Chapter 5

CONTROL STATEMENTS

In this chapter, the statements that take the control from one location to another during execution are given. The control statements help to make branching, looping, skipping and exiting a block of codes.

Basically, any computer-oriented problem can be solved using any combination of sequential, branching and looping structure. Therefore, a computer programming language should support these three structures. In this chapter, the statements supporting the branching and looping structures in Java are given. These statements help to branch or loop a segment of statements and are called control statements.

5.1 The if..else Statement

This statement helps to select one out of two possibilities based on the given condition. Hence, this statement is also called as conditional if statement.

The general form of the statement is:

```
if (conditional expression)
        statement1;
 else
        statement2;
```

The conditional expression should result in a boolean value. If the condition, on evaluation, gives **true**, statement1 is executed and the control skips statement2, otherwise statement1 is skipped and statement2 is executed.

The statement1 and the statement2 can be a simple or block statement. The else part is optional and, if needed, can be left out and can take the form:

if (conditional expression)
statement;

In this form, the statement will be executed, if the conditional expression gives **true**, otherwise it is skipped.

The flowchart for if...else and if... are given in fig.5.1 and fig.5.2 respectively.

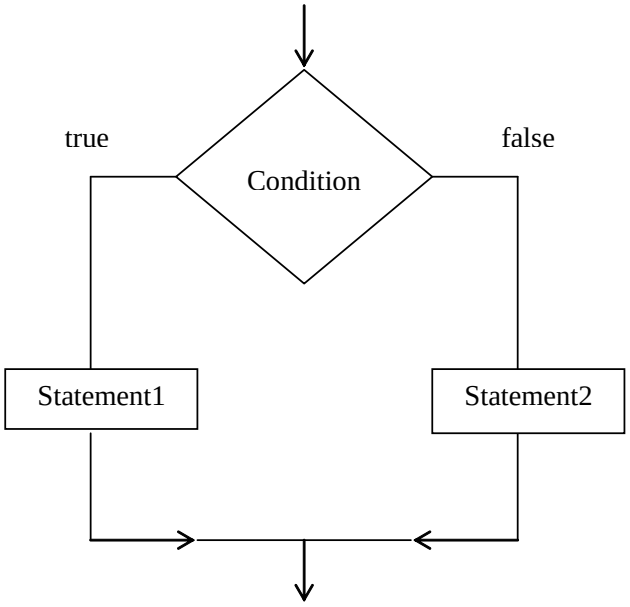

Fig.5.1 Flowchart for if...else Statement

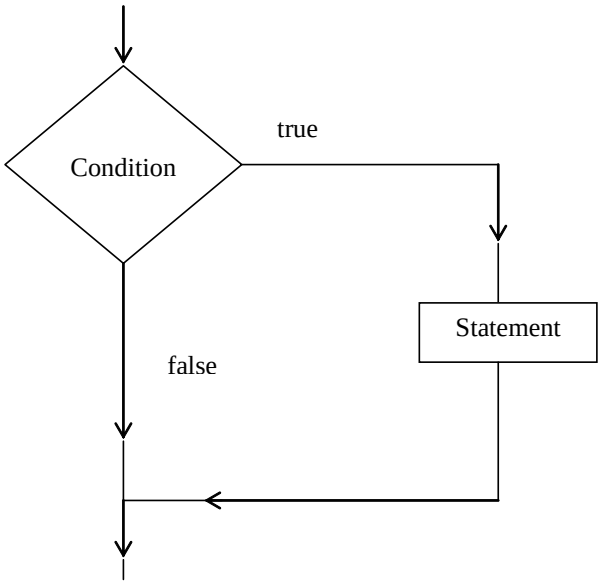

Fig.5.2 Flowchart for if...Statement

The following examples, 5.1 and 5.2, illustrate the if...else and if statement:

Example 5.1

```
int mark;
String result;
if (mark >= 40)
        result = “Pass”;
else
        result = “Fail”;
```

Example 5.2

```
double a, b, c, discr, term;
discr = b*b-4.0*a*c;
if (discr < 0)
discr = -discr;
term = Maths.sqrt(discr);
root1 = (-b+term)/(2.0*a);
root2 = (-b-term)/(2.0*a);
```

Nested if..else

Nested if..else statement is made by placing one if..else inside another if..else statement. Nested if..else statement helps to select one out of many choices. The general form of if..else statement is :

```
if (condition1)
        if (condition2)
                if (condition3)
                   statement 4
                else
                        statement 3
        else
                statement 2
else
        statement 1
```

In the **nested if..else** statement, the outermost **if** is evaluated first. If the condition1 tested is false, the statement1 in the outmost **else** is evaluated and **if..else** ends. If the condition1 results in true, the control goes to execute the next inner **if** statement. If condition2 is false, statement2 is executed. Otherwise, condition3 is evaluated. If condition3 is false, statement3 is executed, otherwise statement4 is executed.

