

ii) The test conditions must be relationa or logical expression.

Flow diagram :



Eg/..

```
if (a>b)
System. out. println ("a is larger");
else if (a < b)
System. out. println ("b is larger");
else
System. out. println ("a&b are equal");
```

v) switch statement :

```
switch (expression)
{
case label 1 :
              statement block-1;
              break;
Case label 2 :
              statement block-2;
              break;
 - - - - - - - - - - - - -
 - - - - - - - - - - - - -
case label n :
              Statement block-n;
              break;
default :
              default statement;
              break;
}
next statement;
```

when this statement is executed the Computer first evaluates the value of expression in the keyword switch. This value is successively compared with the case label 1, label 2, ... label n. If a case lable matches with the value, the statement block associated with the case label is executed. Then the control is transferred to the next statement.

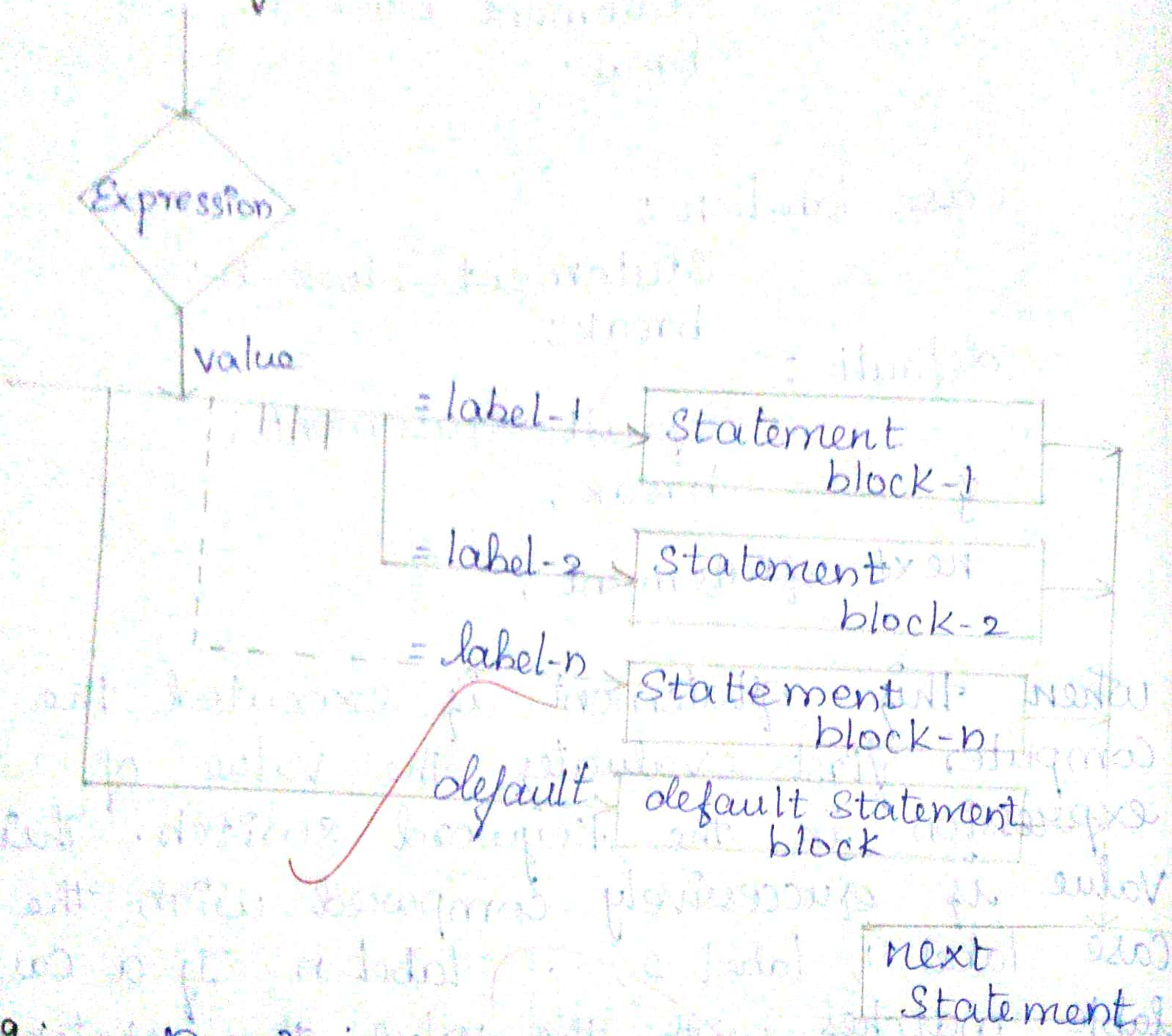
If none of the case matches with the value, the default statement block is executed.

Rules :

i) The expression should be placed in Parenthesis.

ii) Case label should terminate with a Colon.

Flow diagram :



Eg : n = 2 ;

```
Switch (n)
{
case 1 : System.out.println ("ONE");
         break;
Case 2 : System.out.println("TWO");
         break;
Case 3 : System.out.println("THREE");
         break;
default : System.out.println("Invalid number");
         break;
}
```

3) Explain looping statements.

