**Chapter Three**

**Overview of Java Statement**

Any computing problem can be solved by executing a series of actions in a specific order. A

procedure for solving a problem in terms of

1. the **actions** to execute and

2. the **order** in which these actions execute is called an **algorithm**.

Specifying the order in which statements (actions) execute in a program is called **control statement**.

Statements in a program are executed one after the other in the order in which they are written. Java has only **three** kinds of control structures, which from this point forward we refer to as control statements

1. Selection statements,

2. Repetition statements and

3. Transfer statements

**3.1 The Selection statement**

Java provides selection statements that allow the program to choose between alternative actions during execution. The choice is based on criteria specified in the selection statement. These selection statements are

 simple if Statement

 if-else Statement

 The if-else-if Ladder

 switch Statement

**Simple if Statement**

The simple if statement has the following syntax:

if (<conditional expression>)

<statement>

e.g

if ( studentGrade >= 60 ) System.out.println( "Passed" );

**if-else Statement**

The if-else statement has the following syntax:

if (<conditional expression>)

<statement1>

else

<statement2>

It is used to decide between two actions, based on a condition.

The <conditional expression> is evaluated first. If its value is true, then <statement1> (the if block) is executed and execution continues with the rest of the program. If the value is false, then

<statement2> (the else block) is executed and execution continues with the rest of the program. In other words, one of two mutually exclusive actions is performed.

e.g

if ( grade >= 60 )

System.out.println( "Passed" );

else

System.out.println( "Failed" );

The if statement normally expects only one statement in its body. To include several statements in the body of an if (or the body of an else for an if...else statement), enclose the statements in braces ({ and }). A set of statements contained within a pair of braces is called a **block**. A block can be placed anywhere in a program that a single statement can be placed.

**Exercise:** What will be the result of attempting to compile and run the following class?

public class IfTest {

public static void main(String[] args) {

if (true)

if (false)

else

}

}

System.out.println("a"); System.out.println("b");

**Conditional Operator (?:)**

Java provides the conditional operator (**?:**) that can be used in place of an if...else statement. This is Java's only **ternary operator** this means that it takes three operands. Together, the operands and the ?: symbol form a **conditional expression.** The first operand (to the left of the ?) is a **boolean** expression (i.e., a condition that evaluates to a boolean value true or false), the second operand (between the ? and :) is the value of the conditional expression if the boolean expression is true and the third operand (to the right of the :) is the value of the conditional expression if the boolean expression evaluates to false. For example, the statement

System.out.println( studentGrade >= 60 ? "Passed" : "Failed" );

Exercise: What is the result of these statements? And How…?

int abebe=70;

int jamal=40;

System.out.println( abebe >= 60 ? "Passed" : "Failed" ); System.out.println( jamal >= 60 ? "Passed" : "Failed" );

**The if-else-if Ladder(Multi way if-else Statement)**

A common programming construct that is based upon the nested if is the if-else-if ladder. As soon as a true condition is found, the statement associated with it is executed, and the rest of the ladder is bypassed. It looks like this:

if(condition)

statement;

else if(condition)

statement;

else if(condition)

statement;

e.g

….

else

statement ;

if ( studentGrade >= 90 ) System.out.println( "A" );

else if ( studentGrade >= 80 ) System.out.println( "B" );

else if ( studentGrade >= 70 ) System.out.println( "C" );

else if ( studentGrade >= 60 )

System.out.println( "D" );

else

System.out.println( "F" );

Exercice: Write a multi way if-else statement that classifies the value of an int variable n in to one of the following categories and writes out an appropriate message:

n < 0 or 0 ≤ n < 100 or n ≥ 100

**The switch Statement**

The switch statement is the only other kind of Java statement that implements multiway branches.

SYNTAX

switch ( Controlling\_Expression ){

case Case\_Label\_1 : Statement\_Sequence\_1 break;

case Case\_Label\_2 : Statement\_Sequence\_2 break ;

.

.

.

case Case\_Label\_n : Statement\_Sequence\_n break ;

default : Default\_Statement\_Sequence

break;

}

Each Case\_Label is a constant of the same type as the Controlling\_Expression. The Controlling\_Expression must be of type char, int, short, byte, or string.

A break may be omitted. If there is no break, execution just continues to the next case.

The default case is optional.