The flowchart for the **nested if..else** is given in fig.5.3:

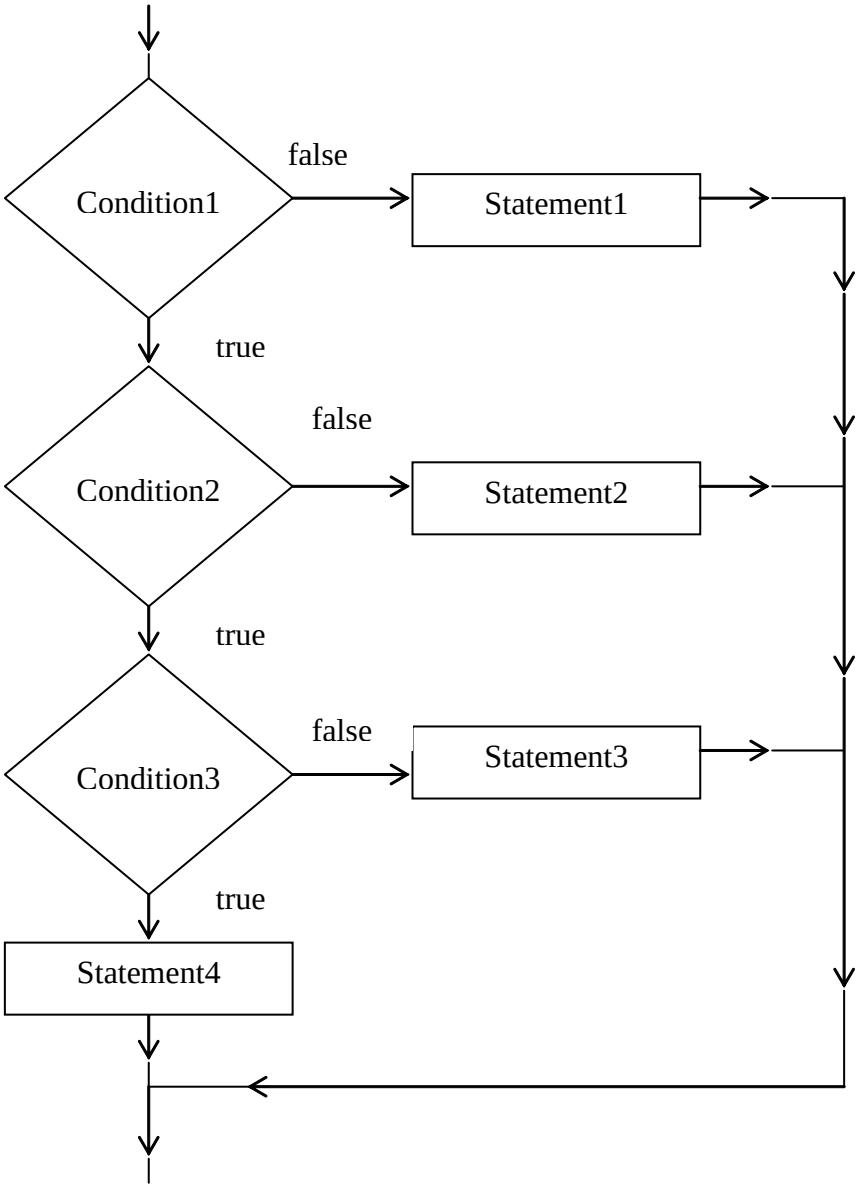

Fig.5.3 Flowchart for nested if...else Statement

The following program 5.1 illustrates the use of nested if..else statement:

Program 5.1

```
//This program  illustrates the nested if... else  statement.
//somasundaramk@yahoo.com

class NestedIf
    {
    public static void main(String args [])
        {
        String result_class;
        int avg_mark = 62;
```

```
        if (avg_mark > 50)
            if (avg_mark >= 60)
                if (avg_mark >= 75)
                    result_class = “Distinction”;

                else
                    result_class = “First Class”;

            else
                result_class = “Second Class”;

        else
            result_class = “Third Class”;

        if (avg_mark < 40)
            System.out.println(“Failed”);

        else
            System.out.println(“Passed with “ +result_class);
        }
    }
```

The above program gives the following output:

```
Passed with First Class
```

Another form of nested if..else is the ladder if..else statement. The structure of the ladder if..else statement is :

```
if (condition1)
        statement1;
else if (condition2)
        statement2;
        else if (condition 3)
        statement 3;
        else
        statement4;
```

The flowchart for the above ladder if...else statement is given in fig. 5.4:

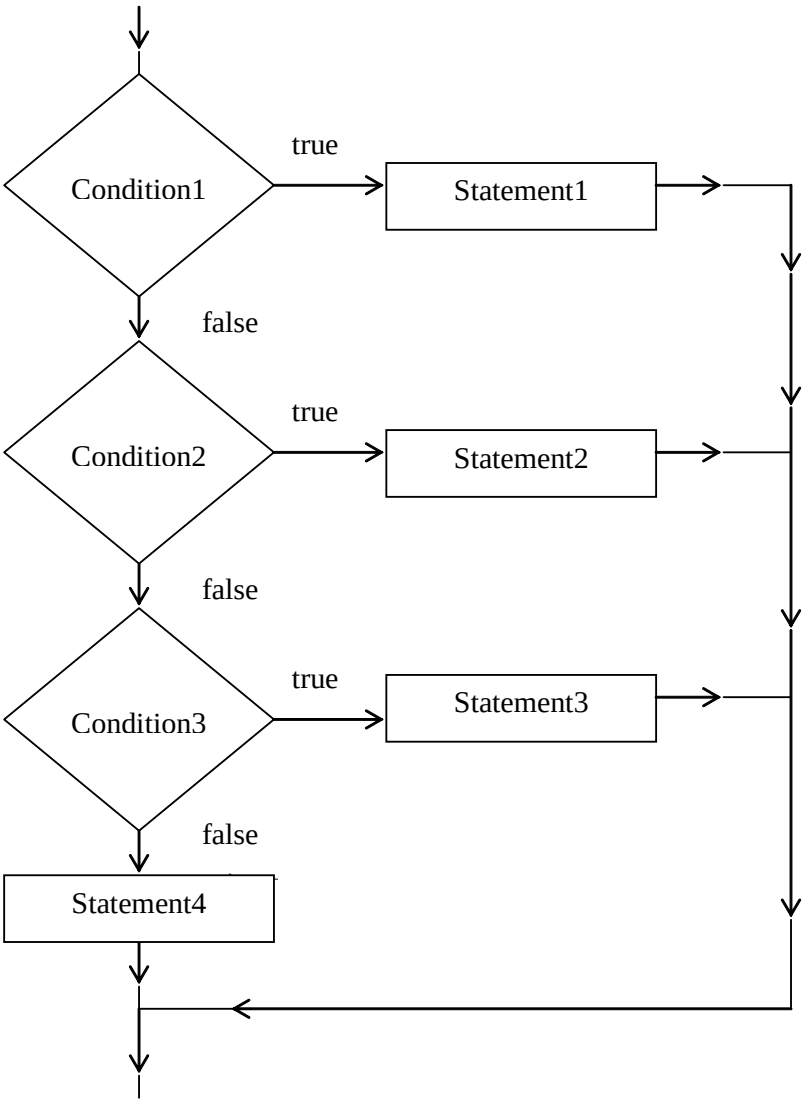

Fig.5.4 Flowchart for Ladder if..else Statement

The following program 5.2 illustrates the ladder if...else statement:

Program 5.2

```
//This program  illustrates the ladder if...else if statement.
//somasundaramk@yahoo.com

class IfElseIf
    {
    public static void main(String args [])
        {
        String result_class;
        int avg_mark = 62;
        if (avg_mark > 75)
            result_class = “Distiction”;
        else if (avg_mark >= 60)
            result_class = “First Class”;
```

```
        else if (avg_mark >= 50)
            result_class = “Second Class”;
        else
            result_class = “Third Class”;
        if (avg_mark < 40)
            System.out.println(“Failed”);
        else
            System.out.println(“Passed with “ +result_class);
        }
    }
```

The above program gives the following output:

```
Passed with First Class
```

5.2 The switch Statement

The **switch** statement helps to select one out of many choices. This helps to write a clear statement when compared to nested if...else statements. The general form of **switch** statement is:

```
switch (expression)
{
case val1 :   statement1;
                break;
case  val2 :  statement2;
                break;
case  val3 :  statement3;
                break;
.
.
.
case valN : statementN;
                break;
default : statement;
}
```

It is expected that the expression, when evaluated, should give discrete values in the range Val1 to valN. If the expression gives val1, the statement1 is executed and the control exits the switch block. If the expression gives val2, statement2 is executed and so on (fig.5.5). If the expression gives any value that is not matching between val1 and valN, the statement given in default is executed. The expression should give any of the type **byte**, **short**, **int** or **char**. In this way, multiple branching can be effected. The **break** statement when executed will take the control out of the **switch** block. Generally, **switch** statement is used for expressions that may give discrete and predefined values.

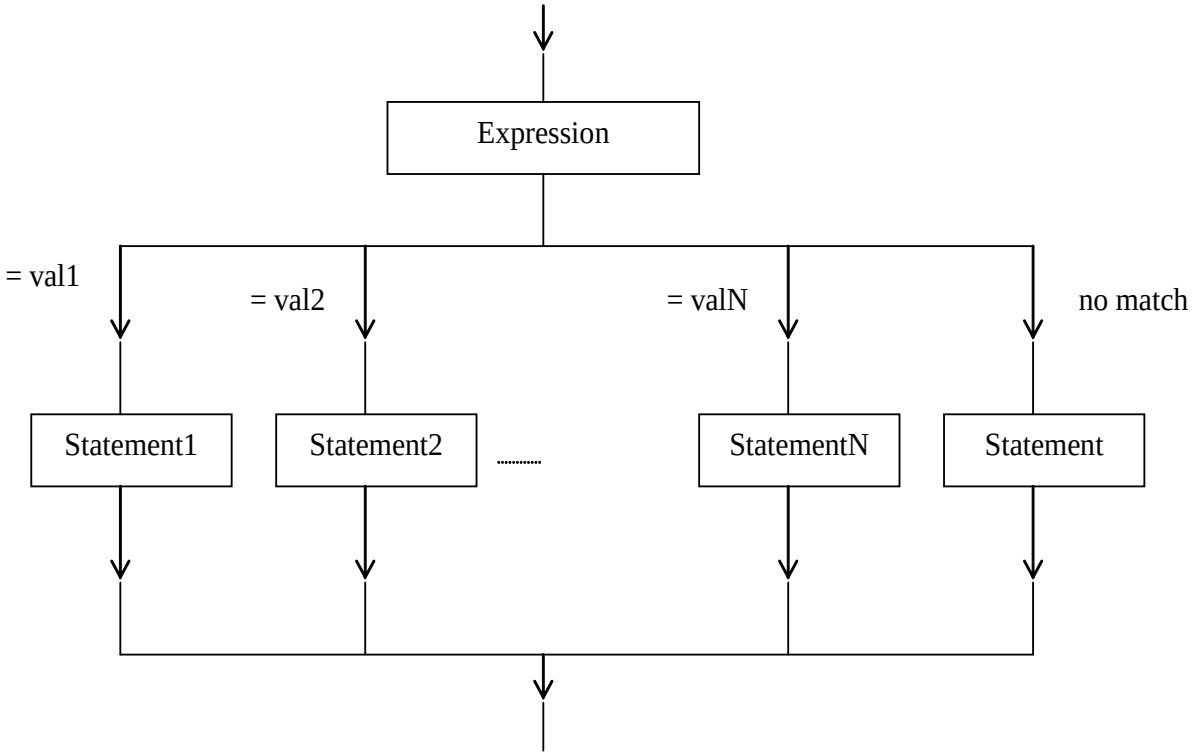

Fig.5.5 Flowchart for switch Statement

The following program 5.3 illustrates the use of **switch** statement:

Program 5.3

```
// This program illustrates the switch statement.
class SwitchDemo
    public static void main(String args [])
        {
        int month = 5;
        switch (month)
            {
            case 1:
                System.out.println(“January”);
              break;
            case 2:
              System.out.println(“February”);
              break;
            case 3:
              System.out.println(“March”);
              break;
            case 4:
              System.out.println(“April”);
              break;
            case 5:
              System.out.println(“May”);
              break;
            case 6:
```

```
                System.out.println("June");
                break;
            case 7:
                System.out.println("July");
                 break;
            case 8:
                System.out.println("August");
                 break;
            case 9:
                System.out.println("September");
                 break;
            case 10:
                System.out.println("October");
                 break;
            case 11:
                System.out.println("November");
                 break;
            case 12:
                 System.out.println("December");
              break;
            default:
                 System.out.println("No match");
            }
        }
    }
```

The above program gives the following output:

```
May
```

Sometimes, for several case values, one common process may be required. In such problems, several case values can be clubbed so that statements common to all of them can be executed at one point. The following program 5.4 shows how several cases can be combined:

Program 5.4

```
// This program illustrates the switch statement.
class GroupCase
    {
    public static void main(String args [])
        {
        int choice = 6;
        switch (choice)
            {
            case 1:
            case 2:
            case 3:
           System.out.println("Numbers between 1 and 3");
```

```
            break;
        case 4:
        case 5:
        case 6:
        case 7:
            System.out.println("Numbers between 4 and 7 ");
            break;
        case 8:
        case 9:
        case 10:
            System.out.println("Numbers between 8 and 10");
            break;
        default:
            System.out.println("Numbers outside 1 and 10");
             }
        }
}
```

The above program gives the following output:

```
Numbers between 4 and 7
```

The type of data handled in **switch** statement must be of the type **byte**, **short**, **int** or **char**. The values used must be discrete.

5.3 The while Statement

The **while** statement is used for looping or iterating a block of statements while the given condition is **true**. The general form of the **while** statement is:

```
while (condition)
{
statements;
}
```

The condition, when evaluated, must result in the boolean value **true** or **false**. As long as the condition gives **true,** the statement block will be executed. When the condition becomes **false**, the control leaves the block statement. The flow of control in **while** statement is given in fig. 5.6.

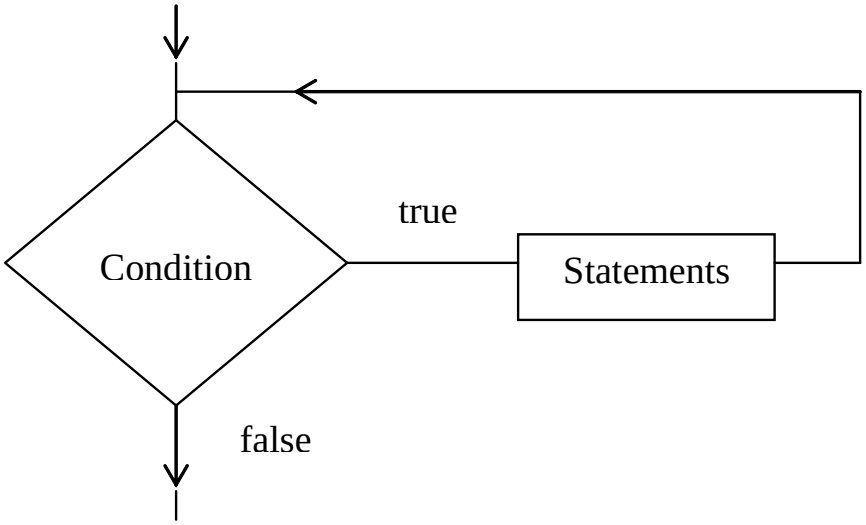

Fig.5.6 Flowchart for while Statement

When the condition tested is **false** in the first instance itself, the block statement will not be executed at all. The following program 5.5 shows the use of **while** structure:

Program 5.5

```
// This program illustrates the use of while statement.
class WhileDemo
    {
    public static void main(String args [])
        {
        int n = 10;
        int sum = 0;
        while (n > 0)
            sum += n--;
        System.out.println("Sum of numbers from 1 to 10 is= "
                                                + sum);
        }
    }
```

The above program gives the following output:

```
Sum of numbers from 1 to 10 is = 55
```

In **while** statement, the condition is checked before executing the codes.

5.4 The do..while Statement

This **do..while** control statement is used for looping a block of statements while the given condition is **true**. In this structure, the condition is tested at the end of the block. This is in contrast to the **while** .. statement, where the condition is tested at the start of the block. The general form of **do..while** statement is:

```
do
 {
  statements;
 } while (condition);
```

The condition tested can be any expression that will yield a boolean value. As long as the condition tested is **true,** the whole block will be executed. Once the condition tested becomes **false**, the control leaves the block. It is to be noted that the block is executed once before the condition is tested. Therefore, even if the condition tested is **false** at the first instance itself, the block statement is executed once. The flowchart of a **do..while** structure is given in fig.5.7.

