JAVA PROGRAMMING

1. Decision Making and Branching:

When a program breaks the sequencial flow and jumps to another part of (program) code, it is called branching. If the branching is based on particular condition, it is conditional branching and if it is processes without condition is called unbrunconditional branching.

In Java language, such decision making and branching (conditional) is based on three form of statements known as control statements :-

1. If statement.
   a Simple if
   b if else
   c else if
   d nested if statemet.
2. Switch statement.
3. Conditional Operator statement.

1. If statements :

It is used to control the flow of the exeecution of statements, based on the condition as per syntax;

if (test expression).

If the condition is true, then it flows to the next line of a code, else, it moves out of the specified lines and jumps to the other line of the program.

There are 4 implementations of if-statements,

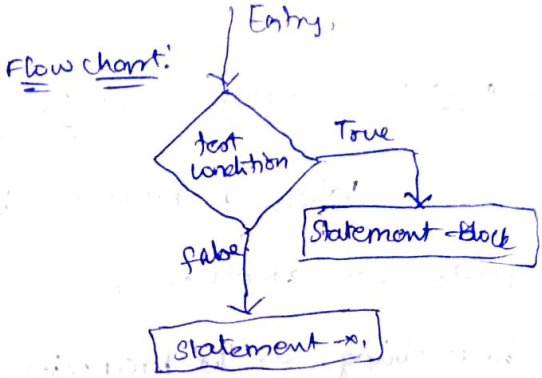
1. Simple if
2. If else
3. else if (elif).
4. Nested if...else. st

(i) Simple 'If' statement:

Syntax:

```
if (test_condition)
{
    statement-block;
}
statement-x;
```

Flow chart:



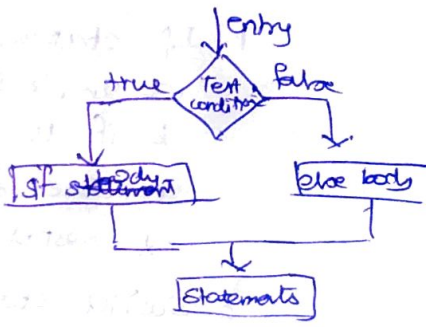
The general form of if statement is followed as if the test-condition is logically true, it moves to statement block, if not, it skips that statement and moves to statem

(ii) If---else statement:

Syntax:

```
If (test_condition)
{
    statement-block of if;
}
else
{
    Statement block of else;
}
```

Flowchart



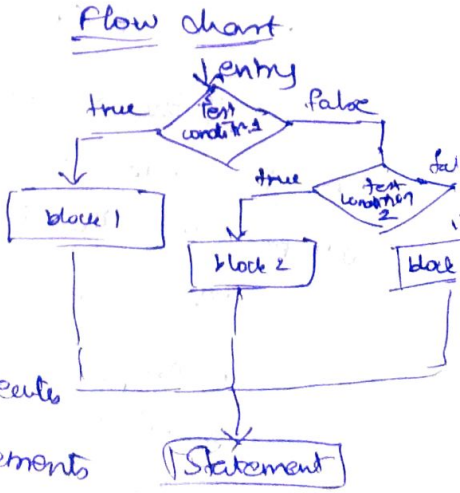
If the condition in the 'test_condition' is true, it moves to the if statement block, if the condition is false, then it jumps to statement after 'else' keyword.

(iii) Else if statement:

Syntax

```
if (condition)
{ block 1; }
else if (condition)
{ block 2; }
else condition
{ block 3; }
```

Flow chart



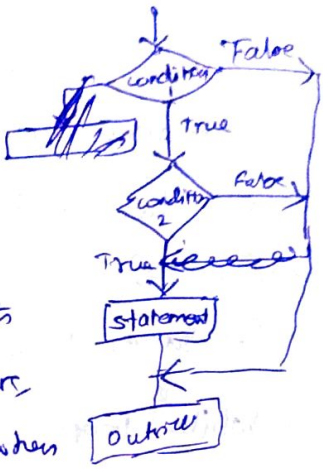
The else if ladder statement executes one condition from multiple statements

(iv) Nested if statement:

Syntax:

```
if(condition)
{
   if (condition)
   { statement; }.
}
```

Flow chart:



The nested if statements represents the if block within another block. Here, the inner if condition executes only when outer if block condition is true.

2. Switch Statement:

→ It is called multiple branching statement

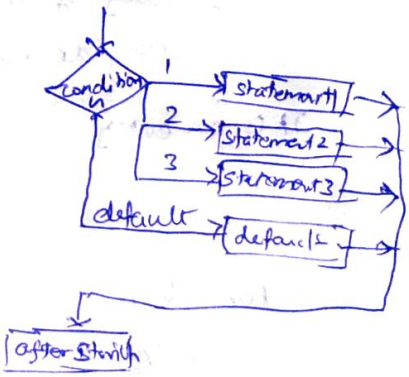
→ It executes the statement from the multiple statements based on the condition.