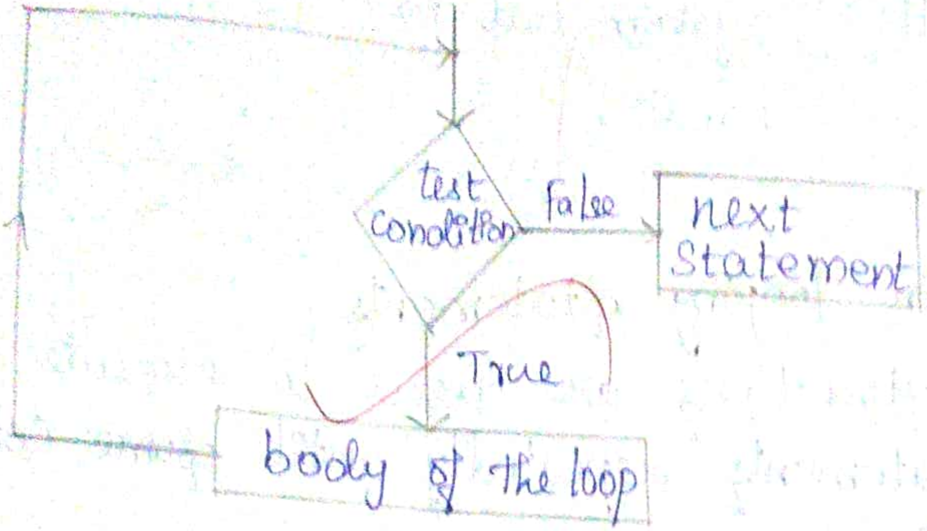
Loop structures are used to execute a group of statements repeatedly until some condition is satisfied.

- * while structure.
- * do... while structure.
- * for structure.

i) While structure :

```
while (test condition)
{
    body of the loop;
}
next statement;
```

when this statement is executed, the Computer first evaluates the test condition. If the value is false, the control is tranyferred to next gstatement. If the value is true, then the body of the loop is executed repeatedly until the test condition becomes false.



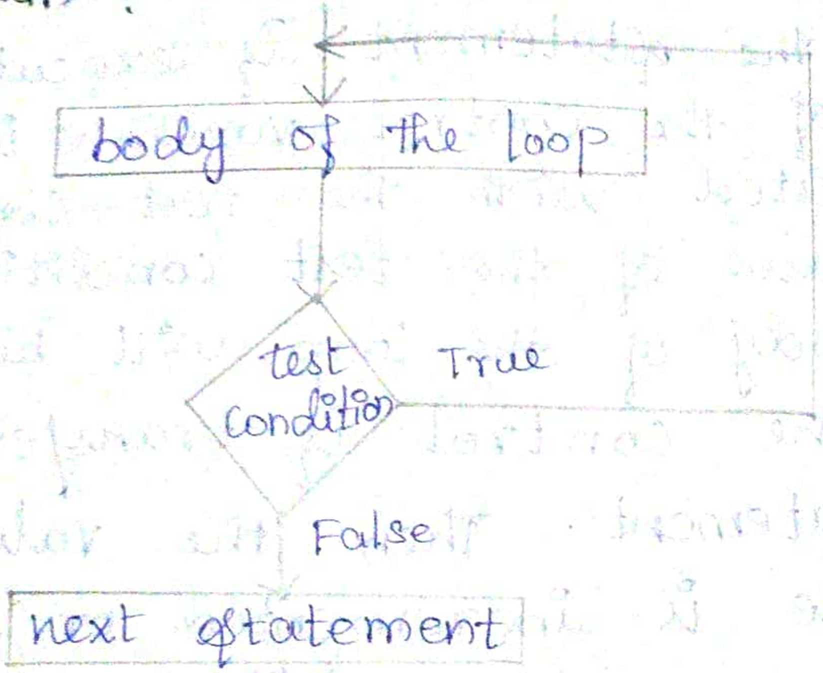
Eg/..
```
  i=1;
 while (i<5)
 {
  System.out.println(i);
  i++;
 }
```

do-while Structure:

```
do
{
body of the loop;
}
while (test condition);
next gstatement;
```

when this statement is executed the body of the loop is executed first. Then the test condition is evaluated. If the value is false, the control is transferred to the next statement. If the value is true the body of the loop is executed repeatedly until the test condition becomes false.

flow diagram :



