Java Programming

1. Decision Making and Branching

When a program breaks the sequential 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 unconditional 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 statement
2. Switch statement
3. Conditional Operator statement

1. If statements

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

```
if (test expression)
```

If the condition is true then it moves 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.

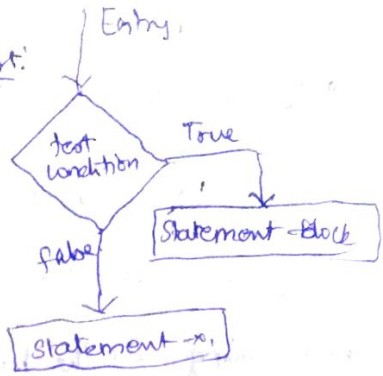
1. Simple if
2. If else
3. else if (elif)
4. Nested if - else if

(i) Simple 'If' statement:

Syntax:

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

Flow chart:



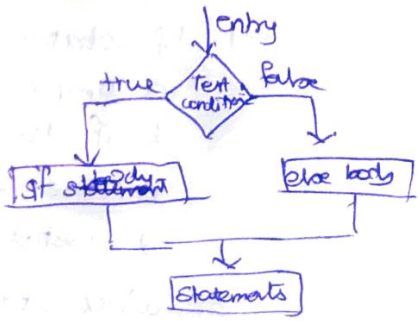
The general form of if statement is followed as if the test-condition is logically true, it moves to statement block, if not, it skips that statement and moves to statement-x.

(ii) If...else statement:

Syntax:

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

Flowchart



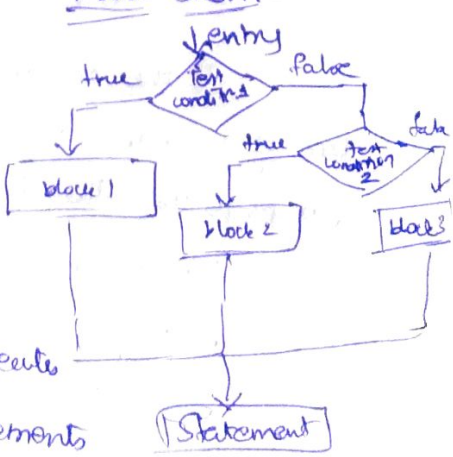
If the condition in the 'test_condition' is true, it moves to the if statement block, if the condition is false, then it jumps to statement after 'else' keyword.

(iii) Else if statement:

Syntax

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

Flow chart



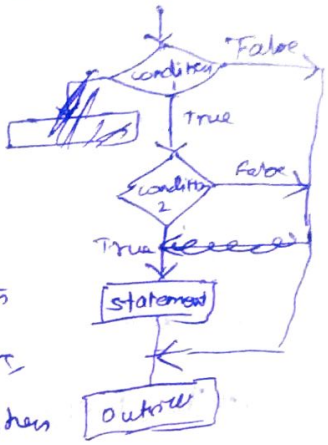
The else if ladder statement executes one condition from multiple statements

(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

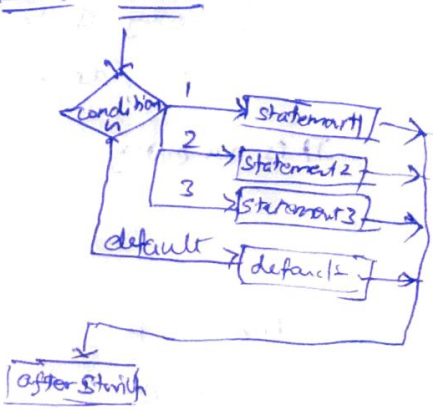
→ It executes the statement from the multiple statements based on the condition.