Syntax:

```
Switch(n)
{
  case 1: statement 1
  case 2: statement 2
  case 3: statement 3

  case n: statement n
  default: default statement
}
After switch statements.
```

Flow chart:



3. Ternary (conditional) operator statement.

→ The conditional operator is so called ternary operator as it performs 2 operations with 3 operands.

op1? op2: op3

→ ~~op1 ? op2 : op3~~ is the general form of conditional statement. if the condition in op1 is true, then returns op2, else op3.

→ Example (n % 2 = 0) ? n is even : n is odd

2. Looping Statements

The process of repeatedly executing a block of statements is known as looping. The statements are iterative maybe which executes ~~many~~ number of times based on the condition (conditional loop) and unconditional (infinite loops).

In Java, there are three forms of looping statements.

1. while loop. } → Entry controlled
2. For loop }
3. do ... while loop. → Exit controlled.

Entry controlled:

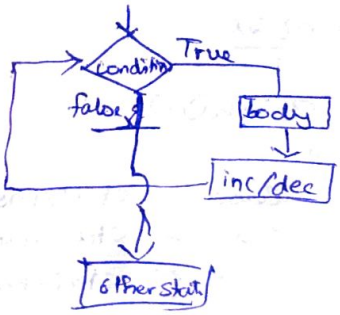
1. While loop:

Syntax:

```
initialisation;
while(condition)
{ Body of the loop;
  incr/decrem; }
```

It is an entry

flow chart:

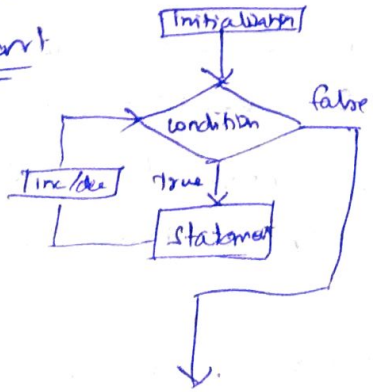


2. For loop:

Syntax:

```
for(initialisation, condition; increment)
{ body of for ; }
```

flow chart:



Exit controlled

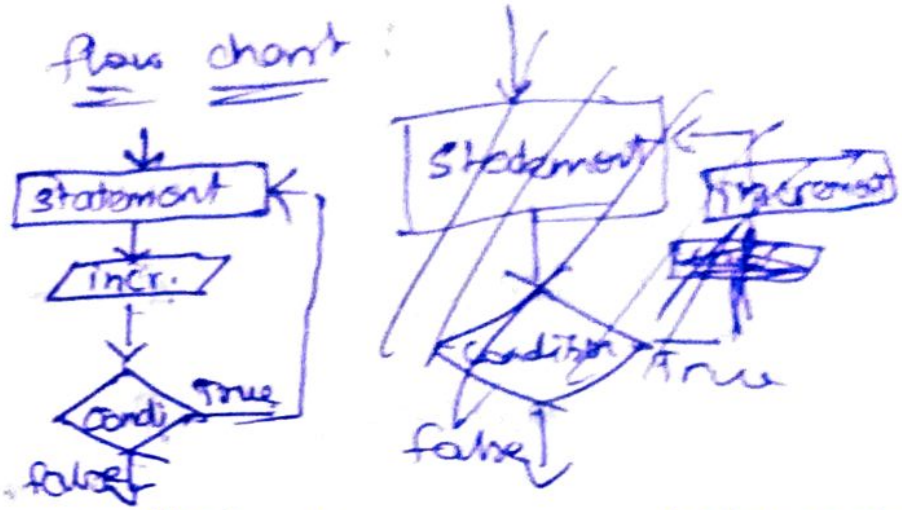
3. do -- while

Syntax

```
do {

body of do while ;
}while (condition)
```

flow chart



It is an exit ~~check~~ loop, such that the ~~given~~ statement inside do keyword is ~~executed~~ executed by compiler and Then checks the condition, if true, the incremented value will be brought to same statements and performs the statement.

3. Vector

Vector is a class in java or utility package that achieves the concept of variable arguments to methods. This class can be used to create generic dynamic array known as vector that can hold object of any type and any number, but are not homogeneous.

Advantages:

1. Convinient to use vectors to store objects.
2. Can be used to store a list of objects that may vary in size
3. We can add or delete objects from a list whenever required.

Disadvantages:

→ We cannot directly store simple data types into an vector. So we need to convert the objects.

This disadvantage is overcome by the wrapper class.

Constructors:

Vector()

Vector(int size)

Vector(int size, int increment)

Vector(collection of elements)

Methods:

1. addElement
2. capacity
3. clone()
4. copyInfo()
5. Enumeration()
6. isEmpty()
7. indexof()
8. lastElement()
9. setSize()
10. size()
11. trimtoSize()
12. contains → boolean.

