Chapter 4: Control Structures I

Java Programming:

From Problem Analysis to Program Design,

Second Edition

Chapter Objectives

- Learn about control structures.
- Examine relational and logical operators.
- Explore how to form and evaluate logical (Boolean) expressions.
- Learn how to use the selection control structures `if`, `if...else`, and `switch` in a program.

Control Structures

- Three methods of processing a program:
  - In sequence
  - Branching
  - Looping
- Branch: Altering the flow of program execution by making a selection or choice.
- Loop: Altering the flow of program execution by repeating statements.

Control Structures

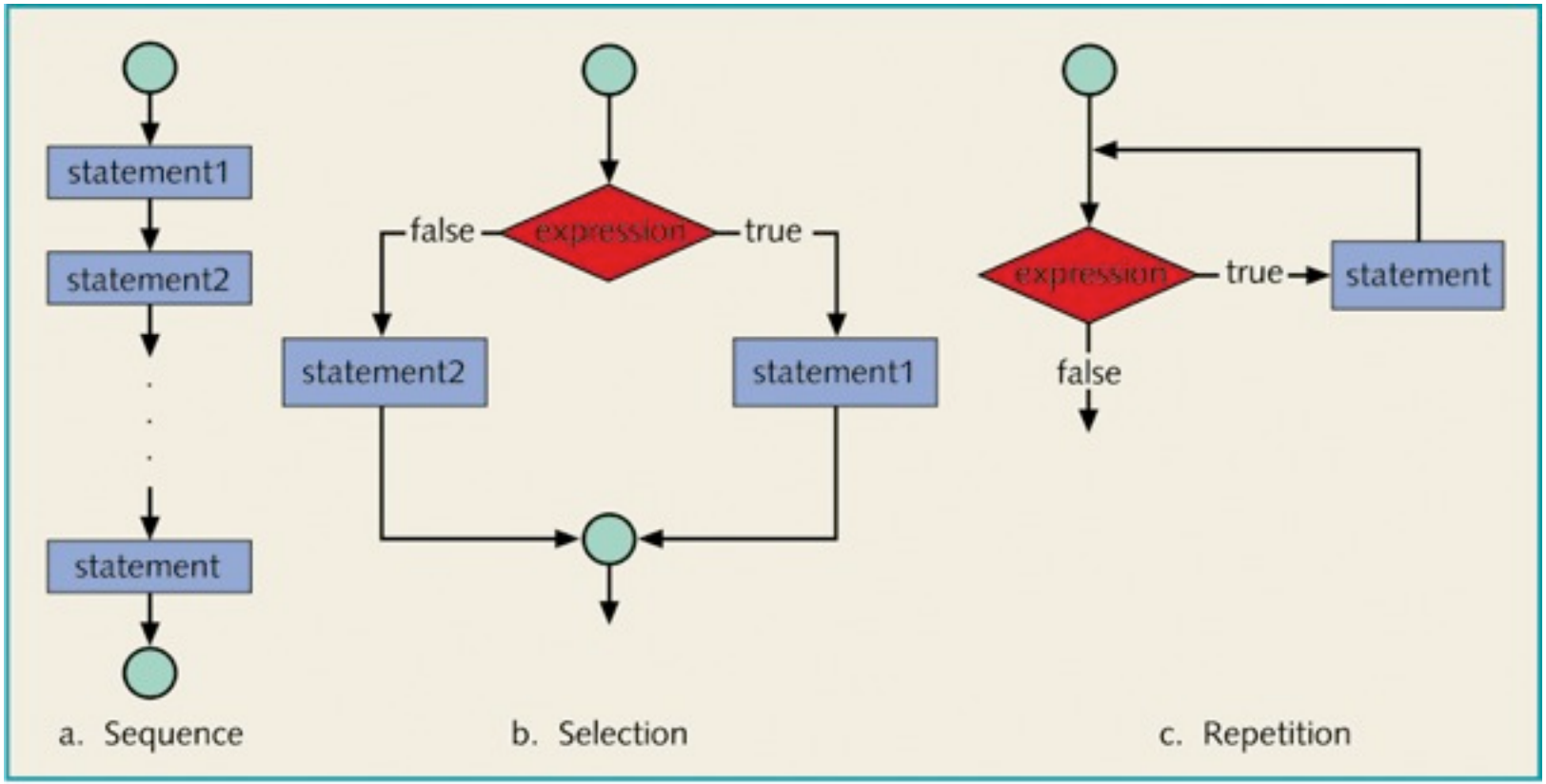


Figure 4-1 Flow of execution

Relational Operators

- Relational operator:
  - Allows you to make comparisons in a program.
  - Binary operator.
- Condition is represented by a logical expression in Java.
- Logical expression: An expression that has a value of either true or false.

