BCA PART- III

PAPER – XIX

TOPIC: DECISION MAKING AND BRANCHING

PREPARED BY: DR. KIRAN PANDEY

(SCHOOL OF COMPUTER SCIENCE)

EMAIL ID: kiranpandey.nou@gmail.com

Introduction

When a program breaks the sequential flow and jumps to another part of the code, it is called *branching*. When the branching is based on a particular condition, it is known as *conditional branching*. If branching takes place without any decision, it is known as *unconditional branching*.

Java language possesses such decision making capabilities and supports the following statements known as *control* or *decision making* statements.

1. **if** statement
2. **switch** statement
3. conditional operator statement

In this unit, we shall discuss the features, capabilities and applications of these statements which are classified as *selection* statements.

Decision Making with If Statement

The **if** statement is a powerful decision making statement and is used to control the flow of execution of statements. It is basically a *two-way* decision statement and is used in conjunction with an expression. It takes the following form:

```
if (test expression)
```

It allows the computer to evaluate the *expression* first and then, depending on whether the value of the *expression* (relation or condition) is 'true' or 'false', it transfers the control to a particular statement. This point of program has two paths to follow, one for the ***true*** condition and the other for the ***false*** condition as shown in Fig. 4.1

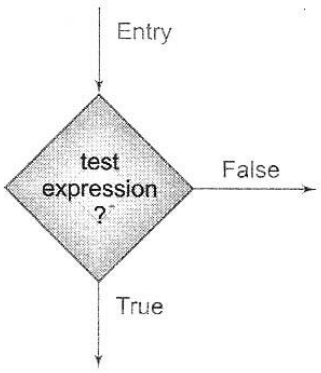


Fig. 4.1 Two-way branching

Some examples of decision making, using **if** statement are:

```
1. if (balance is zero)
   borrow money

2. if (room is dark)
   put on lights

3. if (code is 1)
   person is male

4. if (age is more than 55)
   person is retired
```

The **if** statement may be implemented in different forms depending on the complexity of conditions to be tested.

1. Simple `if` statement
2. `if..else` statement
3. Nested `if..else` statement
4. `else if` ladder

Simple If Statement

The general form of a simple **if** statement is

```
if(test expression)
{
   Statement-block;p
}
Statement-x
```

The 'statement-block' may be a single statement or a group of statements. If the *test expression* is true, the *statement-block* will be executed; otherwise the statement-block will be skipped and the execution will jump to the *statement-x*.

It should be remembered that when the condition is true both the statement-block and the statement-x are executed in sequence. This is illustrated in Fig. 4.2

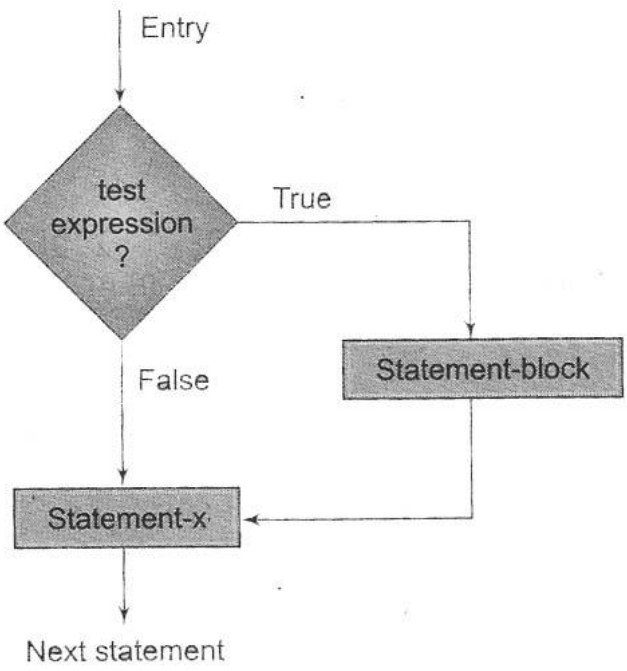


Fig. 4.2 Flowchart of simple if control

Consider the following segment of a program that is written for processing of marks obtained in an entrance examination.

```
.............
.............
if (category == SPORTS)
{
   marks = marks + bonus_marks;
}
System.out.println(marks);
.............
.............
```

The program tests the type of category of the student. If the student belongs to the SPORTS category, then additional bonus_marks are added to his marks before they are printed. For others, bonus_marks are not added.