Eg:-

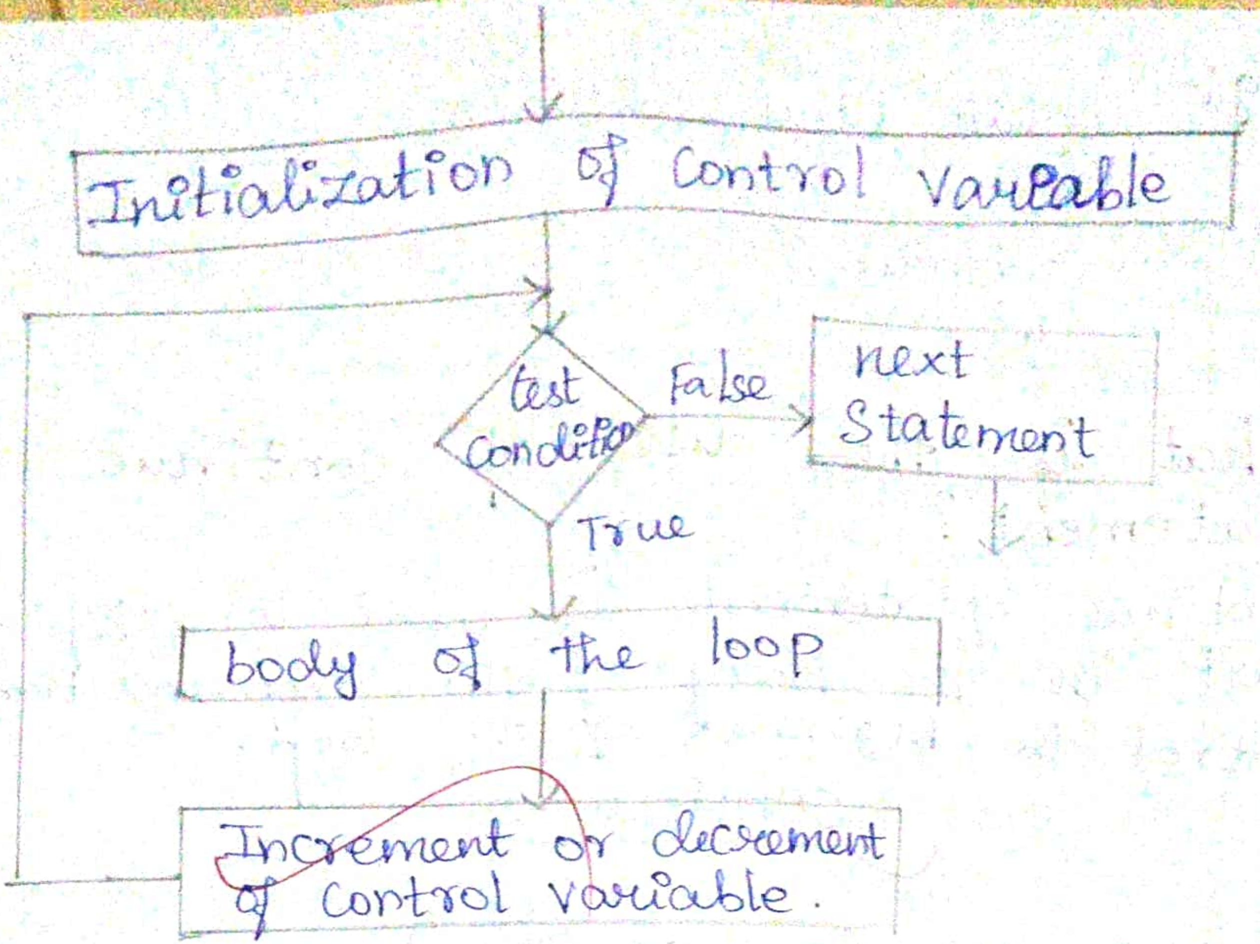
```
i=1;
do
{
 System.out.println(i);
 i++;
}
while(i<5);
```

iii) for statement :

```
for (control variable ; test condition ; increment
        or decrement)
 {
     body of the loop ;
 }
  next statement ;
```

when the statement is executed the value of the control variable is initialized and tested with the test condition. If the value of the test condition is True the body of the loop will be executed and the control is transferred to the for statement. Then the value of control variable is incremented or decremented. when the test condition becomes false the control is transferred to the next statement.

Flow diagram :


Initialization of Control Variable
test Condition
False
next Statement
True
body of the loop
Increment or decreament of control variable.

Eg].

```
for(i=1; i<=10; i++)
{
System.out.println(i);
}
```

1. Define Array. Explain how will you create one dimensional auay.

An Array is a group of same type of data items that are referred by a common name.

One - dimensional Arrays

An array with one subscript is called one dimensional auay.

Creating an array

An array must be declared and created before it is used. The steps to be followed are,

Step 1 : Declare the array.
Step 2 : create memory space.
Step 3 : Store data into the created memory space.

Step 1 - Declaration of arrays

There are two methods to declare an array in java

method - 1

```
datatype arrayname [];
```

method - 2

```
datatype[] array name;
```

eg : int x[];

Step 2 - Creation of arrays (Memory space)

After declaring an auay, we must allocate memory spaces for the declared auay. This is done with the help of new operator.

```
arrayname = new datatype[size];
```

void print2()

Step 3 - Storing values during declaration or Initia

we can store values at the time of declaration. The compiler allocates the required depending upon the list of values.

```
datatype arrayname[] = {value1, value2, ... valuen};
```

eg: int a[] = {10, 20, 15, 33, 25};

2) Explain about vectors.

Vector is a class available in the java.util package. It is used to create variable size dynamic array for storing different objects.

Vector creation

Vectors can be created in three ways. They are

i) creating without size
ii) creating with size
iii) Creating with size and increment.

i) Creating without size

The general form is,

```
Vector vectorname = new vector();
```

eg: Vector A = new vector();

ii) creating with size

The general form is,

```
Vector vectorname = new vector(n);
```

eg: Vector A = new Vector(10);

iii) Creating with size and increment

The general form is

```
Vector vectorname = new vector (n, increment);
```

This creates a vector size n and is

the value of increment

portant methods.

Method	use	example
1) capacity() Syntax Vectorobject.capacity();	it returns the maximum number of elements to be stored in the vector	V1. capacity();
2) size()	it returns the actual number of elements in the vector	V1. size();
3) addElement(object)	it adds the object at the end of the vector	V1. addElement("CE");
4) insertElementAt(object, n)	it insert the given object in nth position	V1. insertElementAt("ECE", 2);
5) removeAllElements()	it remove all the elements in the vector	V1. removeAllElement();
6) removeElementAt(n)	it remove nth element of the vector	V1. removeElementAt(2);
7) removeElement(object)	it remove the given element	V1. removeElement("CE");
8) ElementAt(n)	it returns nth element	V1. elementAt(2);

3) Differences between Array and ArrayList.

Array	ArrayList
1) Arrays ~~are only~~ size is ~~fixed~~ fixed ~~size~~.	Array list size is not fixed.
2) Array has no methods to insert and delete elements. So insertions and deletions are difficult.	Array List has methods to insert and delete elements. So insertion and deletion can be done easily.
3) Array accept only same	Array list accept any

4) Explain about Arraylist.