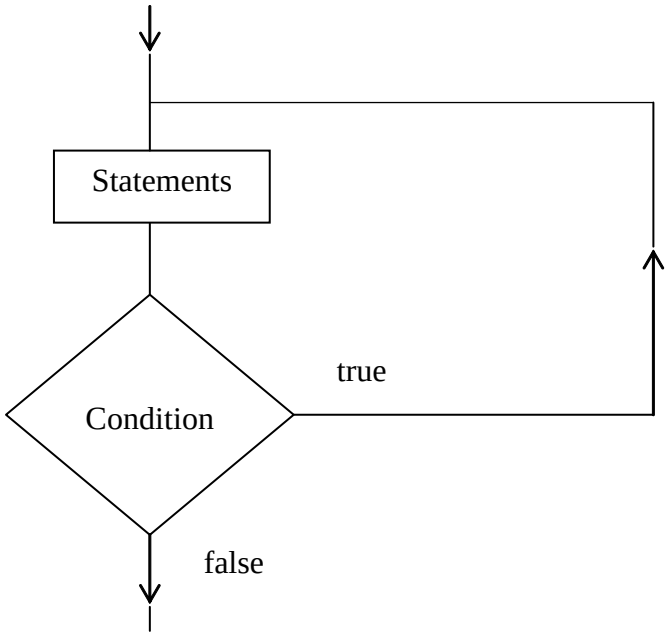

Fig.5.7 Flowchart for do....while Statement

The following program 5.6 illustrates the **do ... while** statement:

Program 5.6

```
// This program illustrates the use of do..while statement.
class DoWhileDemo
    {
    public static void main(String args [])
        {
        int n = 10;
        int sum = 0;
        do
           sum += n--;
        while (n > 0);
        System.out.println("Sum of numbers from 1 to 10 is
                    = " + sum);
        }
    }
```

The above program gives the following output:

```
Sum of numbers from 1 to 10 is = 55
```

In **do...while** statement, the condition is checked after executing the codes.

5.5 The for ... Statement

This is the third form of looping statement. This statement is a self-contained one, that is the initial value, termination condition, iterator (increment/ decrement) are all given in the **for** structure itself. The general form of **for** statement is:

```
for (initializer; condition; iterator)
{
  statements;
}
```

Here, the condition can be any expression that yields boolean value **true** or **false**. The initializer is used to store the initial value for a loop variable. The iterator modifies the loop variable. The statements in the block are executed, as long as the condition tested gives **true.** When the condition becomes **false**, the control leaves the block statement. The flowchart for the **for** loop is given in fig. 5.8.

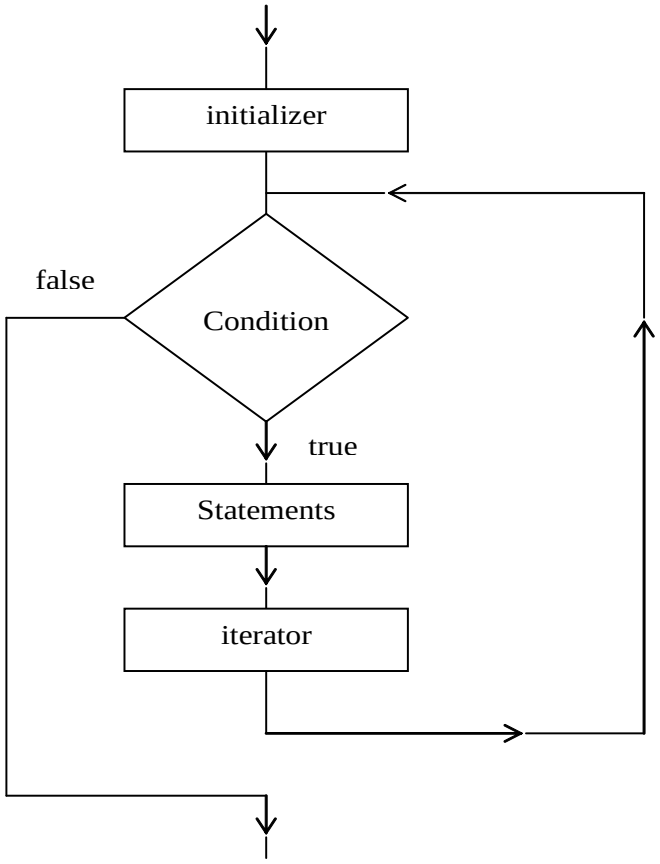

Fig.5.8 Flowchart for the for Loop

Program 5.7 shows the use of **for**.. statement.

Program 5.7

```
/* This program illustrates the use of for statement.
   This program finds the sum of all numbers from 1 up to 10.
*/
class ForDemo1
    {
    public static void main(String args [])
        {
        int n;
        int sum = 0;
        for (n = 10; n > 0; n--)
            sum += n;
        System.out.println("Sum of numbers from 1 up to 10 ="
                                + sum);
        }
    }
```

The above program gives the following output:

```
Sum of numbers from 1 up to 10 = 55
```

The above program 5.7 can be written in another form as given in program 5.8. The loop variable n can be declared inside the **for** structure itself. The iterator is optional. If there is no need for an iterator in a program, it can be left blank.