Consider a case having two test conditions, one for weight and another for height. This is done using the compound relation

```
if (weight < 50 && height > 170) count = count +1;
```

This would have been equivalently done using two **if** statements as follows:

```
if(weight<50)
       if(height>170)
              count = count +1;
```

If the value of **weight** is less than **50,** then the following statement is executed, which in turn is another **if** statement. This if statement tests **height** and if the **height** is greater than **170,** then the **count** is incremented by **1.** Program 4.1 illustrates the implementation of the above statement.

Program 4.1 Counting with if statement

```
class IfTest
{
   public static void main(String arg[])
   {
      int i, count, count1, count2;
      float[] weight = { 45.0F, 55.0F, 47.0F, 51.0F, 54.0F };
      float[] height = { 176.5F, 174r.2F, 168.0F, 170.7F, 169.0F };
      count   =   0;
      count1  =   0;
      count2  =   0;
      for (I  =   0; <= 4; i++)
      {
         If(weight[i] < 50.0 && height[i] > 170.0]
         {
```

```
            Count1 = count1 + 1;
          }
          Count = count + 1; // Total persons
       }
       Count 2 = count – count1;
       System.out.println("Number of persons with ...");
       System.out.println("Weight<50 and height>170 = " +count1);
       System.out.print("Others = " + count2);
    }
}
```

The output of Program 4.1 will be:

```
Number of persons with ...
Weight < 50 and height > 170 = 1
Others = 4
```

The If…Else Statement

The **if….else** statement is an extension of the simple **if** statement. The general form is

```
if(test expression)
{
    True-block statement(s)
}
else
{
    False-block statement(s)
}
statement-x
```

If the *test expression* is true, then *the true-block statement(s)* immediately following the **if** statement, are executed; otherwise, the *false-block statement(s)* are execute. In either *true-block* or *false-block* will be executed, not both. This is illustrated in Fig. 4.3. In both the cases, the control is transferred subsequently to be *statement-x*.

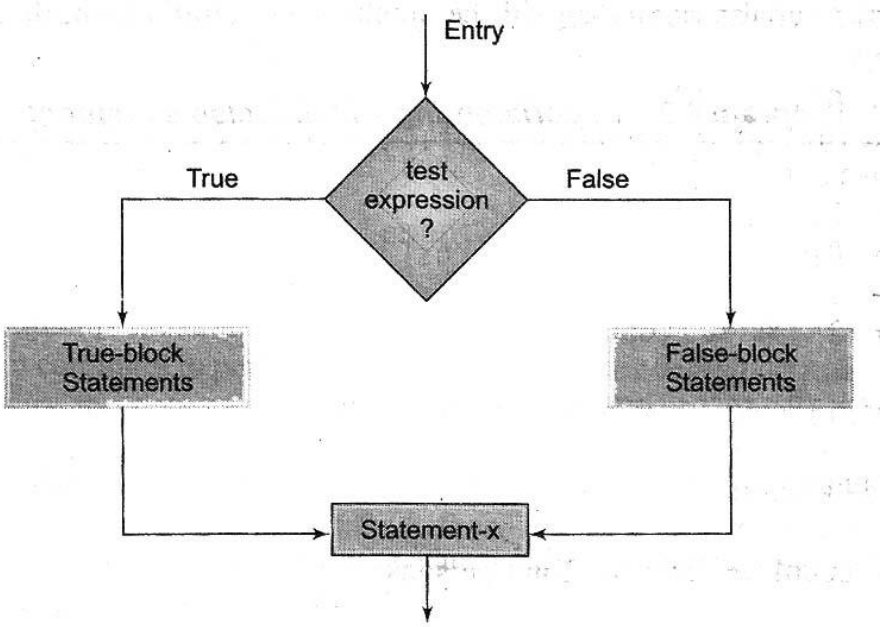


Fig. 4.3 Flowchart of if….else control

Let us consider an example of counting the number of boys and girls in class. We use code 1 for a boy and 2 for a girl. The program statements to do this may be written as follows:

```
.....
.....
if(code == 1)
    boy = boy + 1
if(code == 2)
    girl = girl + 1;
.....
.....
```