Example

```
import java.util.*;
class Vector_Sample
{
    public static void main(String args[])
    {
        Vector st=new Vector();
        int length = args.length;
        for(int i=0;i<length;i++)
        {st.addElement(args[i]);}
        st.insertElementAt("COBOL",2);
        int s=st.size();
        String arr=new String();
        st.copyInfo(arr);
        System.out.println("List of languages");
        for(int i=0; i<s; i++)
        {System.out.println(" "+arr[i]);}
    }
}
```

Output:

```
Java LanguageVector A B D
List of languages
A
B
COBOL
D
```

5. String Buffer class :

→ String Buffer class is the peer class of string.

→ While string creates strings of fixed_length, String buffer creates strings of flexible length.

→ Using this, we can insert characters, substrings in middle of a string, or append another string to the end.

Constructor:

1. String Buffer ( ); (→ capacity of 16 chars)
2. string Buffer (int size); (→ capacity of size as many as possible)
3. String Buffer (String str);

Methods : [Let str be string object].

1. append (String str)
2. insert (index , "str")
3. replace (index x, index e, "str");
4. delete (sindex , eindex)
5. reverse (str);
6. capacity (str);
7. length (str);
8. setChar At (index, 'str');
9. substring (index);
10. deleteChar At (index);

Example:

```
import java.util.*;
class A
{
    public static void main (String args[])
    {
        StringBuffer str = new StringBuffer ("Hello");
```

```
        str.append("java"); // H Java
        str.insert(1, "ello");
        System.out.println(str); // Hello java)
        str.replace(6,10,"cpp");
        System.out.println(str); // Hello cpp
        str.delete(6,9);
        str.reverse();
        System.out.println(str); // olleH
        str.capacity();
        System.out.println(str); // 4
    }
}
```

Output :

Hello Java

Hello cpp

Hello

olleH

4.

5. Method Overloading

In Java, it is possible to create methods that have the same name, but different parameter lists and different definitions. This is called method overloading. This is used when an objects are required to perform similar tasks but using different input parameters.

When we call a method in an object, Java matches up the argument or datatypes in the arguments with the called function.

Example:

```
class Room
{
    float l;
    float b;
    Room(float x, float y)
    {
        l=x; b=y;
    }
    Room(float x)
    {
        length = breath = x;
    }
    int area()
    {
        return (l*b);
    }
}
class Main
{
    public static void main(String args[])
    {
        Room r= new Room(2)
        System.out.println("Square room:"+r);
        Room t= new Room(2,3)
        System.out.println("Room:"+t);
    }
};
```

6. String class :

In Java, strings is basically an object that represents sequence of character values. An array of characters works same as Java string. It is more reliable and predective compared to C.

But,

→ It is not an array of characters.

→ It is not a NULL terminated.

There are 2 forms to ~~declare~~ create an string object :

1) By String literal

2) By 'new' keyword.

1) String literals :

It is definable and also declarable. It is defined (or declared) by,

```
String stringname = "welcome" ; // definition
String stringname ; // declaration.
```

2) New keyword :

```
String str = new String();
```

String array :

We can also create string arrays that contains strings.

```
String arr[] = new String[3];
```

Here, arr is declared as size 3.

String methods :

* Let S_2 be the new or converted string objects and S_1 be the existing string objects.

1. $S_2 = S_1$.toLowerCase();
2. $S_2 = S_1$.toUpperCase();
3. $S_2 = S_1$.replace('x', 'y');
4. $S_2 = S_1$.trim();
5. S_1.equals(S_2);
6. S_1.equalIgnoreCase(S_2);
7. S_1.length();
8. S_1.charAt(n)
9. S_1.CompareTo(S_2)
10. S1.concat(S2)
11. S1.substring(n);
12. S1.substring(n,m);
13. String.valueOf(p)
14. p.toString()
15. S1.indexOf('x')
16. S1.indexOf('x', 6)
17. String.valueOf(variable)

7. Java Utility Stack Package

Stack is the subclass of Vector class that implements a standard last-in first-out. Stack only defines a default constructor, which creates an empty stack.

- → To put an object on the top of stack, call push()
- → To remove and return the top element of stack, call pop()
- → To display the top of the element, call peep().
- → To ~~return~~ check ~~true~~, if the stack is empty, ~~search~~ empty() is used
- → Search() method determines whether an object exists on the stack and returns the number of pops that requires to bring it to the top of stack.

Example Program

```
import java.util.Stack;
class StackDemo {
    Public static void main (String args [])
    {
        Stack S1 = new Stack ();
        S1. Push (42);
        S1. push (46);
        S1. push (99);
        System.out.println(S1.pop());
        System.out.println(S1.peek());
        System.out.println(S1.search(99));
    }
}.
```