Relational Operators

Table 4-1 Relational Operators in Java

Operator	Description
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Relational Operators and Primitive Data Types

- Can be used with integral and floating-point data types.
- Can be used with the char data type.
- Unicode Collating Sequence.
- 8 < 5 always evaluates to false.
- 8 < '5' always evaluates to true. //'5'= 53

Relational Operators and Primitive Data Types

Table 4-2 Evaluating Expressions Using Relational Operators and the Unicode (ASCII) Collating Sequence

Expression	Value of the Expression	Explanation
`' ' < 'a'`	`true`	The Unicode value of `' '` is 32, and the Unicode value of `'a'` is 97. Because 32 < 97 is `true`, it follows that `' ' < 'a'` is `true`.
`'R' > 'T'`	`false`	The Unicode value of `'R'` is 82, and the Unicode value of `'T'` is 84. Because 82 > 84 is `false`, it follows that `'R' > 'T'` is `false`.
`'+' < '*'`	`false`	The Unicode value of `'+'` is 43, and the Unicode value of `'*'` is 42. Because 43 < 42 is `false`, it follows that `'+' < '*'` is `false`.
`'6' <= '>'`	`true`	The Unicode value of `'6'` is 54, and the Unicode value of `'>'` is 62. Because 54 <= 62 is `true`, it follows that `'6' <= '>'` is `true`.

Comparing Strings

- Strings are compared character by character, using the collating sequence, until one of three conditions is met:
  1. A mismatch is found.
  2. One string is exhausted.
  3. The last characters have been compared and are equal.

Comparing Strings

For example,

- `"Air" < "Big" // because 'A' < 'B'`
- `"Air" < "An" // because 'i' < 'n'`
- `"Hello" < "hello" // because 'H' < 'h'`
- `"Bill" < "Billy"`

Comparing Strings

- Strings can not be compared with the usual <, <=, >, or >= operators,
- and the == and != operators don't compare the characters in the strings.

Comparing Strings

- class String
  - Method `compareTo` (`<0` , `0` , `>0`)

- Given string `str1` and `str2`

$$\texttt{str1.compareTo(str2)} = \begin{cases} \text{an integer} < 0 \text{ if string } \texttt{str1} < \texttt{str2} \\ 0 \text{ if string } \texttt{str1} \text{ is equal to string } \texttt{str2} \\ \text{an integer} > 0 \text{ if string } \texttt{str1} > \texttt{str2} \end{cases}$$

Comparing Strings

```
String str1 = "Hello";
String str2 = "Hi";
String str3 = "Air";
String str4 = "Bill";
String str5 = "Bigger";
```

Table 4-3 Comparing Strings with the Method `compareTo`

Expression	Value	Explanation
`str1.compareTo(str2)`	`< 0`	`str1 = "Hello"` and `str2 = "Hi"`. The first character of `str1` and `str2` are the same, but the second character `'e'` of `str1` is less than the second character `'i'` of `str2`. Therefore, `str1.compareTo(str2)< 0`.
`str1.compareTo("Hen")`	`< 0`	`str1 = "Hello"`. The first two characters of `str1` and `"Hen"` are the same, but the third character `'l'` of `str1` is less than the third character `'n'` of `"Hen"`. Therefore, `str1.compareTo("Hen") < 0`.

Comparing Strings

Table 4-3 Comparing Strings with the Method `compareTo` (continued)

Expression	Value	Explanation
`str4.compareTo(str3)`	`> 0`	`str4 = "Bill"` and `str3 = "Air"`. The first character `'B'` of `str4` is greater than the first character `'A'` of `str3`. Therefore, `str4.compareTo(str3) > 0`.
`str1.compareTo("hello")`	`< 0`	`str1 = "Hello"`. The first character `'H'` of `str1` is less than the first character `'h'` of `"hello"` because the Unicode value of `'H'` is 72, and the Unicode value of `'h'` is 104. Therefore, `str1.compareTo("hello") < 0`.
`str2.compareTo("Hi")`	`= 0`	`str2 = "Hi"`. The strings `str2` and `"Hi"` are of the same length and their corresponding characters are the same. Therefore, `str2.compareTo("Hi") = 0`.
`str4.compareTo("Billy")`	`< 0`	`str4 = "Bill"` has four characters and `"Billy"` has five characters. Therefore, `str4` is the shorter string. All four characters of `str4` are the same as the corresponding first four characters of `"Billy"` and `"Billy"` is the larger string. Therefore, `str4.compareTo("Billy") < 0`.
`str5.compareTo("Big")`	`> 0`	`str5 = "Bigger"` has six characters and `"Big"` has three characters. Therefore, `str5` is the larger string. The first three characters of `str5` are the same as the corresponding first three characters of `"Big"`. Therefore, `str5.compareTo("Big") > 0`.