The first determines whether or not the student is a boy. If yes, the number of boys is increased by 1 and the program continues to the second test. The second test again determines whether the students is a girl. This is unnecessary. Once a students is identified as a boy, there is no need to test again for a girl. A student can be either a boy or girl, not both. The above program segment can be modified using the **else** as follows:

```
.....
.....
if(code == 1)
    boy = boy + 1;
else
    girl = girl + 1;
xxx;
.....
```

Here, if the code is equal to 1, the statement **boy = boy + 1;** is executed and the control is transferred to the statement **xxx,** after skipping the **else** part. If the code is not equal to 1, the statement **boy = boy + 1;** is skipped and the statement in the **else** part **girl = girl + 1;** is executed before the control reaches the statement **xxx**.

Program 4.2 counts the even and odd numbers in a list of numbers using the **if….else** statement. **number[ ]** is an array variable containing all the numbers and **number.length** gives the number of elements in the array.

Program 4.2 Experimenting with if….else statement

```
class
{
  public static void main(String args[])
  {
    int number[] = { 50, 65, 56, 71, 81 };
    int even = 0, odd = 0;
    for (int i = 0; i < number.length; i++)
    {
       if ((number[i] % 2) == 0) // Decide even or odd
       {
          even += 1; // counting EVEN numbers
       }
       else
       {
```

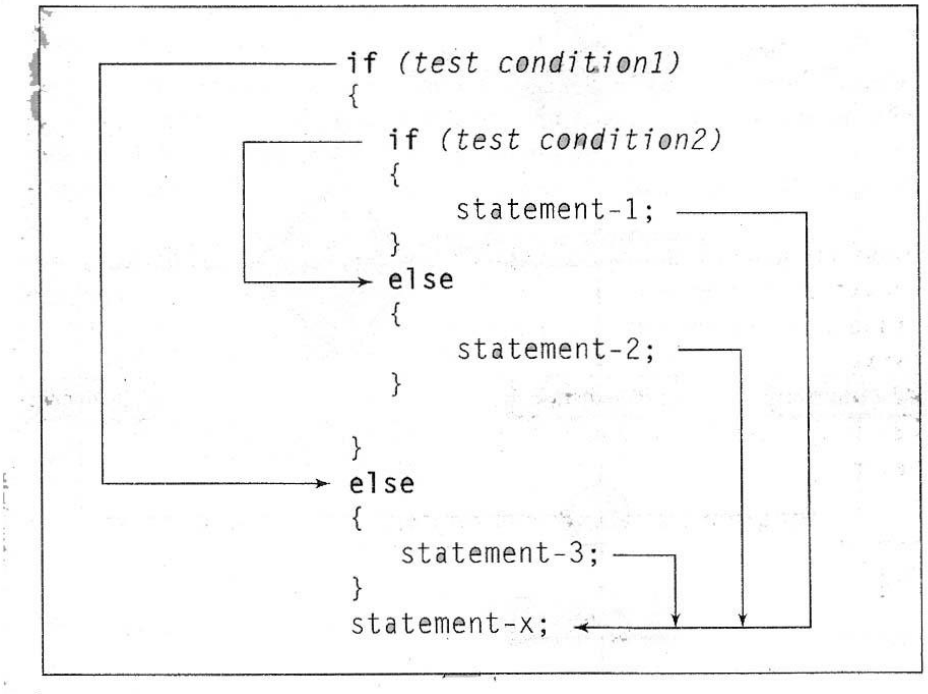
```
            Odd += 1; // counting ODD numbers
        }
    }
    System.out.println("EVEN Numbers ; " + even +
            " Odd Numbers ; " + odd);
  }
}
```

Output of Program 4.2

```
Even Numbers : 2 Odd Numbers : 3
```

Nesting of If…..Else Statements

When a series of decisions are involved, we may have to use more than one **if…..else** statement is *nested* form as follows:



The logic of execution is illustrated in Fig. 4.4. If the *condition-1* is false, the statement-3 will be executed; otherwise it continues to perform the second test. If the *condition-2* true, the statement-1 will be evaluated; otherwise the statement-2 will be evaluated and then the control is transferred to statement-x.