Program 5.8

```
/* This program illustrates the use of for statement.
   Notice that n is declared inside the for loop.
   The iterator is left blank as it has been
   done in the block statement itself.
 */
class ForDemo2
    {
    public static void main(String args [])
        {
        int sum = 0;
        for (int n = 10; n > 0; )
            sum += n--;
         System.out.println("Sum of numbers from 1 up to 10 =
                                " + sum);
        }
    }
```

The **for** loop in program 5.8 can be improved further and written in a compact form as :

for (int n=10; n>0; sum+=n--);

In a **for** structure, one, two or all the three components, initializer, condition, iterator, can be absent. The statement for (; ;) is valid and will make an infinite loop.

The following program 5.9 finds the factorial of numbers from 1 to 10.

Program 5.9

```
// This program finds the factorial of a number using for
// statement.
class FactProg
    {
    public static void main(String args [])
        {
        int n = 10;
        long fact = 1;
```

```
        for (int i = 1; i <= n; ++i)
            {
            fact *= i;
             System.out.println("Factorial of  " + i + " = "
                                                + fact);
            }
        }
    }
```

The above program gives the following output:

```
Factorial of 1 = 1
Factorial of 2 = 2
Factorial of 3 = 6
Factorial of 4 = 24
Factorial of 5 = 120
Factorial of 6 = 720
Factorial of 7 = 5040
Factorial of 8 = 40320
Factorial of 9 = 362880
Factorial of 10 = 3628800
```

The **for** loops can be nested one within another as shown below:

```
- - - - - - - -
- - - - - - - -
- - - - - - - -
for (i=1; i<n; i++)          // outer loop begins
{
      - - - - - - - - -
      - - - - - - - - -
for (k=1; k<m; k++)          // inner loop begins
      {
      .
      .
      .
        }                    // inner loop ends
        .
        .
        .
}                            // outer loops ends
        .
        .
        .
```

The following program 5.10 illustrates the nested **for** loop. It generates multiplication tables for 2, 3 and 4.

Program 5.10

```
// This program illustrates the nested for loop.
class MulTable
    {
    public static void main(String args [])
        {
        int i, j, m = 4, n = 5;
        int prod;
        for (i = 2; i <= m; ++i)
            {
             System.out.println("Multiplication
                                        Table for" + i);
             for (j = 1; j <= n; ++j)
                {
                prod = j * i;
                System.out.println(j + "X" + i + " = " +
                                    prod);
                }
            System.out.println(" ");//create one line space
            }
        }
    }
```

The above program gives the following output:

```
Multiplication Table for 2
1X2 = 2
2X2 = 4
3X2 = 6
4X2 = 8
5X2 = 10
Multiplication Table for 3
1X3 = 3
2X3 = 6
3X3 = 9
4X3 = 12
5X3 = 15
Multiplication Table for 4
1X4 = 4
2X4 = 8
3X4 = 12
4X4 = 16
5X4 = 20
```

5.6 The break Statement

The **break** statement is to be used only in loops and **switch** statement. When this statement is executed, the control is taken out of the loop. The loop becomes dead. The flowchart for the break statement is given in fig. 5.9.

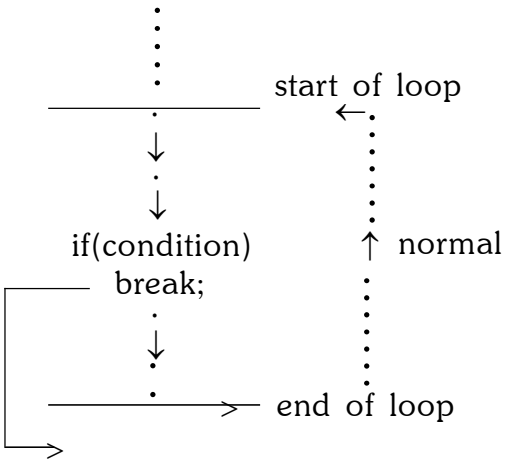

Fig.5.9 The Flowchart for break Statement

The following program 5.11 illustrates the use of **break** in a **for** loop:

Program 5.11

```
// This program illustrates the use of break statement.
class BreakDemo
    {
    public static void main(String args [])
        {
        int n = 5, m = 20;
        for (int i = 1; i <= m; i++)
            {
            if (n * i > 30)
                {
                 System.out.println("Exiting the loop");
                break;
                }
            System.out.println(i + "X" + n + " = " + n * i);
            }
        }
    }
```

The above program gives the following output:

```
1X5 = 5
2X5 = 10
3X5 = 15
4X5 = 20
5X5 = 25
6X5 = 30
Exiting the loop
```

Blocks can be labeled and the **break** control can be made to exit any labeled block. This is called labeled break. The label for a block can be

formed by using the rules to form a variable. The labeled break is useful when using nested loops. The following program 5.12 illustrates the labeled break statement:

Program 5.12

```
// This program illustrates the use of labeled break.
class LabelBreak
    {
    public static void main(String args [])
        {
        int n = 0, m = 0;
        loop1:
        while (true)
            {
            n++;
            m = 0;
            System.out.println("\n  loop1 " + "n = "  + n);
            loop2:
            while (++m <= n)
                {
                if (n * m > 10)
                    {
                    System.out.println("Exiting loop2 and
                                        loop1");
                    break loop1;
                    }
                System.out.println("loop2   " + m + "X" +
                     n + " = " + n * m);
                }
            }
        System.out.println("outside loop1");
        }
    }
```

The above program gives the following output:

```
  loop1 n = 1
loop2  1X1 = 1

  loop1 n = 2
loop2  1X2 = 2
loop2  2X2 = 4

  loop1 n = 3
loop2  1X3 = 3
loop2  2X3 = 6
loop2  3X3 = 9
```

```
 loop1  n = 4
loop2   1X4 = 4
loop2   2X4 = 8
Exiting loop2 and loop1
outside loop1
```