Array list is a class available in the java.u
This is used to create dynamic array ~~of variable size~~ to s
different objects.

Array list creation

i) creating without size

ii) creating with size

i) creating without size.

```
ArrayList name = new ArrayList();
```

eg: ArrayList A = new ArrayList();

ii) creating with size

```
ArrayList name = new ArrayList(n);
```

eg: ArrayList A = new ArrayList(10);

Important methods.

1) add (n, object)

Inserts the ~~given~~ object in the given position.

eg: ~~a1. (2, "CE");~~ a1. add (2, "CE");

2) add (object)

~~Appends~~ Add the object at the end of the list.

eg: ~~a1. ("ECE");~~ a1. add ("ECE");

3) clear()

Removes all the element from the list.

eg: a1. clear();

4) ~~Object~~ get (n)

Returns the element in ~~from~~ nth ~~the~~ specified ~~index~~ position of the list.

eg: a1. get (2);

5) remove (n)

Removes the element at nth position ~~from the given index~~.

eg: a1. remove (3);

6) size()

Returns the number of elements in the list.

nat is wrapper class? Explain. 3

wrapper classes are used to convert primitive data types into object type and object ~~vice versa~~ to primitive data type.
These classes are available in java.lang package.

primitive	Wrapper
boolean	Boolean
byte	Byte
Short	Short
char	Character
int	Integer
float	Float
long	Long
double	Double

Uses of wrapper class.

1. Converting primitive number into number object.
2. Converting object numbers into primitive numbers.
3. Converting primitive numbers to String object.
4. Converting String object to number object.
5. Converting numeric string object to primitive numbers

① Converting primitive datatype to Object

Constructor is used to Convert Primitive datatype to object.

classname objectname = new classname(value);

(2) converting object to primitive datatype

The typeValue() method is used to convert object to primitive datatype.

datatype variablename = objectname. typeValue();

eg. convert float object to float datatype -

float s1 = f1. floatValue();

(3) Converting primitive number to stringobject

The toString() method is used to convert primitive datatype to string

String objectname = classname. toString(value);

eg. Convert int to string object

String s1 = Integer. toString (67);

(4) Converting string object to primitive number

The parsetype() method is used to convert to string object to primitive number

datatype variable = classname. parsename(string object);

eg. Convert string to int

int k2 = Integer. parseInt (s1);

(5) Converting string object to number object

The valueOf() method is used to convert string object to number object.

= classname. valueOf (string).

III UNIT

Explain about string.

String is a sequence of characters enclosed within double quotes. In java strings are treated as objects of the class String. This class present in the package java.lang.

Created string objects are unchangeable.

String creation

i) creating a empty string

```
String name = new String();
```

eg: `String S1 = new String();`

ii) creating string with characters

```
String name = new String(value);
```

eg: `String S2 = new String ("good");`

iii) creating a string with another string

```
String name = new string (string object);
```

eg: `String S3 = new String (S2);`

iv) creating a string using substring

```
String name = new String (string, m, n);
```

m - starting position

n - number of characters

eg:

```
Char a[] = {'p', 'r', 'o', 'g'}
String S4 = new String (x, 2, 3);
```

(S4 = rog)

Methods

```
stringobject. methodname();
```

`String S1 = new String ("Example");`

Method	use	Exam
length()	This method is used to find the length of the String.	S1. length(

i) ~~a~~ length()

This method is used to find the length of the string.

```
Stringobject. length();
```

eg: S1.length();

ii) toLowercase()

This method is used to convert the string ~~in~~ ~~upper case letter~~ to lower case letters.

```
Stringobject. toLowercase();
```

eg: S1. toLowercase();

iii) touppercase()

This method is used to convert the string ~~to~~ ~~lower case letters~~ to upper case. letters.

```
Stringobject. touppercase();
```

eg: S1. touppercase();

iv) trim()

This method is used to remove the leading and trailing blank spaces in a string.

```
Stringobject . trim();
```

eg: S1.trim();

v) concat (S1)

This method is used to join the calling string with ~~string~~.S1.

```
Stringobject. concat (S1);
```

S2. concat (S1);

vi) equals (S1)

This method is used to find whether the calling

```
Stringobject1.equals(s1);
```

eg: s2.equals(s1);

(i) substring(n)

This method is used to find a substring starting from nth character.

```
stringobject.substring(n);
```

eg: s1.substring(3);

viii) indexof(ch)

This method is used to find the index of the first occurrence of the character ch in the calling string.

```
stringobject.indexof(ch);
```

eg: s1.indexof('a');

2) Explain about stringbuffer class.

Stringbuffer is a class present in the package java.lang. This class is used to create strings of variable length. That is, the created object content of stringbuffer class can be modified.

Stringbuffer creation

i) creating a empty string buffer

```
StringBuffer name = new StringBuffer();
```

eg: StringBuffer n1 = new StringBuffer();

This creates a empty string buffer b with initial capacity of 16 characters.

ii) creating a stringbuffer of size n

```
StringBuffer name = new StringBuffer(n);
```

eg: StringBuffer n2 = new StringBuffer(20);

(iii) creating a stringbuffer with initial value.