A commercial bank has introduced an incentive policy of giving bonus to all its deposit holders. The policy is as follows: A bonus of 2 per cent of the *balance* held on 31st December is given to everyone, irrespective of their balances, and 5 per cent is given to female account holders if their balance is more than Rs. 5000. This logic can be coded as follows:

```
.....
if(sex is female)
{
   If (balance > 5000)
     bonus = 0.05 * balance;
   else
     bonus = 0.02 * balance;
}
else
{
   bonus = 0.02 * balance;
```

```
}
balance = balance + bonus;
.............
.............
```

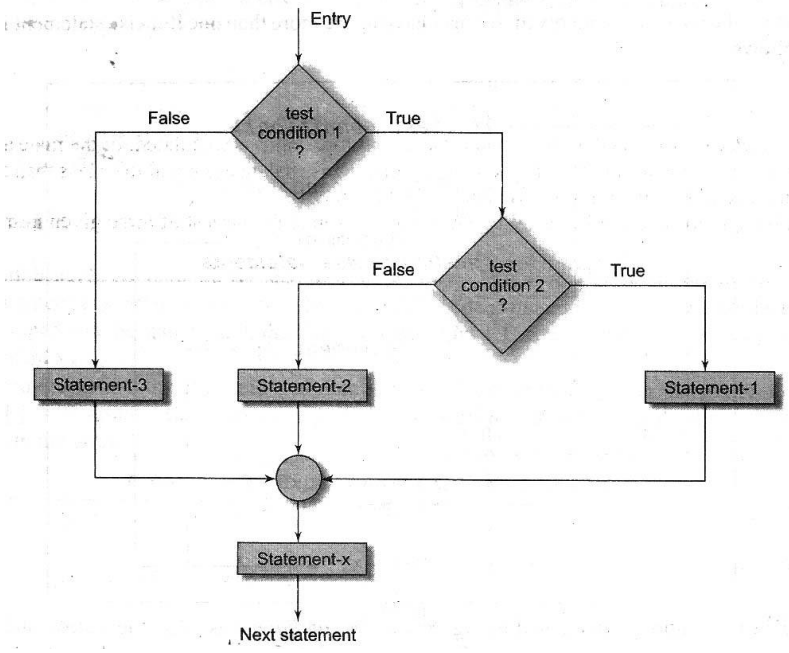


Fig. 4.4 Flowchart of nested if….else statements

When nesting, care should be exercised to match every **if** with an **else**. Consider the following alternative to the above program (which looks right at the first sight);

```
if(sex is female)
  if(balance > 5000)
    bonus = 0.05 * balance;
else
  bonus = 0.02 * balance;
  balance = balance + bonus;
```

There is an ambiguity as to over which **if** the **else** belongs to. In Java an **else** is linked to the closest non-terminated if. Therefore, the **else** is associated with the inner **if** and there is no **else** option for the outer **if.** This means that the computer is trying to execute the statement.

```
balance = balance + bonus;
```

without really calculating the bonus for the male account holders.

Consider another alternative, which also looks correct:

```
if(sex is female)
{
  if (balance > 5000)
     bonus = 0.05 * balance;
}
else
  bonus = 0.02 * balance;
  balance = balance + bonus;
```

In this case, **else** is associated with the outer **if** and therefore bonus is calculated for the male account holders. However, bonus for the female account holders, whose balance is equal to or less than 5000 is not calculated because of the missing **else** option for the inner **if**.

Program 4.3 employs nested **if…..else** statements to determine the largest of three given numbers.