Comparing Strings

```
public class Example4_2 {
  public static void main(String[] args) {

   String str1 = "Hello";                                    //Line 1
   String str2 = "Hi";                                       //Line 2
   String str3 = "Air";                                      //Line 3
   String str4 = "Bill";                                     //Line 4
   String str5 = "Bigger";                                   //Line 5

   System.out.println("Line 6: " + "str1.compareTo(str2) evaluates to "
                      +  str1.compareTo(str2));              //Line 6

    System.out.println("Line 7: " + "str1.compareTo(\"Hen\") evaluates to "
                      +  str1.compareTo("Hen"));             //Line 7

    System.out.println("Line 8: " +   "str4.compareTo(str3) evaluates to "
                      + str4.compareTo(str3));               //Line 8


    System.out.println("Line 9: " +"str1.compareTo(\"hello\") evaluates to "
                      + str1.compareTo("hello"));            //Line 9

    System.out.println("Line 10: " + "str2.compareTo(\"Hi\") evaluates to "
                      + str2.compareTo("Hi"));               //Line 10

    System.out.println("Line 11: " + "str4.compareTo(\"Billy\") evaluates to "
                      +  str4.compareTo("Billy"));           //Line 11

    System.out.println("Line 12: " + "str5.compareTo(\"Big\") evaluates to "
                      + str5.compareTo("Big"));              //Line 12
```

Comparing Strings

```
Line 6: str1.compareTo(str2) evaluates to -4
Line 7: str1.compareTo("Hen") evaluates to -2
Line 8: str4.compareTo(str3) evaluates to 1
Line 9: str1.compareTo("hello") evaluates to -32
Line 10: str2.compareTo("Hi") evaluates to 0
Line 11: str4.compareTo("Billy") evaluates to -1
Line 12: str5.compareTo("Big") evaluates to 3
```

- Values such as **-4, -2, 1** and so on, are differences of the collating sequence of the first unmatched characters of the string.
- For example:

  in line 6: where, `str1= "Hello", str2="Hi"`
  - **'e' → 101**
  - **'i' →105**
  - **101 - 105 → -4**

Comparing Strings

- In addition to the method `compareTo`, you can use the method `equals` of the `class String`.
- Returns `true` or `false`.
- Example: **`str1 = "Hello", str2= "Hi"`**

```
str1.equals("Hello");   // returns
  true
str1.equals(str2);      //returns false
```

Comparing Strings

- You should use one of the following tests to compare the contents of two strings:
  - `string1.equals(string2)`
  - `string1.compareTo(string2)`

- **Here's the wrong way to do it:**

```
string1 == string2
```

Why wrong?

A comparison of objects (such as Strings) using the == operator doesn't compare the contents of the Strings. Instead, it compares the *address* of the two Strings.

http://www.javabeginner.com/java-string-comparison.htm

Comparing Strings

```
String s = "hi";
```

- `s == "hi"` // true
- `"hi".equals(s)` // true
- `s == new String(s)` // false

Logical (Boolean) Operators

Table 4-4 Logical (Boolean) Operators in Java

Operator	Description
`!`	`not`
`&&`	`and`
`||`	`or`

- **!** is unary operator.
- **&&** is binary operator.
- **||** is binary operator.

Logical (Boolean) Operators

Table 4-5 ! (not) Operator

Expression	!(Expression)
`true`	`false`
`false`	`true`

Example:

`!('A' > 'B')` is `true`.

Because `'A' > 'B'` is `false` ➔ `!('A' > 'B')` is `true`

Logical (Boolean) Operators

This is called the truth table for the operator !

Table 4-5 ! (not) Operator

Expression	!(Expression)
`true`	`false`
`false`	`true`

Example:

`!('A' > 'B')` is `true`.