Methods

```
Stringbufferobject. methodname();
```

StringBuffer b1 = new StringBuffer ("college");

i) length()

This method is used to find the current length of the stringbuffer.

```
Stringbufferobject.length();
```

eg: b1.length();

ii) capacity()

This method is used to find the maximum length of the stringbuffer.

```
Stringbufferobject.capacity();
```

eg: b1.capacity();

iii) setLength()

This method is used to set new length for the string buffer object.

```
Stringbufferobject.setLength(1);
```

eg: b1. setLength(1);

iv) SetcharAt(n, ch)

This method is used to change the character at the specified index by the new character.

```
stringobject.setcharAt(n, ch);
```

eg: b1.SetcharAt(5, "E");

v) insert(n, s1)

This method is used to insert a new string in the specified index.

```
Stringobject.insert(n, s1);
```

eg: b1. insert(0, "adj");

vi) append(s1)

This method is used to join a new string at the end of the calling string buffer.

reverse()

This method is used to reverse the calling String buffer object.

```
Stringbufferobject . reverse();
```

eg: b1. reverse( ) ;

3) Explain about class and object.

A class is a user defined data type. It contains data and its related methods.

```
class classname
{
   datatype variable1;
   - - - - -
   datatype variablen;
   datatype methodname (parameters)
   {  Statements
   }
   - - -
}
```

Rules

1) The field and methods defined inside a class are called instance variables and instance methods.
2) The fields and methods declared within a class are called members of a class.

eg :
```
class Point
{
   int x, y;
   void main (int a, int b)
   {  x = a;
      y = b;
   }
   void display ()
   {
```

object - class type variable is called

Objects are created from the defined cla a class we can create any number of objects. A created objects can use the instance variables. Memo space for the instance variables will be allocated on during object creation.

Creating object.

1) Declare the object

```
Classname object 1, object 2 --- object n ;
```

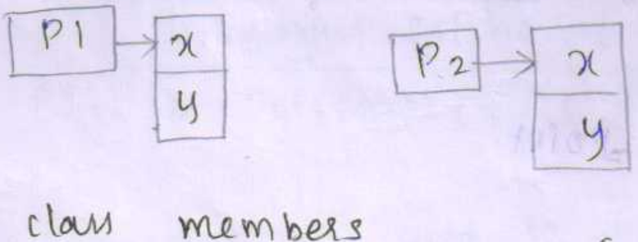
eg: Point p1, p2 ;

2) Create memory space

After object declaration, we must allocate memory space for the instance variables for each object. This is done with the help of new operator.

```
object1 = new classname();
Object 2 = new classname();
- - - - - -
- - - - - -
object n = new classname();
```

eg: p1 = new point();
p2 = new point();



Accessing class members

The members of class are accessed using object.

```
object . variable
object . methodname();
```

eg: p1. x = 10; p1 . display ();

A) Explain about constructor.

constructor is a special method in the class.

1) Default constructor or constructors without arguments.

2) parameterised constructor or constructor with arguments.

```
Constructorname (arguments)
{
    Statements
}
```

eg:

```
point()
{
  x = 40;
  y = 60;
}
```

```
class point
{ int x, y;
  Point()
  { x = 40;
    y = 60;
} }
```

Rules.

1. The name of the constructor is same as its class name.
2. The constructor can not be called explicitly.
3. The constructor has no return type.
4. If the constructor has no arguments then that constructor is called constructor without arguments. If it has arguments, then it is called constructor with arguments.

eg:

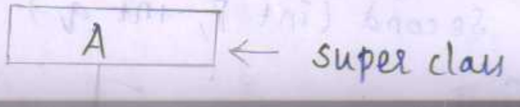
```
class point
①{ int x, y;
  Point()
  {
    x = 40;
    y = 50;
  }
}
```

```
class point
{ int x, y;
② Point (int a, int b)
  { x = a;
    y = b;
} }
```

7) Explain about inheritance or Explain the different types of inheritance.

Inheritance is the process of creating new classes from the existing classes. The new classes are called super classes. The existing classes are called sub classes.

The derived classes inherit all the properties of the base classes plus its own properties.



ii) It reduces program coding time.

iii) It increases the reliability of the program

Types of inheritance

i) Single inheritance.

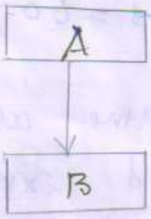
ii) Hierarchical inheritance.

iii) Multilevel inheritance.

iv) Multiple inheritance.

i) Single inheritance

A class derived from one superclass is called single inheritance.



```
Class name 2 extends name 1
{
   - - -
   - - -
}
```

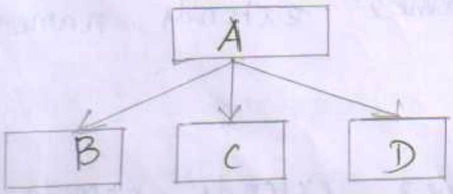
Where
Class, extends - Keyword
name 2 - subclass
name 1 - superclass.

eg:
```
class First
{
  int x;
  First (int a)
  {
    x=a;
  }
void print()
{
  System.out.println(x);
}
}
class Second extends First
{
  int y;
  Second (int p, int q)
{
```

```
void print1()
{
  System.out.println(y);
 }
}
class Sinheritance
{
Public static void main (String args[])
 {
   Second S1 = new Second (5,20);
    S1.print();
    S1.print1();
  }
 }
```

ii) Hierarchical inheritance

More classes derived from one super class is called hierarchical inheritance.



```
Class name2 extends name1
{
  ---
}
Class name3 extends name1
{
  ---
  --
}
--
Class namen extends name1
{ ---
}
```

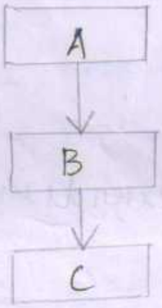
```
Clan First
{
  int x;
  First (int a)
  {
   x = a;
  }
  void print()
  {
   System.out.println(x);
  }
```

```
  {
   super(p);
   y = q;
  }
  void print1()
  {
    System.out.println(y);
  }
}
Class Third extends First
 { int k;
   Third (int p, int q)
```

```
Void print2()
{
  System.out.println (K);
 }
}
class Hinheritance
{
 public static void main (String args[])
 {
  Second s1 = new Second (10, 20);
  Third t1 = new Third (40, 60);
     s1. print1();
     t1. print2();
     s1. print();
  }
 }
```

iii Multilevel inheritance

A class derived from other derived class is called multilevel inheritance.



```
class  name2  extends  name 1
{
  ----
}
class  name 3  extends  name 2
{ ----
}
 ---
class  namen  extends  namen-1
{
 ---
}
```

```
Class First
{
 int x;
 First (int a)
 {
  x=a;
 }
```

5/ Explain 1) constructor overloading 2) Nesting of methods 3) this keyword.