Program 4.3 Nesting if…else statements

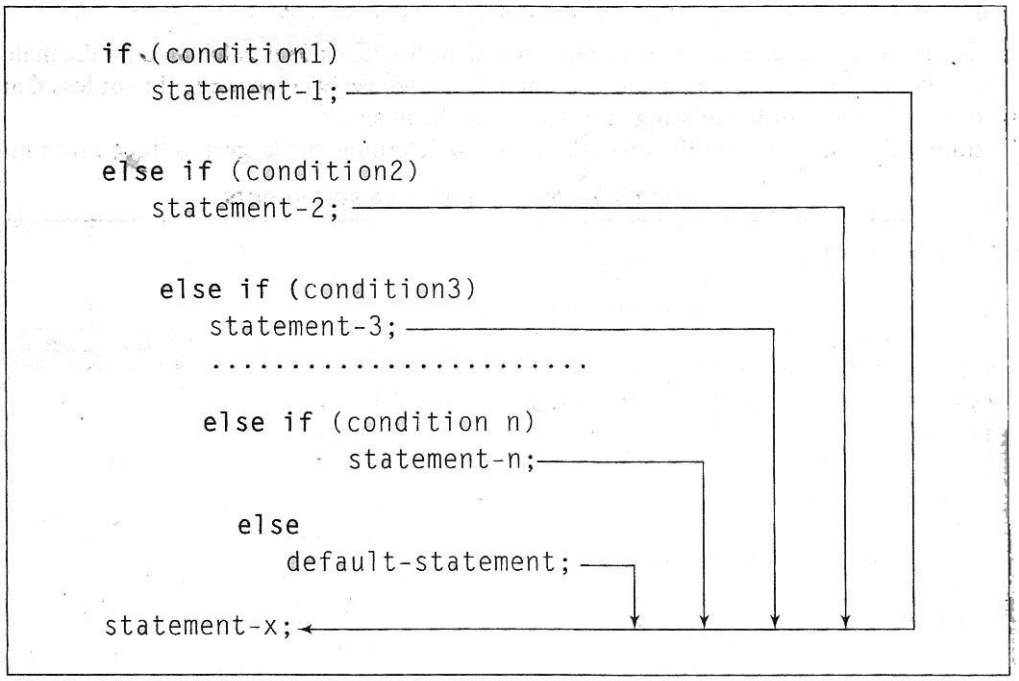
```
class IfElseNesting
{
   public static void main(String args[])
   {
      int a = 325, b = 712, c = 478;
      System.out.print("Largest value is : ");
      if (a > b)
      {
         if (a > c)
         {
            System.out.println(a);
         }
         else
         {
            System.out.println(c);
         }
      }
      else
      {
         if (c > b)
         {
            System.out.println(c);
         }
         else
         {
            System.out.println(b);
         }
      }
   }
}
```

Output of Program 4.3

```
Largest value is : 712
```

The Else If Ladder

There is another way of putting ifs together when multipath decisions are involved. A multipath decision is a chain of ifs in which the statement associated with each **else** is an **if.** It takes the following general form:



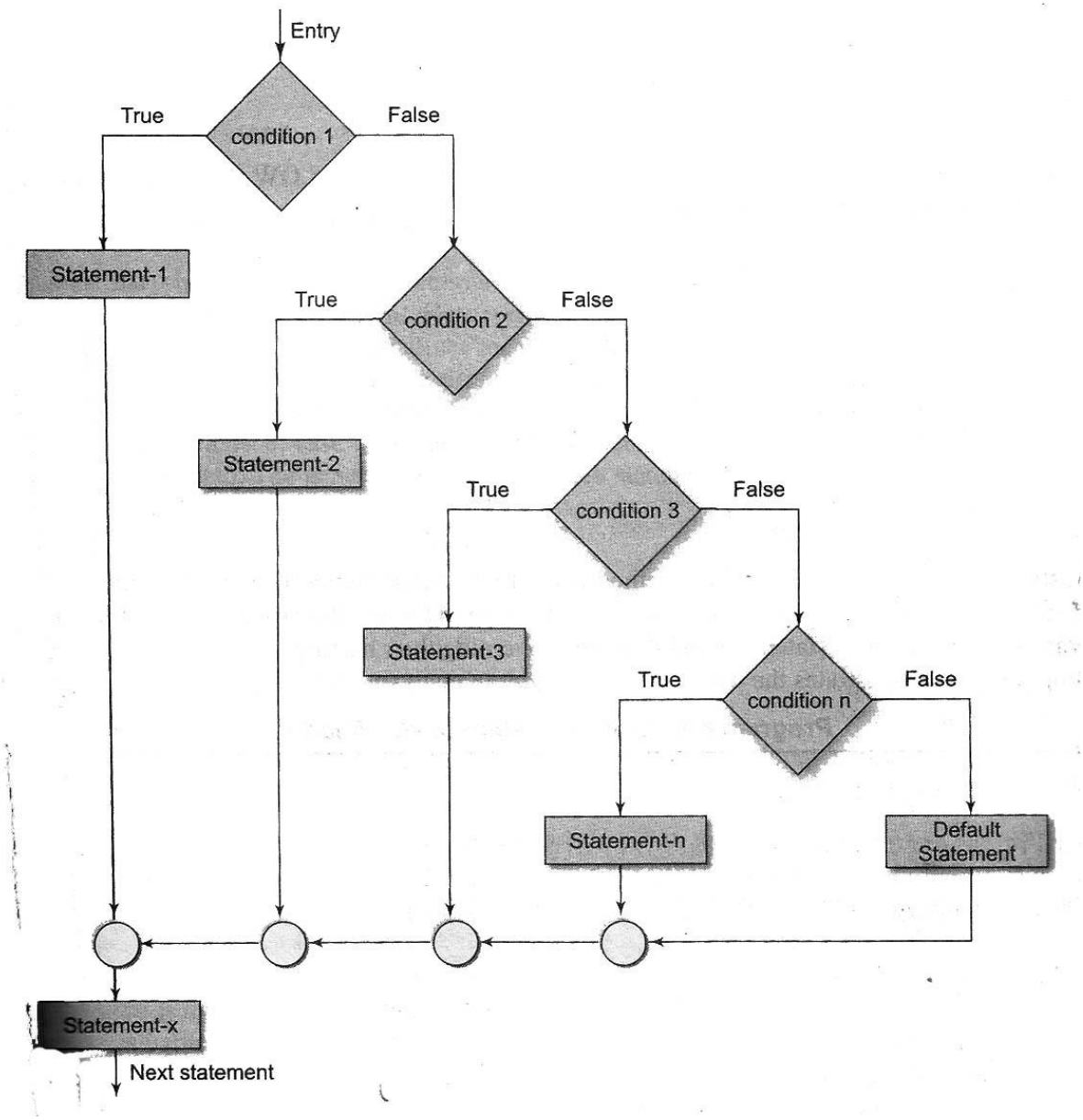
This construct is known as the **else if** ladder. The conditions are evaluated from the top (of the ladder), downwards. As soon as the true condition is found, the statement associated with it is executed and the control is transferred to the *statement-x* (skipping the rest of the ladder). When all the n conditions become false, then the final **else** containing the *defeault-statement* will be executed. Figure 4.5 shows the logic of execution of **else if** ladder statements.

Let us consider an example of grading the students in an academic institution. The grading is done according to the following rules:

Average marks	Grade
80 to 100	Honours
60 to 79	First Division
50 to 59	Second Division
40 to 49	Third Division
0 to 39	Fail

This grading can be done using the **else if** ladder as follows:

```
if(marks > 79)
      grade = "Honours";
      else if(marks > 59)
          grade = "First Division";
          else if(marks > 49)
              grade = "Second Division";
              else if(marks > 39)
              grade = "Third Division";
              else
              grade = "Fail";
      System.out.println("Grade: " + grade);
```



4.5 Flowchart of else….if ladder

Consider another example given below:

```
.............
.............
if (code == 1)
    colour = "RED";
else if (code == 2)
    colour = "GREEN";
    else if (code == 3)
           code = "WHITE";
           else
           colour = "Yellow";
.............
.............
```

Code numbers other than 1, 2 or 3 are considered to represent YELLOW colour. The same results can be obtained by using nested **if….else** statements.

```
if(code != 1)
    if (code != 2)
    if (code != 3)
     colour = "YELLOW"
    else
     colour = "WHITE"
    else
     colour = "GREEN";
else
    colour = "RED";
```

In such situations, the choice of the method is left to the programmer. However, in order to choose an **if** structure that is both effective and efficient, it is important that the programmer is fully aware of the various forms of an **if** statement and the rules governing their nesting.