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 ? Op~~ 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:

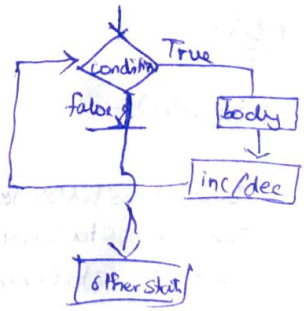
1. while loop:

Syntax:

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

It is an entry

flow chart:

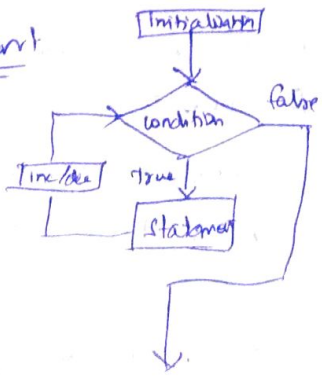


2. For loop:

Syntax:

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

flow chart:



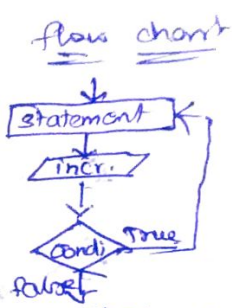
Exit controlled

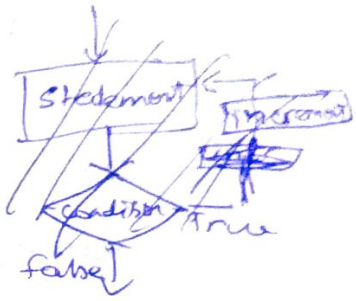
3. do -- while

Syntax:

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

flow chart





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

---

3. Inheritance:

Inheritance is a mechanism in which an objects acquires all the properties and behaviour of the parent object. In other words, proceeding of acquiring base class (or) properties by child class is called inheritance.

→ The parent class is called superclass.

→ The child class is called subclass.

Defining subclass:

The 'extend' keyword is used in between the name of subclass and corresponding super class. This keyword indicates that you are making a new class that derives from existing class.

Syntax:

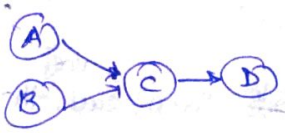
```
class SubClass extends SuperClass
{
    variable declaration;
    function declaration;
}
```

Types of Inheritance :

On basis of class, there are four types of inheritance in java.

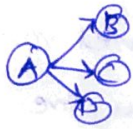
1. Single : A → B

2. Multiple :



3. Multilevel : A → B → C

4. Hierarchial :



1. Single inheritance :

When a class extends to another one class, it is called single inheritance.

Syntax :

```
class A extends B
{ //body A ; }

class A extends B
{//body B ; }.
```

2. Multiple inheritance :

~~When one class can inherit from other~~

~~There are~~

The derivation by a class from two or more super class is called multiple inheritance. Due to the case that the derived class will have to manage the dependency of 2 or more classes at the time, Java does not support direct multiple inheritance. So the concept of interface is brought out, which is has common properties of all parent classes.

Syntax :

```
[Visibility] interface Name [extends other interface]
{
    variables;
    functions;
}
class A implements AName
{
  //body of A;
}
class B
{
  //body of B;
}
class C implements Name
{
  //body of C;
}
```

3. Multilevel inheritance :

Here, one can inherit from derived class, thereby making this derived class as base class of other base class. It is called multilevel inheritance

Syntax :

```
class A
{
  //body of A
}
class B extends A
{
  //body of B
}
class C extends B
{
  //Body of C
}
```

4. Hierarchial inheritance:

Here, a super class gets inherited by many subclasses

Syntax:

```
class A
{
} // Body of A.
class B extends A
{
} // body of B
class C extends A
{
} // body of C
```

---

4. 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. S_1.concat(S_2)
11. S_1.substring(n);
12. S_1.substring(n, m);
13. String.valueOf(p)
14. p.toString()
15. S_1.indexOf('x')
16. S_1.indexOf('x', 6)
17. String.valueOf(variable)

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 (index , index)
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[])
    {
        String Buffer str = new String Buffer ("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
```