Because `'A' > 'B'` is `false` ➔ `!('A' > 'B')` is `true`

Logical (Boolean) Operators

Table 4-6 && (and) Operator

Expression1	Expression2	Expression1 && Expression2
true	true	true
true	false	false
false	true	false
false	false	false

Table 4-7 || (or) Operator

Expression1	Expression2	Expression1 || Expression2
true	true	true
true	false	true
false	true	true
false	false	false

Logical (Boolean) Operators Examples

Expression	Value	Explanation
!('A' > 'B')	**true**	Because 'A' > 'B' is **false**, !('A' > 'B') is **true**.
!(6 <= 7)	**false**	Because 6 <= 7 is **true**, !(6 <= 7) is **false**.
(14 >= 5) && ('A' < 'B')	**true**	Because (14 >= 5) is **true**, ('A' < 'B') is **true**, and **true** && **true** is **true**, the expression evaluates to **true**.
(24 >= 35) && ('A' < 'B')	**false**	Because (24 >= 35) is **false**, ('A' < 'B') is **true**, and **false** && **true** is **false**, the expression evaluates to **false**.

Order of Precedence

```
11 + 5 <= 9 || 6 < 15 && 7 >= 8
```

which to solve first:

arithmetic, relational or logical ?

Order of Precedence

```
11 + 5 <= 9 || 6 < 15 && 7 >= 8
```

which to solve first:

arithmetic, relational or logical ?

Operators	Precedence
!, +, – (unary operators)	first
*, /, %	second
+, –	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

Order of Precedence

- For more complex expressions:

Unary operators:	++, --, !, unary - and +, type-cast
Multiplication and division:	*, /, %
Addition and subtraction:	+, -
Relational operators:	<, >, <=, >=
Equality and inequality:	==, !=
Boolean and:	&&
Boolean or:	||
Conditional operator:	?:
Assignment operators:	=, +=, -=, *=, /=, %=

Operators on the same line have the same precedence. When they occur together, unary operators and assignment operators are evaluated right-to-left, and the remaining operators are evaluated left-to-right. For example:

`A*B/C` means **`(A*B)/C`**, while **`A=B=C`** means **`A=(B=C)`**.

Precedence Rules

Example 4_6: Evaluate the following expression:

```
  (17 < 4*3+5)||(8*2 == 4*4) && !(3+3 == 6)
= (17 < 12+5)||(16 == 16) && !(6 == 6)
= (17 < 17) || true && ! (true)
= false || true && false
= false || false
= false
```

Precedence Rules

Example: suppose the following declarations:

```
boolean found = true;
boolean flag =false;
double x =5.2;
```

Evaluate:

```
!found  ➔ false
x > 4.0 ➔ true
flag && found ➔ false
```

Go through example 4_8

Short-Circuit Evaluation

Suppose:

- `int age = 25;`
- `char grade='B';`

- `(age >= 21) || ( 3 + 8 == 5)`

  Because `(25 >= 21)` is **`true`** and the operator used is `||`,due to short-circuit evaluation the computer does not evaluate `( 3 + 8 == 5)`.

- `(grade == 'A') && (3 - 2 >= 7)`

  Because `('B' == 'A')` is **`false`** and the operator used is **`&&`**,due to short-circuit evaluation the computer does not evaluate `( 3 - 2 >= 7)`.

Short-Circuit Evaluation

- A process in which the computer evaluates a logical expression from left to right and stops as soon as the value of the expression is known.

- If the operators **|** and **&** are used, NO short circuit evaluation is used.

Selection

- One-way selection
- Two-way selection
- Compound (block of) statements
- Multiple selections (nested if)
- Conditional operator
- `switch` structures

One-Way Selection

- Syntax:

```
if (expression)
     statement ;
```

Must be in ( ) and no ;

put ; after end of statement

- Expression referred to as decision maker.
- If the value of the `expression` is `true` ➔ `statement` executes.
- If the value of the `expression` is `false` ➔ `statement` does not executes.