// Demonstrate the switch. class SwitchDemo {

public static void main(String args[]) {

int i;

for(i=0; i<10; i++)

switch(i) {

case 0:

case 1: case 2: case 3: case 4:

default:

}

}

}

System.out.println("i is zero");

break;

System.out.println("i is one");

break;

System.out.println("i is two");

break;

System.out.println("i is three");

break;

System.out.println("i is four");

break;

System.out.println("i is five or more");

Exercise: what is the output of the following program?

class NoBreak {

public static void main(String args[]) {

int i;

for(i=0; i<=5; i++) {

switch(i) {

case 0:

case 1:

case 2:

case 3:

case 4:

System.out.println("i is less than one");

System.out.println("i is less than two");

break;

System.out.println("i is less than three"); System.out.println("i is less than four");

break;

System.out.println("i is less than five");

default:

}

}

System.out.println("i is five or more");

}

You can have emptycases, as shown in this example:

switch(i) {

case 1:

case 2:

case 3: System.out.println("i is 1, 2 or 3");

break;

case 4: System.out.println("i is 4");

break;

}

It is possible to have a switch as part of the statement sequence of an outer switch. This is called a nested switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise. For example, the following code fragment is perfectly acceptable.

switch(ch1) {

case 'A': System.out.println("This A is part of outer switch.");

switch(ch2) {

case 'A':

System.out.println("This A is part of inner switch");

break;

case 'B': // ...

} // end of inner switch

break;

case 'B': // ...

**3.2 Repetition statements**

Loops allow a block of statements to be executed repeatedly (i.e., iterated). A boolean condition (called the loop condition) is commonly used to determine when to terminate the loop. The statements executed in the loop constitute the loop body. The loop body can be a single statement or a block.

**For loop statement**

The general form of the for loop for repeating a single statement is for(**initialization; condition; iteration**)statement;

For repeating a block, the general form is for(initialization;condition;iteration)

{

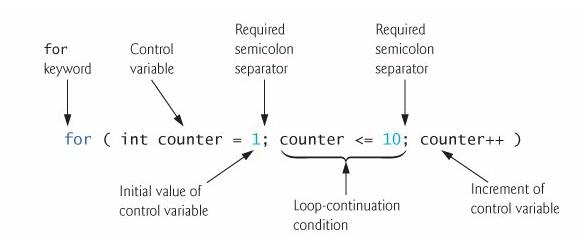
statement sequence

}

The **initialization** is usually an assignment statement that sets the initial value of the loop control variable, which acts as the counter that controls the loop.

The **condition** is a Boolean expression that determines whether or not the loop will repeat.

The **iteration** expression defines the amount by which the loop control variable will change each time the loop is repeated.



e.g

int sum = 0;

int[] array = {12, 23, 5, 7, 19};

for (int index = 0; index < array.length; index++) // (1)

sum += array[index];

You can have any number of initialization and iteration statements e.g

int i, j;

for(i=0, j=10; i < j; i++, j--)

System.out.println("i and j: " + i + " " + j);

In Java, it is possible for any or all of the initialization, condition, or iteration portions of the for loop to be blank. For example, consider the following program.

class Empty {

public static void main(String args[]) {

int i;

for(i = 0; i < 10; ) { // The iteration expression is missing

System.out.println("Pass #" + i);

i++; // increment loop control var

}

}

}

In the next example, the initialization portion is also moved out of the for class Empty2 {

public static void main(String args[]) {

int i;

i = 0; // move initialization out of loop for(; i < 10; ) {

System.out.println("Pass #" + i);

i++; // increment loop control var

}

}}

It is perfectly legal to nest one loop statement inside another loop statement. For example, the following nests one for loop inside another for loop:

int rowNum, columnNum;

for (rowNum = 1; rowNum <= 3; rowNum++)

{

for (columnNum = 1; columnNum <= 2; columnNum++) System.out.print(" row " + rowNum + " column " + columnNum);

System.out.println( );

}

This produces the following output:

row 1 column 1 row 1 column 2 row 2 column 1 row 2 column 2

row 3 column 1 row 3 column 2

A while loop, do-while loop, or for loop does not terminate as long as the controlling Boolean expression evaluates to true. A loop that runs forever is called an **infinite loop**.

Exercise :

**1.** Write an application that displays the following patterns separately, one below the other. Use **for loops**

**to generate the patterns.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) | b) |  | c) |  |
| \* |  | \*\*\*\*\*\* |  | \*\*\*\*\*\* |
| \*\* |  | \*\*\*\*\* |  | \*\*\*\*\* |
| \*\*\* |  | \*\*\*\* |  | \*\*\*\* |
| \*\*\*\* |  | \*\*\* |  | \*\*\* |
| \*\*\*\*\* |  | \*\* |  | \*\* |
| \*\*\*\*\*\* |  | \* |  | \* |

2. What will be if the following for loop is executed

for( ; ; )

{

System.out.println(“ for loop “);

}

**The while** **Loop Statement**

A while STATEMENT WITH A SINGLE-STATEMENT BODY

while ( Boolean\_Expression ) Statement

A while STATEMENT WITH A MULTISTATEMENT BODY

while (Boolean\_Expression)

{ Statement\_1

Statement\_2

.

.