Program 4.4 demonstrates the use of **if….else** ladder in analysing a mark list.

Program 4.4 Demonstration of else if ladder

```
class ElseIfLadder
{
   public static void main(String args[])
   {
      int rollNumber[] = { 111, 222, 333, 444 };
      int marks[] = { 81, 75, 43, 58 };
      for (int i = 0; 1 < rollNumber.length.; i++)
      {
         if (marks[i] > 79)
            System.out.println(rollNumber[i] + "" Honours");
      else if (marks[i] > 59)
            System.out.println(rollNumber[i] + " I Division");
      else if (marks[i] > 49)
            System.out.println(rollNumber[i] + " II Division");
         else
            System.out.println(rollNumber[i] + " FAIL");
      }
   }
}
```

Program 4.4 produces the following output:

```
111 Honours
222 I Division
333 FAIL
444 II Division
```

The Switch Statement

We have seen that when one of the many alternatives is to be selected, we can design a program using **if** statements to control the selection. However, the complexity of such a program increases dramatically when the number of alternative increases. The program becomes difficult to read and follow. At times, it may confuse even the designer of the program. Fortunately, Java has a built-in multiway decision statement known as **switch**. The **switch**

statement tests the value of a given variable (or expression) against a list of **case** values and when a match is found, a block of statements associated with that **case** is executed. The general form of the **switch** statement is as shown below:

```
Switch (expression)
{
      Case value-1;
                block-1
                break;
      case value-2;
                block-2
                break;
      .........
      .........
                default-break
                break;
}
statement-x;
```

The *expression* is an integer expression or characters. *Value-1, value-2* …. Are constants or constant expressions (evaluable to an integral constant) and are known as *case labels*. Each of these values should be unique within a **switch** statement. *block-1, block-2* …. Are statement lists and may contain zero or more statements. There is no need to put braces around these blocks but it is important to note that case labels end with a colon (;).

When the switch is executed, the value of the expression is successively compared against the values *value-1*, *value-2*, ….. If a **case** is found whose value matches with the value of the expression, then the block of statements that follows the case are executed.

The **break** statement at the end of each block signals the end of a particular case and causes an exit from the **switch** statement, transferring the control to the *statement-x* following the **switch**.

The **default** is an optional case. When present, it will be executed if the value of the expression does not match with any of the case values. If not present, no action takes place when all matches fail and the control goes to the *statement-x*.

The selection process of **switch** statement is illustrated in the flowchart shown in Fig. 4.4.