One-Way Selection

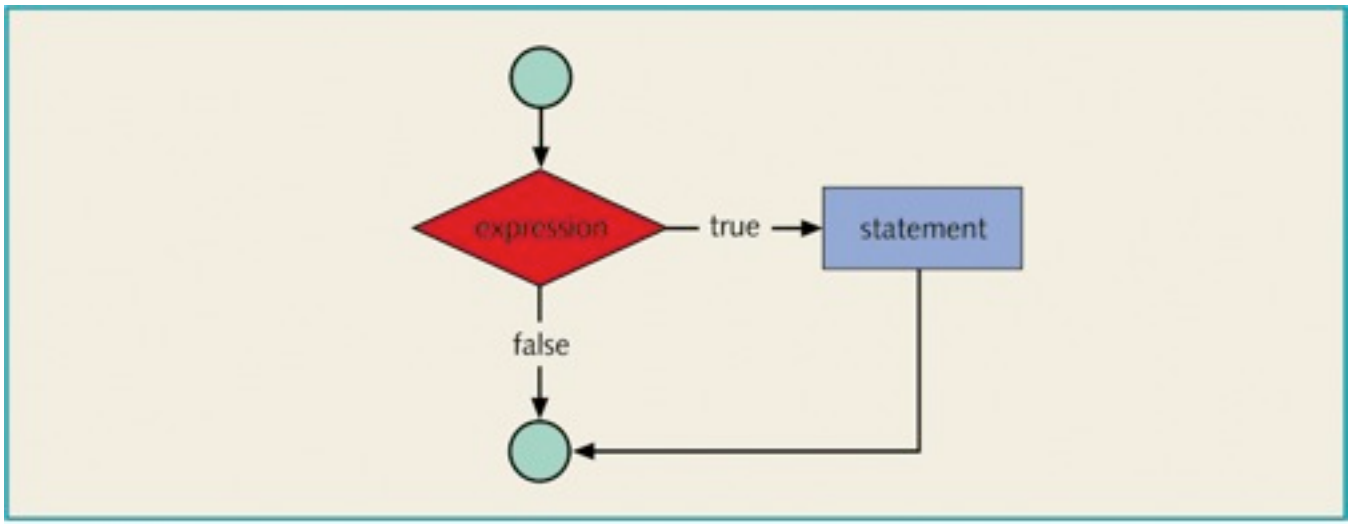


Figure 4-4 One-way selection

Example:

```
char grade='';
if ( score >= 90 )
 grade = 'A';
```

Example 4-11

```
//Determine the absolute value of an integer
import java.util.*;
public class AbsoluteValue
{   static Scanner console = new Scanner(System.in);
   public static void main(String[] args)
   {
      int number;
      int temp;
      System.out.println("Enter an integer:");      //Line 1
      number = nextInt();                          //Line 2
      temp = number;                               //Line 3
      if (number < 0)                                 //Line 4
          number = -number;                           //Line 5

     System.out.println("The absolute value of " + temp+ " is " +
   number+"Absolute Value");
   }
```

Two-Way Selection

- Syntax:

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

- `else` statement must be paired with an `if`.