Constructor overloading

If a class contains more than one constructor, then it is known as constructor overloading.

eg

```
Class Point
{
 int x, y;
 point()
 { x = 100;
   y = 40;
 }
 point(int a, int b)
 {
   x = a;
   y = b;
 }
}
```

eg2

```
Class Sum
{
  int x, y;
  Sum()
  {
   x = 10;
   y = 20;
  }
  int total()
  { return (x+y);
  }
  Void display()
  {
   S.O.P ("Sum =" +
            total());
  }
}
```

2) Nesting of methods

If a method calls another method in the same class, then it is called nesting of methods.

eg2 : In the above example, display() method calls total() method.

3) <u>this keyword</u> : It is a implicit p

It contains the address of the objec which Calls the method. It is used to access the hidden class instance Variable.

eg

```
Class Example
{
   int x=10;
   Void display()
   {
      int x = 40;
      S.O.P ("x =" +x);
      S.O.P ("x =" + this.x);
   }
}
```

6. Explain about static members.

The variable or method in a class ~~is~~ ~~ate~~ are declared with keyword static, then it is known as static member.

```
Class name
{
 static datatype Variable;
 . . . . . .
 . . . . . .
 . . . . .
 static datatype methodname()
 {
   . . . .
   . . . .
   . . .
 }
}
```

eg

```
Class Ex1
{
 static int x=30;
 static void
        print()
   {
    S.O.P("x ="+x);
   }
}
```

es

1) static method can be called by using Classname

```
Classname . staticmethodname ( );
```

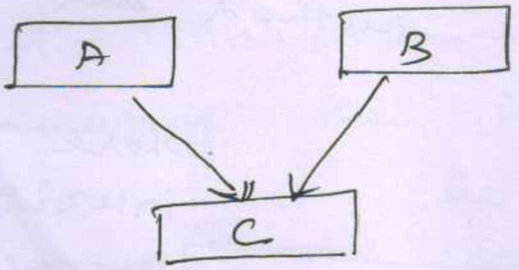
2) static method use static Variable only.

3) For static Variable, common memory is allocated

4) this or super keyword cannot be used with static members.

7) Multiple inheritance

A class is derived from more than one superclass is called multiple inheritance. It is implemented using interface.

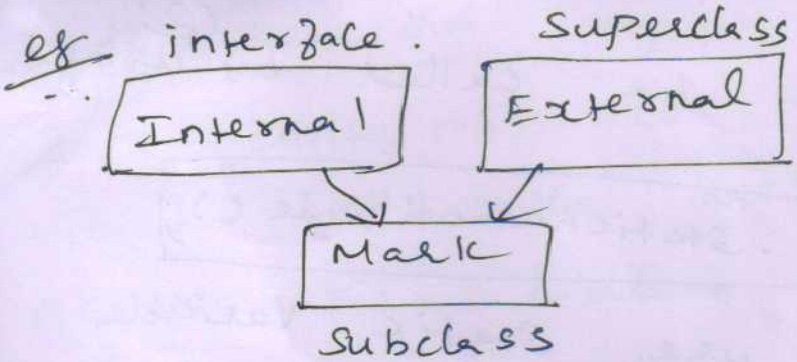


name 2 - Subclassname
name 1 - Superclassname
interface1, interface2 . . .
interfacen — name of the interface
class, extends, implements — keywords.

Syntax

```
class name2 extends name1 implements interface1,
                         interface2, . . . . interfacen.
  {
      . . . . .
      . - . ; . . .
      . - - - .
  }
```

eg



```
interface Internal
{
  . . . . .
  . . . . .
}
class External
{
  . . . . .

  . . . .

}
```

```
Class Mark exte
External implements Int
{ . . .
  . . .
}
Class Minheritance
{
 P S V m ( )
 {
 Mark M1 = new Mark
              ( );
  . - - -
  - - - .
 }
}
```

8. Define interface. Explain how it is created, extended and implemented with example.

Interface is a user defined datatype. It contains abstract method and final variables.

Syntax

```
interface name
{
final datatype Variable = Value;
. . . . .
. . . . .
abstract datatype methodname();
. . .
   . . . .
}
```

where interface, final, abtract - keys words
name - interface name.

```
interface Area
{
 float pi = 3.14f;
 abstract void display();
}
```

eg 2

```
interface Y extends Area
{ .....
  .....
}
```

extending interface

Deriving a new interface from the existing interfaces is called extending interface

```
interface name2 extends interface1, interface2
                        .... interfacen
{
  ....
  ....
}
```

eg2

implementing interface

The process of using the already defined interface in a class is called implementing the interface. The methods of interface must be defined in the class.

syntax

```
class name1 implements interface1, interface2
                        .....
{
  ....
  .....
}
```

eg

```
interface Area
{
  float pi = 3.14f;
```

```
abstract void display()
} {
class circle implements Area
{ int r;
  circle (int a)
  { r = a;
  }
  public void display()
  {
    S.O.P("Area =" + (pi*r*r));
  }
}
```

9) Explain about final variable, final method and final class.