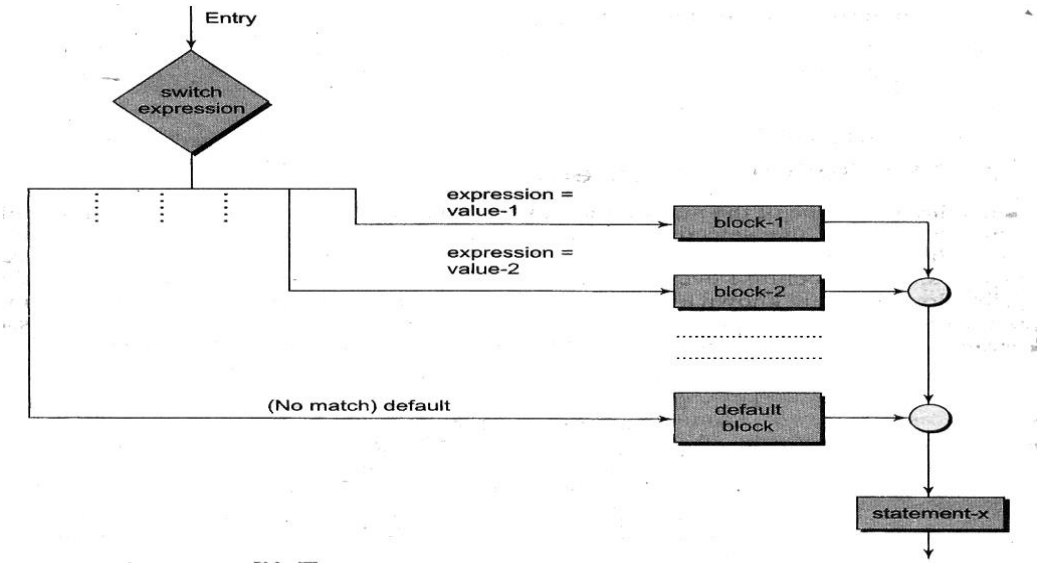


Fig. 4.6 Selection process of the switch statement

The **switch** statement can be used to grade the students as discussed in the last section. This is illustrated below:

```
.............
.............
index = marks/10;
switch(index)
{
case 10;
case 9;
case 8;
      grade = "Honours";
      break;
case 7;
case 6;
      grade = "First Division";
      break;
case 5;
      grade = "Second Division";
      break;
case 4;
      grade = "Third Division";
      break;
default;
      grade = "Fail";
      break;
}
System.out.println(grade);
.............
.............
```

Note that we have used a conversion statement

```
index = marks/10;
```

where, index is defined as an integer. The variable index takes the following integer values.

Marks	Index
100	10
90 – 99	9
80 – 89	8
70 – 79	7
60 – 69	6
50 – 59	5
40 – 49	4
30 – 39	3
20 – 29	2
10 – 19	1
0 – 9	0

The segment of the program illustrates two important features. First, it uses empty cases. The first three cases will execute the same statements

```
grade = "Honours";
break;
```

Same is the case with case 7 and 6. Second, default condition is used for all other cases where marks is less than 40.

Program 6.5 illustrates the use of **switch** for designing a menudriven interactive program.

Program 4.5 Testing the switch ( )

```
class
{
   public static void main(String args[])
   {
      char choice;
      System.out.println("Select your choice");
      System.out.println("  M -> Madras");
      System.out.println("  B -> Bombay");
      System.out.println("  C -> Calcutta");
      System.out.println("Choice ---> ");
      System.out.flust( );
      try
      {
         switch (choice = (char) System.in.read())
         {
            case 'M';
            case 'm'; System.out.println("Madras : Booklet 5");
                      break;
         case 'B';
         case 'b';    System.out.println("Bombay : Booklet 9");
                      break;
         case 'C';
         case 'c';    System.out.println("Calcutta : Booklet 15");
                      break;
         }
      }
      catch (Exception e)
      {
         System.out.print("I/O Error");
      }
   }
}
```

Output of the Program 4.5

```
Run1
Select your choice
    M ---> Madras
    B ---> Bombay
    C ---> Calcutta
Choice ---> m
Madras : Booklet 5
Run2
Select your choice
    M ---> Madras
    B ---> Bombay
    C ---> Calcutta
Choice ---> m
Madras : Booklet 5
Run3
Select your choice
    M ---> Madras
    B ---> Bombay
    C ---> Calcutta
Choice ---> m
Madras : Booklet 15
```

The ? : Operator

The Java language has an unusual operator, useful for making two-way decisions. This operator is a combination of ? and :, and takes three operands. This operator is popularly known as the conditional operator. The general form of use of the *conditional operator* is as follows:

```
conditional expression ? expression1 : expression2
```

The *conditional expression* is evaluated first. If the result is true, *expression1* is evaluated and is returned as the value of the conditional expression. Otherwise, *expression2* is evaluated and its value is returned. For example, the segment

```
if (x < 0)
    flag = 0;
else
    flag = 1;
```

can be written as

```
    flag = (x<0) ? 0 : 1;
```

Consider the evaluation of the following function;

```
y = 1.5x + 3     for x <= 2
y = 2x + 5       for x > 2
```

This can be evaluated using the conditional operator as follows:

```
y = (x>2) ? (2*x+5) : (1.5*x+3);
```

The conditional operator may be nested for evaluating more complex assignment decisions.