Two-Way Selection

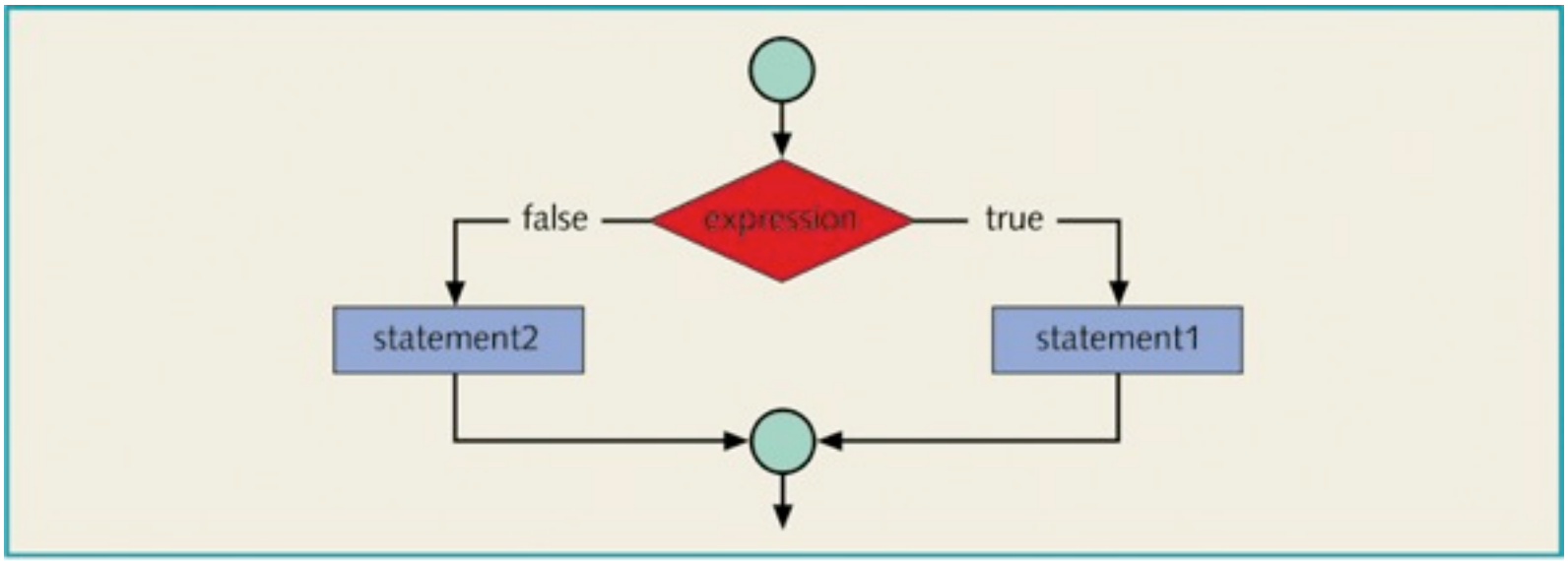


Figure 4-6 Two-way selection

Example:

```
boolean positive, negative;
if (number >= 0 )
    positive = true;
else                          //number < 0
    negative =true;
```

Two-Way Selection

Example 4-14

```
if (hours > 40.0)   // includes overtime payment
    wages = 40.0 * rate +
            1.5 * rate * (hours - 40.0);
else
    wages = hours * rate;
```

Given that `rate = 100`, what `wages` will be if :

a) `hours = 50`

b) `hours = 30`

Two-Way Selection

Example 4-15

```
if (hours > 40.0);                                    //Line 1
    wages = 40.0 * rate +
            1.5 * rate * (hours - 40.0);              //Line 2
else                                                  //Line 3
    wages = hours * rate;                             //Line 4
```

• Because a semicolon follows the closing parenthesis of the `if` statement (Line 1), the `else` statement stands alone. The semicolon at the end of the `if` statement (see Line 1) ends the `if` statement, so the statement at Line 2 separates the else clause from the `if` statement. That is, `else` is by itself. Because there is no separate `else` statement in Java, this code generates a syntax error.

• For some common errors made by beginning programmers check ex 4_17, 4_18.

Compound (Block of) Statements

Syntax:

```
{
    statement1
    statement2
        .
        .
        .
    statementn
}
```

Compound (Block of) Statements

```
if (age > 18)
{
    System.out.println("Eligible to vote.");
    System.out.println("No longer a minor.");
}
else
{
    System.out.println("Not eligible to vote.");
    System.out.println("Still a minor.");
}
```

Multiple Selection: Nested if

- Syntax:

```
if (expression1)
     statement1;
else
    if (expression2)
         statement2;
    else
         statement3;
```

- `else` is associated with the most recent incomplete `if`.
- Multiple `if` statements can be used in place of `if…else` statements.
- May take longer to evaluate.

Multiple Selection: Nested if

Example 4_20 :

```
if (score >= 90)
  System.out.println ("Grade is A");
else if (score >=80 )
  System.out.println ("Grade is B");
else if (score >=70 )
  System.out.println ("Grade is C");
else if (score >=60 )
  System.out.println ("Grade is D");
else System.out.println ("Grade is
  F");
```

Multiple Selection: Nested if

Example 4_21:

```
if( temperature >= 50 )
  if (temperature >= 80)
    System.out.println ("Good swimming day");
  else
     System.out.println ("Good golfing day");
else
  System.out.println ("Good tennis day");
```

Multiple Selection: Nested if

Example4_22 :

```
if( tempreture >= 50 )
   if (tempreture >= 80)
      System.out.println ("Good swimming day");
    else
      System.out.println ("Good golfing day");
```

Multiple Selection: Nested if

Example4_23 :

```
if ( GPA >= 2.0 )
   if (GPA >= 3.9)
        System.out.println("Dean Honor list");
else
 System.out.println("GPA below graduation requirement");
```

If GPA = 3.8 what will be printed?