__final variable__ : If a variable in a class is declared with the keyword final, then it is called final variable. The value of final variable cannot be changed.

final datatype variable = value;

__eg__ final int x= 20;

__final method__ : If a method in a class is declared with the keyword final, then it is called final method. final method cannot be override in the subclass.

```
final datatype methodname()
   {
       - - - -
       - - - -
   }
```

eg:

```
final void display()
   {
     S.O.P(x);
   }
```

final class: If a class is declared with a keyword final, then it is called final Class. Subclass cannot be created from final class.

```
final class name
{
   .....
   .....
}
```

eg:

```
final class point
   {
     ....
     ----
   }
```

(10) Explain about abstract method & abstract class.

abstract method: If a method is declared with keyword abstract, then it is called abstract method. The abstract method has no definition. It must be override in the subclass.

```
abstract datatype methodname();
```

eg: abstract void print();

abstract class: If a class is declared with a keyword abstract, then it is

Called abstract class. Object cannot be created from the abstract class.

```
abstract class name
 {. . . . .
   . . . .

 }
```

```
abstract class X1
{
  . . . . . .
      . . . .
}
```

11) Write short notes on overriding.

The process of redefining the method in the super class is called overriding. To call the method in the superclass, the keyword super is used.

ef

12) Explain about visibility control.

pg 3.82.

The default modifier is friendly.

Syntax

```
accessmodifier datatype variable;
access modifier datatype methodname()
 {. . . . .
   . . . . . .
 }
```

	Private	Public	Protected	friendly	Private Protected.
Same class	yes	Yes	Yes	Yes	Yes
Subclass in the same package.	NO	Yes	Yes	Yes	Yes
Other class in the same package	NO	Yes	Yes	Yes	NO
Subclass in other package	NO	Yes	Yes	NO	Yes
Other class in other package	NO	Yes	NO	NO	NO.

UNIT - V

xplain Exception handling.

Exceptions are errors occurring at run time of a program. Exception handling is a technique to deal with the exceptions. In java, exception handing is done with the help of exception objects.

Advantages of exception handling.

i) It avoids abnormal situation during runtime.

ii) It helps the user by reporting the reason for abnormal situation.

Exception types

There are different types of exception classes for handling various errors.

Name	Description
ArithmeticException	This is used find the arithmetic error such as divide by zero.
ArrayIndexOutofBounds Exception	This is used to find the array index which exceeds the index limit.
ClassNotFoundException	This is used to find whether the class is defined or not.
IOException	This is used to find the I/o failure such as inability to read from a file.
EOFException	This is used to find the end of file.

Basics of exception handling

```
try
{
  Statements for checking the errors
}
catch (Exceptionclass1 obj1)
{
  Statements for handling the error
}
catch (Exceptionclass2  obj2)
{
  Statements for handling the error
}
  - - - - - - -
  - - - - - - -
catch (Exception classn  objn)
{
  Statements for handling the error
}
finally
{
  Statements to be executed before existing
}
```

Try block

This block is used to test the program statements for run time errors. If an error is found, the try block throws the error and is caught by catch blocks. In a program there can be any number of try blocks.

Catching an exception

Catching an exception means, catching the thrown exception objects from try block by the corresponding catch blocks. This block should be written immediately after try block. There can be more than one catch blocks.

finally block

This block is an optional block in exception handli

Example.

```
class EX2
{
public static void main (String args[])
{
int a = 25;
int b=0;
int c;
try
{
  c=a/b;
 System.out. println ( "c = "+c);
}
catch (ArithmeticException e)
{
  System.out. println (" Divide by zero");
}
finally
  {
   System.out. println (" Exception over");
  }
}
}
```

2) Explain about thread methods.

i) run()

This method contains the statements for the particular thread.

```
public void run()
  {
   Statements related to a particular thread
  }
```

ii) start()

This method is used to start the run() method.

iii) sleep()

(iv) ISAlive()

check if the thr

This method is used to ~~stop the running thread~~.

is running or not.

v) Stop()

This method is used to stop the running thread.

vi) yield()

This method is used to bring the stopped thread to run mode.

vii) wait()

This method is used to stop the currently running thread.

3. Explain the lifecycle of Thread with neat diagram.

1. Newborn state
2. Runmode state
3. Running state
4. Blocked state
5. Dead state.

1) Newborn state

At once the thread object is created for the defined thread the new thread is born. This state ~~of the thread~~ is called new born state. From this state the new born thread can go to any one of the following state.

a) runmode state.
b) dead state.

If start() is called it goes to runmode state. If stop() is called it goes to dead state.

2) Runmode State.

If a thread is ready for execution, then the state of the thread is called runmode state.

3) Running state

If a thread is in execution then this state is called running state. This state continues until any one of the following happens.

d) When wait() is called

e) When suspend() is called.

4) Blocked state

A thread becomes blocked state if any one of the following method is called while thread is running.