5.7 The continue Statement

The **continue** statement is used inside the loop control blocks. When the **continue** statement is executed, the control skips the remaining portion of the loop and goes to the beginning of the loop and continue. The flowchart for the continue statement is given in fig. 5.10:

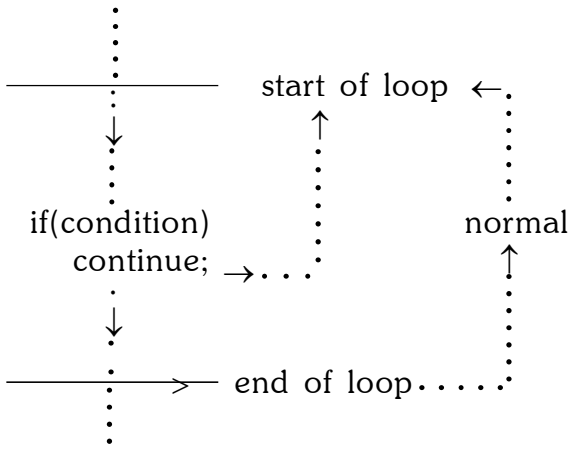

Fig.5.10 Flowchart for continue Statement

The following program 5.13 illustrates the use of continue statement:

Program 5.13

```
// This program illustrates the use of continue statement.
class ContinueProg
    {
    public static void main(String args [])
        {
        int i = 0;
        while (++i < 10)
            {
            if (i % 2 == 0)
                continue;
            System.out.println(i);
            }
        }
    }
```

The above program gives the following outptut:

```
1
3
5
7
9
```

Like labeled break, labeled continue structure is also available. The following program 5.14 illustrates the use of labeled continue:

Program 5.14

```
// This program illustrates the use of labeled continue
// statement.
// This program generates prime numbers between 1 and 15.
// somasundaramk@yahoo.com
class LabelContinue
    {
    public static void main(String args [])
        {
        int i = 1, n, max = 15;
        System.out.println("Prime numbers between 1 and 15");
        System.out.println(i);
        start:
        for (n = 2; n <= max; ++n)
            {
            test:
            for (i = 2; i < n; ++i)
                if (n % i == 0)
                    continue start;
            System.out.println(n);
            }
        }
    }
```

The above program gives the following output:

```
Prime numbers between 1 and 15
1
2
3
5
7
11
13
```

5.8 The comma Statement

Java allows multiple initialization and iteration in the **for** statement. With this capability, more than one variable can be initialized and more than one iteration can be done. Each of such statements are to be separated by a comma(,). But there can be only one conditional expression. The general form of the multiple initialization and iteration is:

```
for (init1, init2, init3;  condition; itr1, itr2, itr3)
{
 Statements;
}
```

The following program 5.15 illustrates the use of comma statement.

Program 5.15

```
// This program illustrates the use of comma statement.
class CommaDemo
    {
    public static void main(String args [])
        {
        int n, i, sum;
        for (i = 1, n = 5; n > 0; i++, n–, sum = 0)
            {
            sum = i + n;
            System.out.println(i + " + " + n + " = " + sum);
            }
        }
    }
```

The output for the above program is :

```
1 + 5 = 6
2 + 4 = 6
3 + 3 = 6
4 + 2 = 6
5 + 1 = 6
```

After reading this chapter, you should have learned the following:

- ➲ Branching using if...else control
- ➲ Branching using switch control
- ➲ Looping with while, do...while and for...control
- ➲ Using break and continue

In the next chapter, you will learn about arrays.

Worked Out Problems–5

Problem 5.1w

The trigonometric functions Sin(x) and Cos(x) are computed using the following formula:

$$\text{Sin}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!}$$

$$\text{Cos}(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!}$$

Calculate the Sin(x) and Cos(x) values for x=0, 0.5 and 1.5. The following expressions which represent the above are made use of:

$$\text{Sin}(x) = x + \sum_{n=1}^{m} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

$$\text{Cos}(x) = 1 + \sum_{n=1}^{m} \frac{(-1)^n x^{2n}}{2n!}$$

Program 5.1w

```
/* ------------------------------------------------------------
This program calculates the sin(x) and cos(x) functions.
The values obtained are compared with those of Java functions.

somasundaramk@yahoo.com
------------------------------------------------------------ */

class Prob51
    {
    public static void main(String args [])
        {
        double x, sinx, cosx;
        int n, fact = 1, factn, factnplus;
        int m = 5;
        for (int i = 0; i < 80; i++)
```

```
        System.out.print("-");

    System.out.println("\n");
    System.out.println("x \t my sinx \t Java sinx \t my
                          cosx \t Java cosx\n");

    for (int i = 0; i < 80; i++)
        System.out.print("-");

    System.out.println("\n");
    for (x = 0; x < 1.6; )
        {
        sinx = x;

        cosx = 1;

        for (n = 1; n <= m; n++)
            {
            factn = 1;

            factnplus = 1;

            for (int i = 2; i <= 2 * n; i++)
                factn = factn * i;

            for (int i = 2; i <= (2 * n + 1); i++)
                factnplus = factnplus * i;

            sinx = sinx + Math.pow(-1, n) * Math.pow(x, (2 *
                                  n + 1)) / factnplus;
            cosx = cosx + Math.pow(-1, n) * Math.pow(x, 2 *
                                  n) / factn;
            }

           // Reduce the fractional digits for display
           double sx = (int)(sinx * 1000);
           double cx = (int)(cosx * 1000);
           sinx = sx / 1000;
           cosx = cx / 1000;
           double jsx = (int)(Math.sin(x) * 1000);
           double jcx = (int)(Math.cos(x) * 1000);
           jsx = jsx / 1000;
           jcx = jcx / 1000;
```

```
            System.out.println(x + "\t" + sinx + "\t\t" + jsx
                    +"\t\t" + cosx + "\t\t" + jcx);
            x = x + 0.5;
            }
        for (int i = 0; i < 80; i++)
            System.out.print("-");
        System.out.println("\n");
        }
}
```