Multiple Selection: Nested if

Example4_23 :

```
if ( GPA >= 2.0 )
   if (GPA >= 3.9)
        System.out.println("Dean Honor list");
else
 System.out.println("GPA below graduation requirement");
```

If GPA = 3.8 what will be printed?

```
GPA below graduation requirement
```

Multiple Selection: Nested if

Example4_23 : (rewritten)

```
if ( GPA >= 2.0 )
{
   if (GPA >= 3.9)
        System.out.println("Dean Honor list");
}
else
 System.out.println("GPA below graduation requirement");
```

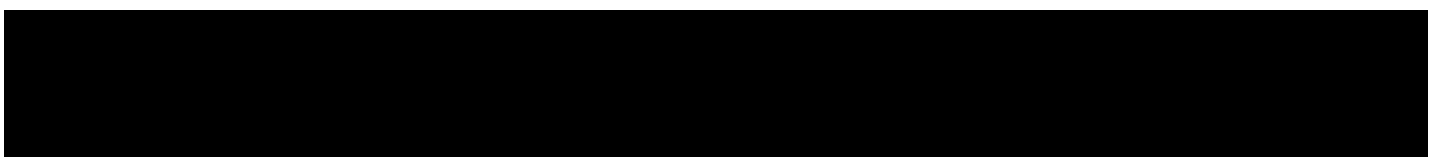
Now, if GPA = 3.8 what will be printed?

Multiple Selection: Nested if

Example4_23 : (rewritten)

```
if ( GPA >= 2.0 )
{
   if (GPA >= 3.9)
        System.out.println("Dean Honor list");
}
else
 System.out.println("GPA below graduation requirement");
```

Now, if GPA = 3.8 what will be printed?



Conditional (? :) Operator

- Ternary operator
- Syntax:

```
expression1 ?  expression2  :
expression3;
```

- If `expression1 = true`, then the result of the condition is `expression2`.

  Otherwise, the result of the condition is `expression3`.

Conditional (? :) Operator

Example :

```
int x = 5 , y =3 , min ;
if ( x <= y )
       min = x ;
else
      min = y ;
```

The above stmt can be written using the conditional operator :

```
 min   =   ( x <= y ) ?    x : y ;
```

`switch` Structures

```
switch (expression)
{
case value1: statements1
             break;
case value2: statements2
             break;
   ...
case valuen: statementsn
             break;
default: statements
}
```

- `expression` is evaluated first.
- `expression` is also known as selector.
- `expression` can be an identifier or an expression and only integral.
- `value` can only be integral.