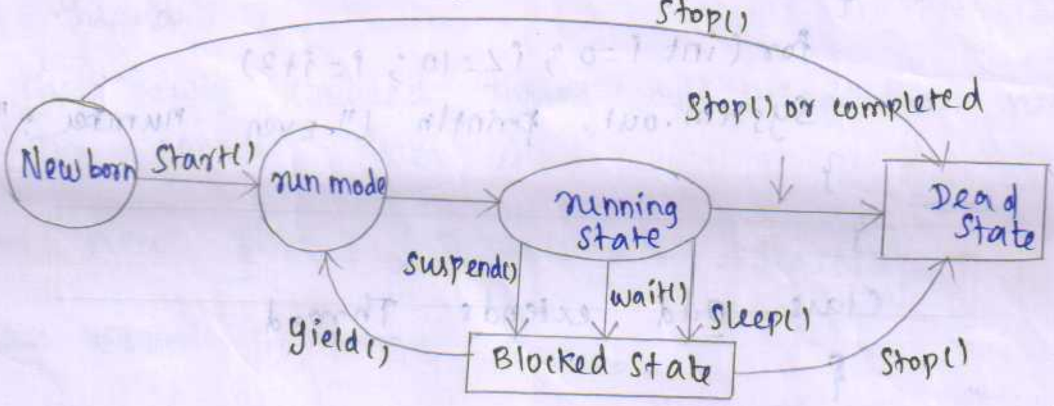
i) Sleep()
ii) wait()
iii) suspend()

From blocked state it comes to runmode state if the yield() method is called.

5) Dead state

A thread is said to be in dead state if any one of the following happens.

a) if it completes its execution - Natural death
b) if it is called by the method stop() - kill



4. Explain about defining and running a thread using Thread class.

There are two methods to create threads. They are

i) creating threads by extending Thread class.
ii) Creating threads by implementing Runnable interface.

Defining and running a thread using Thread class

i) create ~~thread~~ sub classes by extending the super class Thread.
The general form is

```
Class ~~thread~~sub classname extends Thread
{
```

ii) Override the method run() in all extended classes.

```
public void run()
{
    statements for the thread
}
```

iii) write the main class and define thread objects for the created threads.

iv) using the created thread objects start the threads.
The general form is.

```
threadobject . start();
```

eg:

```
Class Even extends Thread
{
  public void run()
  {
   for (int i=0; i<=10; i=i+2)
   System.out. println ("Even number :"+i);
  }
}

Class Odd extends Thread
{
  public void run()
  {
   for (int j=1; j<=10; j=j+2)
   System.out.println ("odd number:"+j);
  }}
Class T1
{
public static void main (String args[])
{
 Even e1 = new Even();
 Odd o1 = new Odd();
 e1.start();
```

efining and running a thread using Runnable interface

i) Create thread sub classes by implementing the interface Runnable.

```
Class subclassname ~~Newthread~~ implements Runnable
{
    - - -
    - - -
    - - -
}
```

ii) Define the method run() in all implemented classes

```
public void run()
{
    - - -
    - -
}
```

iii) write the main class and define thread objects for the created threads.

iv) Start the newly created thread by using the method Start(). The general form is

```
new Thread (threadobject) . Start();
```

Eg:

```
Class Even implements Runnable
{
    Public void run()
    {
        for (int i=0; i<=10; i=i+2)
            System.out.println ("Even no: "+i);
    }
}
Class Odd implements Runnable
{
    Public void run()
    {
    for (int j=1; j<=10; j=j+2)
```

```
class T2
{
  Public static void main (String args[])
  {
   Even e1 = new Even();
   Odd o1 = new odd();
   new Thread (e1).Start();
   new Thread (o1). Start();
  }
}
```

5. Explain Bytestream classes.

Byte stream classes provides facilities to do I/o operations in bytes. There are two abstract classes namely Inputstream and outputstream to do read and write operations.

i) InputStream class

Inputstream is an abstract class. This class contains number of methods to do input operations. If error occurs it throws IOException.

Method	use
1) available()	This method gives the number of ~~Characters~~ bytes currently available for reading.
2) close()	This method is used to close the input stream
3) read()	This method is used to read a ~~character~~ byte. from input stream
4) read (~~character~~ byte b[])	This method is used to read an ~~array~~ array of ~~character~~ bytes
5) read (~~character~~ byte b[], n, m)	This method is used to read m ~~character~~ bytes starting from nth byte.

outputStream classes.

OutputStream is an abstract class. This class contains number of methods to do output operations. If error occurs it throws ~~IO~~ ~~Exception~~ IOException.

Method	use
1) close()	This method is used to close the output stream.
2) flush()	This method is used to clear the output buffers.
3) write (~~int~~ b)	This method is used to write a single byte to an output stream.
4) write (byte b[])	This method is used to write an ~~buffer~~ array ~~o~~ of bytes to an output stream.
5) write (byte b[], n, m)	This method is used to write m bytes from ~~buffer~~ array ~~b~~ starting from nth byte.

6) Explain character stream classes.

character stream classes are used to do I/o operations in 16 bit unicode characters.

i) Reader ii) writer

i) Reader

Reader is an abstract class. This class contains number of methods to do input operations. If error occurs it throws IOException.

ii) writer

writer is an abstract class. This class contains number of methods to do output operations. If error occurs it throws

Method	use
close()	This method is used to c the output stream.
flush()	This method is used to clear the output buffer.
write(b)	This method is used to write a single character to an output stream
write(char b[])	This method is used to write array of character to an output stream.
write(char b[], ~~int~~ n, m)	This method is used to write m characters from buffer array b starting from n^{th} character.

Reader

Method	use
1) available()	This method ~~is used~~ gives the number of characters currently available for reading.
2) close()	This method is used to close the input stream.
3) read()	This method is used to read a ~~b~~ character from input stream.
4) read(~~byte~~ char b[])	This method is used to read an array of character.
5) read(~~byte~~ char b[], n, m)	This method is used read m character starting from n^{th} character.
6) skip(n)	This method is used to skip n character.
7) reset()	This method is used