The above program gives the following output:

```
--------------------------------------------------------------------------------
x       my sinx         Java sinx       my cosx             Java cosx

--------------------------------------------------------------------------------

0.0     0.0             0.0             1.0                 1.0
0.5     0.479           0.479           0.877               0.877
1.0     0.841           0.841           0.54                0.54
1.5     0.997           0.997           0.07                0.07
--------------------------------------------------------------------------------
```

Problem 5.2w

An electricity board charges different rates for different categories of consumption of power. Category 1 is domestic users, category 2 is educational institutions, category 3 is commercial institutions and category 4 is industries. The tariff for energy consumption is Rs.1.00 / unit for category 1, Rs.1.75 / unit for category 2, Rs. 2.50 / unit for category 3 and Rs.3.00 / unit for category 4. Write a program to calculate electricity charges for the following data:

Consumer	Category	Unit Consumed
Raman	1	75
Balaji	1	250
Public School	2	800
ABC Hardware	3	550
R.M.K. Industry	4	12450

Program 5.2w

```
/* ------------------------------------------------------
     This program calculates the electricity charges
     for different categories of consumers.

     somasundaramk@yahoo.com
------------------------------------------------------- */

class Prob52
    {
    public static void main(String args [])
        {
        String consumers [] = { "Raman", "Balaji", "Public
               School",  "ABC Hardware",  "R.M.K.Industry"   };
        int units [] =  {75,250, 800, 550, 12450 };
        int conscat [] =  { 1,  1,  2,  3,  4  };
        double rate [] = {1.0, 1.75, 2.50,  3.0  };
        int noc = consumers.length;
        int i;
        double bill = 0;
        //Calculate the electricity charges
        for (i = 0; i < 80; i++)
            System.out.print("-");
        System.out.println("\n");
        System.out.println("\t Consumer \t Category \t Units \t
                         Charges");
        for (i = 0; i < 80; i++)
            System.out.print("-");
        System.out.println("\n");
        for (i = 0; i < noc; i++)
            {
            switch (conscat[i])
                {
                case 1:
                    bill = units[i] * 1.0;
                    break;
                case 2:
                    bill = units[i] * 1.75;
                    break;
                case 3:
                    bill = units[i] * 2.5;
                    break;
```

```
            case 4:
                bill = units[i] * 3.0;
                break;
            default:
                System.out.println("Category mismatch");
            }
        //reduce the number of decimal places
        bill = (int)(bill * 100);
        bill = bill / 100;
        System.out.println(consumers[i] + "\t\t" +
                conscat[i] + "\t" + units[i] + "\t" + bill);
        }
    for (i = 0; i < 80; i++)
        System.out.print("-");
     System.out.println("\n");
    }
}
```

The above program gives the following output:

```
--------------------------------------------------------------------------------
  Consumer       Category              Units    Charges
--------------------------------------------------------------------------------

Raman                 1              75       75.0
Balaji                1              250      250.0
Public School         2              800      1400.0
ABC Hardware          3              550      1375.0
R.M.K.Industry        4              12450    37350.0
--------------------------------------------------------------------------------
```

Exercise–5

I. Fill in the blanks

5.1. if..*else* structure helps to make a _____ from the sequential computation of statements in a program.

5.2. To select one out of many options in a computation, _____ if-else can be used.

5.3. switch statement can be used to select one out _____ options.

5.4. The case value in a switch statement must be _____ .

5.5. The _____ statement tests the condition at the end of the block.

5.6. In a for..loop the condition is tested at the _____ of the block.

5.7. In a for.. statement more than one initializer is allowed. (True/False)

5.8. In a for..statement with more than one initializer, there can be more than one condition. (True/False)

5.9. The break statement can be used only in _____ and _____ statements.

5.10. The _____ statement when encountered takes the control to the end of the loop block.

II. Write programs for the following:

5.11. A cloth shop during festival season offers a discount 10% for purchases made up to Rs.1,000, 12% for purchase value of Rs.1000 or more up to Rs 1,500 and 15% for purchase made for Rs.1,500 or more. Write a program to implement the above scheme for a given sales and print out the sales value, discount and net amount payable by a customer.

5.12. The roots of a quadratic equation

$ax^2+bx+c=0$

are given by :

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \quad ,$$

$$x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a} \quad ,$$

with the condition if $(b^2-4ac) < 0$, then the obsolute value $|(b^2-ac)|$ is to be taken. Given a, b, c, write a Java program to compute the two roots x_1 and x_2 .

5.13 An electric appliance shop assigns code 1 to motor, 2 to fan, 3 to tube light and 4 for wires. All other items have code 5 or more. While selling the goods, a sales tax of 8% to motor, 12% to fan, 5% to tube light, 7.5% to wires and 3% for all other items is charged. A list containing the product code and price is given for making a bill. Write a Java program using switch statement to prepare the bill.

* * * * * *