switch Structures

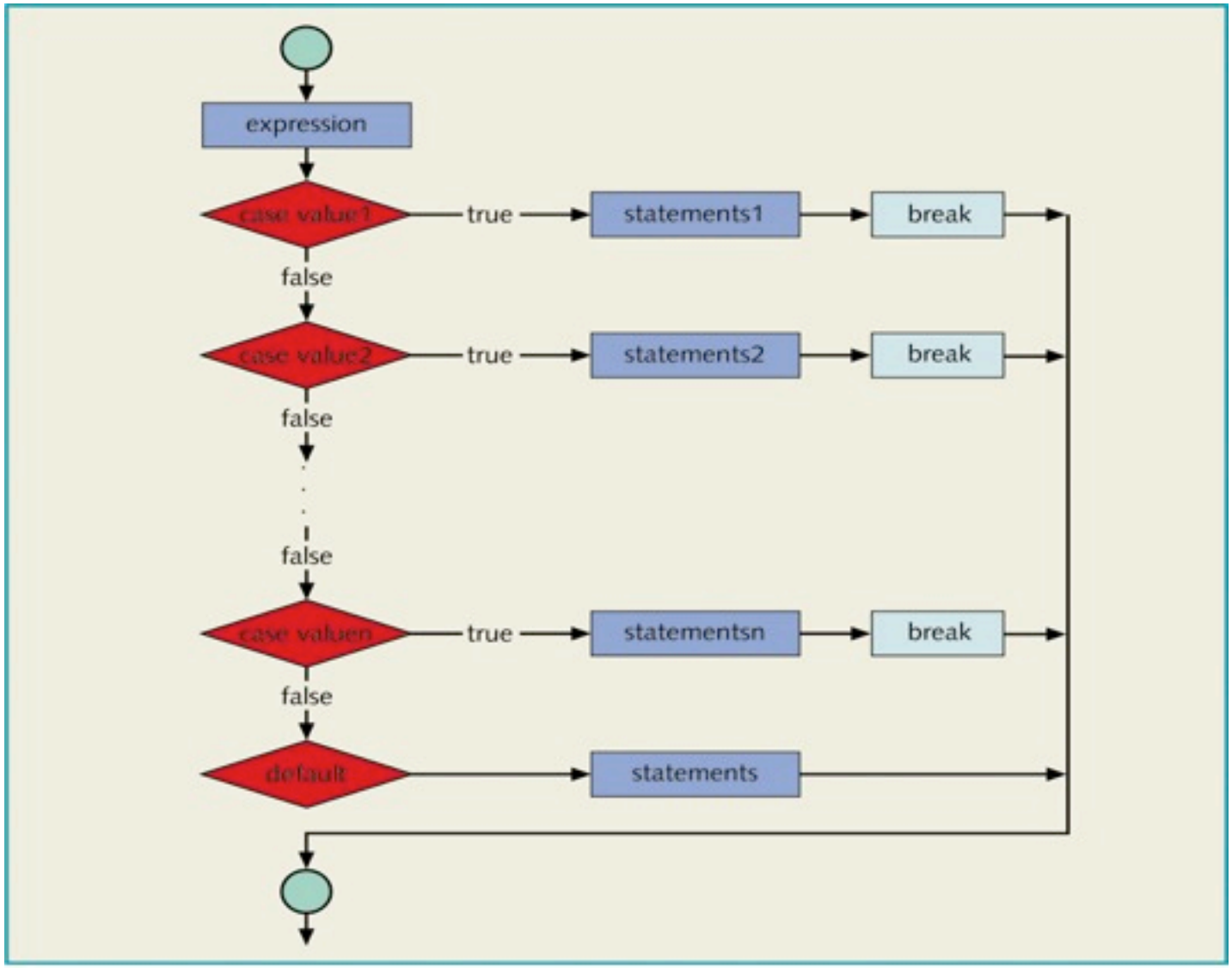


Figure 4-7 switch statement

switch Structures

Example 4-24

```
switch (grade)
{
case 'A': System.out.println("The grade is A.");
          break;
case 'B': System.out.println("The grade is B.");
          break;
case 'C': System.out.println("The grade is C.");
          break;
case 'D': System.out.println("The grade is D.");
          break;
case 'F': System.out.println("The grade is F.");
          break;
default:  System.out.println("The grade is invalid.");
}
```

`switch` Structures

- **`break`** is optional.
- When the value of the **`switch`** expression matches a **`case`** value, all statements execute until a **`break`** is encountered, and the program skips all **`case`** labels in between.

switch Structures

```
import java.util.*;

   public class Example4_25
   {
      static Scanner console = new Scanner(System.in);
      public static void main(String[] args) {

         int num;

         System.out.print("Enter an integer between  0 and 10: ");     //Line 1
         num = console.nextInt();                                       //Line 2

         System.out.println();                                          //Line 3

         System.out.println("\nThe number you entered is " + num);      //Line 4

         switch(num)                                                    //Line 5
         {
         case 0:                                                        //Line 6
         case 1: System.out.print("Hello ");                            //Line 7
         case 2: System.out.print("there. ");                           //Line 8
         case 3: System.out.print("I am ");                             //Line 9
         case 4: System.out.println("Mickey.");                        //Line 10
                  break;                                                //Line 11
         case 5: System.out.print("How ");                             //Line 12
         case 6:                                                        //Line 13
         case 7:                                                        //Line 14
         case 8: System.out.println("are you?");                       //Line 15
                  break;                                                //Line 16
         case 9: break;                                                 //Line 17
         case 10: System.out.println("Have a nice day.");              //Line 18
                  break;                                                //Line 19
         default: System.out.println("Sorry the number is out"
                                     + "of range.");                    //Line 20
         }

         System.out.println("Out of switch structure.");                //Line 21
      }
   }
```

switch Structures

- Sample Run1:

```
Enter an integer between 0 and 10: 0


The number you entered is 0
Hello there. I am Mickey.
Out of switch structure.
```

Sample Run2:

```
Enter an integer between 0 and 10: 9


The number you entered is 9
Out of switch structure.
```

Programming Example: Cable Company Billing

- Input: Customer's account number, customer code, number of premium channels to which customer subscribes, number of basic service connections (in the case of business customers).
- Output: Customer's account number and the billing amount.

Programming Example: Cable Company Billing

Solution:

1. Prompt user for information.
2. Use switch statements based on customer's type.
3. Use an if statement nested within a switch statement to determine the amount due by each customer.

Chapter Summary

- Control structures are used to process programs.
- Logical expressions and order of precedence of operators are used in expressions.
- Compare strings.
- If statements.
- `if`…`else` statements.
- `switch` structures.
- Proper syntax for using control statements.