Statement\_Last

}

Example

int product = 3;

while ( product <= 100 )

product = 3 \* product;

<initialization>

while (<loop condition>) {

<loop body>

<increment expression>

Equivalent with

for (<initialization>; <loop condition>; <increment expression>)

<loop body>

}

**do-while Loop statement**

The do-while loop checks its condition at the bottom of the loop. This means that a do-while loop will always execute at least once. The general form of the do-while loop is

do {

statements;

} while(condition);

Do not forget the final semicolon. e.g

int countDown = 3;

do

{

System.out.println("Hello");

countDown = countDown - 1;

} while(countDown > 0);

The important difference between the while and do-while loops involves when the controlling boolean expression is checked. With a while statement, the Boolean expression is checked before the loop body is executed. If the Boolean expression evaluates to false, then the body is not executed at all. With a do-while statement, the body of the loop is executed first and the Boolean expression is checked after the loop body is executed.

**3.3 Transfer statements**

**The break Statement in loop**

It is possible to force an immediate exit from a loop, bypassing any remaining code in the body of the loop and the loop’s conditional test, by using the break statement. When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

Example

class BreakDemo {

public static void main(String args[]) {

int num;

num = 100;

// loop while i-squared is less than num for(int i=0; i < num; i++) {

if(i\*i >= num) break; // terminate loop if i\*i >= 100

System.out.print(i + " ");

}

System.out.println("Loop complete.");

}

}

This program generates the following output:

0 1 2 3 4 5 6 7 8 9 Loop complete.

Exercise: Given the following code what will be the output?

class MyClass {

public static void main(String[] args) {

int k=0;

int l=0;

for (int i=0; i <= 3; i++) {

k++;

if (i == 2) break;

l++;

}

System.out.println(k + ", " + l);

}

}

If we use a break statement inside a set of nested loops, the break statement will break out of only the inner most loop. For example:

// Using break with nested loops. class Break3 {

public static void main(String args[]) {

for(int i=0; i<3; i++) {

System.out.println("Outer loop count: " + i); System.out.print(" Inner loop count: ");

int t = 0;

while(t < 100) {

if(t == 10) break; // terminate loop if t is 10

System.out.print(t + " ");

t++;

} System.out.println();

}

System.out.println("Loops complete.");

}

}

This program generates the following output: Outer loop count: 0

Inner loop count: 0 1 2 3 4 5 6 7 8 9

Outer loop count: 1

Inner loop count: 0 1 2 3 4 5 6 7 8 9

Outer loop count: 2

Inner loop count: 0 1 2 3 4 5 6 7 8 9

Loops complete.

**Labeled break statement**

A labeled break statement can be used to terminate any labeled statement that contains the break statement. Control is then transferred to the statement following the enclosing labeled statement. In the case of a labeled block, the rest of the block is skipped and execution continues with the statement following the block:

out:

{ // (1) Labeled block

// ...

if (j == 10) break out; // (2) Terminate block. Control to (3). System.out.println(j); // Rest of the block not executed if j == 10.

// ...

}

// (3) Continue here.

// Using break with a label. class Break4 {

public static void main(String args[]) {

int i; for(i=1; i<4; i++) { one: {

two: {

three: {

System.out.println("\ni is " + i);

if(i==1) break one; if(i==2) break two; if(i==3) break three;

// this is never reached

System.out.println("won't print");

} System.out.println("After block three.");

} System.out.println("After block two.");

}

System.out.println("After block one.");

}

System.out.println("After for.");

}

}

The output from the program is shown here:

i is 1

After block one. i is 2

After block two. After block one. i is 3

After block three. After block two. After block one. After for

**The continue Statement in loop**

The continue statement, when executed in a while, for or do...while, skips the remaining statements in the loop body and proceeds with the next iteration of the loop. In while and do - while loops, a continue statement will cause control to go directly to the conditional expression and then continue the looping process. In the case of the for, the iteration expression of the loop is evaluated, then the conditional expression is executed, and then the loop continues.

public class ContinueTest

{

public static void main( String args[] )

{

for ( int count = 1; count <= 10; count++ ) // loop 10 times

{

if ( count = = 5 ) // if count is 5,

continue; // skip remaining code in loop if ( count = = 8)

continue; // skip remaining code in loop

System.out.printf( "%d ", count );

} // end for

System.out.println( "\nUsed continue to skip printing 5" );

} // end main }

The output:

1 2 3 4 6 7 9 10

**Labeled continue** **Statement**

A labeled continue statement must occur within a labeled loop that has the same label. Executions of the labeled continue statement then transfers control to the end of that enclosing labeled loop.

// Use continue with a label. class ContToLabel {

public static void main(String args[]) {

outerloop:

for(int i=1; i < 5; i++) {

System.out.print("\nOuter loop pass " + i +", Inner loop: ");

for(int j = 1; j < 5; j++) {

if(j = = 3) continue outerloop; // continue outer loop

System.out.print(j);

}

}

}

}

The output from the program is shown here: Outer loop pass 1, Inner loop: 12

Outer loop pass 2, Inner loop: 12

Outer loop pass 3, Inner loop: 12

Outer loop pass 4, Inner loop: 12