|  |  |
| --- | --- |
| **Topic** | **Java Control Structures Part 2 (Iteration)** |
| **Learning Outcomes** | The ability of a computer to perform complex tasks is built on just a few ways of combining simple commands into control structures. In Java, there are just six such structures that are used to determine the normal flow of control in a program—and, in fact, just three of them would be enough to write programs to perform any task. The six control structures are: the block, the while loop, the do while loop, the for loop, the if statement, and the switch statement. Each of these structures is considered to be a single “statement,” but each is in fact a structured statement that can contain one or more other statements inside itself. At the end of the lesson, you will be able to:1. Know Java Programming Control Structures
2. Understand the Difference between a looping constructs and branching Statements
3. Realize the importance of Control Structures in creating programs
4. Create program using Java Control Structures
 |
| **References** | Lemay, L., Perkins, C. L. (1996), Teach Yourself JAVA in 21 days,  Indianapolis, Indiana, Sams.net.Doug Lowe, Java® All-in-One For Dummies®, 4th Edition), New Jersey,  John Wiley & Sons, Inc.Barry Burd Ph.D., Java® For Dummies®, 7th Edition, New Jersey,  John Wiley & Sons, Inc.David J. Eck, Introduction to Programming Using Java Version 5.0, December 2006 Geneva, NY, Hobart and William Smith CollegesJester Lhee I. Pandio, Computer Programming 1, Global Port Taguig City, STI College |

**The while statement**

The basic format of the while statement is this:

while (expression)

statement

The while statement begins by evaluating the expression. If the expression is true, statement is executed. Then the expression is evaluated again, and the whole process repeats. If the expression is false, statement is not executed, and the while loop ends.

Note that the statement part of the while loop can either be a single statement or a block of statements contained in a pair of braces. Loops that have just one statement are not very useful, so nearly all the while loops you code use a block of statements.

 **A counting loop**

Here is a simple program that uses a while loop to print the even numbers from 2 through 20 on the console:

public class EvenCounter{

public static void main (String[] args) {

int number = 2;

while (number <= 20) {

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

number += 2;

}

System.out.println();

}}

If you run this program, the following output is displayed in the console window:

2 4 6 8 10 12 14 16 18 20

The conditional expression in this program is while statement is number<=20. That means the loop repeats as long as the value of number is less than or equal to 20. The body of the loop consists of two statements. The first prints the value of number followed by a space to separate this number from the next one. Then the second statement adds 2 to number.



Figure 5-1 shows a flowchart for this program. This flowchart can help you visualize the basic decision-making process of a loop.

**Breaking Out of a Loop**

In many programs, you need to set up a loop that has some kind of escape clause. Java’s escape clause is the break statement. When a break statement is executed in a while loop, the loop ends immediately. Any remaining statements in the loop are ignored, and the next statement executed is the statement that follows the loop.

Suppose that you are afraid of the number 12, You could modify the counting program shown in the preceding section so that when it gets to the number 12, it panics and aborts the loop:

public class Dodecaphobia{

public static void main(String[] args) {

int number = 2;

while (number <= 20) {

if (number == 12)

break;

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

number += 2;

}

System.out.println();

}

}

When you run this program, the following line is displayed on the console:

2 4 6 8 10Whew! That was close. Almost got to 12 there.

**Infinite Looping**

One common form of loop is called an infinite loop. That is a loop that goes on forever. You can create infinite loops many ways in Java (not all of them intentional), but the easiest is to just specify true for the while expression.

Here’s an example:

public class CountForever{

public static void main(String[] args) {

int number = 2;

while (true) {

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

number += 2;

}

}

}

Obviously, infinite loops are something you want to avoid in your programs. So whenever you use a while expression that’s always true, be sure to throw in a break statement to give your loop some way to terminate. You could use an infinite loop with a break statement in the Dodecaphobia program.

**Letting the user decide when to quit**

It turns out that infinite loops are also useful when you want to let the user be in charge of when to stop the loop. Suppose that you do not know what numbers a user is afraid of, so you want to count numbers until the user says to stop. Here’s a program that does that:

import java.util.Scanner;

public class NumberPhobia {

static Scanner sc = new Scanner(System.in);

public static void main(String[] args){

int number = 2;

String input;

while (true) {

System.out.println(number + " ");

System.out.print ("Do you want to keep counting?" + " (Y or N)"); input = sc.next();

if (input.equalsIgnoreCase("N"))

break;

number+= 2;

}

System.out.println("\nWhew! That was close.\n");

}

}

Here’s some typical console output from this program, for a user who has octophobia:

2 Do you want to keep counting? (Y or N)y

4 Do you want to keep counting? (Y or N)y

6 Do you want to keep counting? (Y or N)n

Whew! That was close.

**Letting the user decide in another way**

Another way to write a loop that a user can opt out of is to test the input string in the while condition. The only trick here is that you must first initialize the input string to the value that continues the loop. Otherwise, the loop does not execute at all! Here is a variation of the NumberPhobia program named NumberPhobia2 that uses this technique:

import java.util.Scanner;

public class NumberPhobia2{

static Scanner sc = new Scanner(System.in);

public static void main(String[] args) {

int number = 2;

String input = "Y";

while (input.equalsIgnoreCase("Y")) {

System.out.println(number + " ");

System.out.print ("Do you want to keep counting?" + " (Y or N)");

input = sc.next();

number += 2;

}

System.out.println("\nWhew! That was close.");

}

}

This program works almost the same way as the preceding version, but with a subtle difference. In the first version, if the user says N after the program displays 6, the value of the number variable after the loop is 6 because the break statement bails out of the loop before adding 2 to number. But in this version, the value of number is 8.

**Using the continue Statement**

The break statement is rather harsh: It completely bails out of the loop. Sometimes that is what you need — but just as often, you do not really need to quit the loop; you just need to skip a particular iteration of the loop. The Dodecaphobia program presented earlier in this chapter stops the loop when it gets to 12. What if you just want to skip the number 12, so you go straight from 10 to 14? To do that, you can use the ***“continue”*** statement. The ***“continue”*** statement sends control right back to the top of the loop, where the expression is immediately evaluated again. If the expression is still true, the loop’s statement or block is executed again.

Here’s a version of the Dodecaphobia program that uses a continue statement to skip the number 12 rather than stop counting altogether when it reaches 12:

public class Dodecaphobia3{

public static void main(String[] args) {

int number = 0;

while (number < 20) {

number += 2;

if (number == 12)

continue;

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

}

 System.out.println();

}

}

Run this program, and you get the following output in the console window:

2 4 6 8 10 14 16 18 20

**do-while Loops**

A do-while loop (sometimes just called a do loop) is similar to a while loop, but with a critical difference: In a do-while loop, the condition that stops the loop isn’t tested until after the statements in the loop have executed. The basic form of a do-while loop is this:

do

 Statement

while (expression);

Note that the while keyword and the expression are not coded until after the body of the loop. As with a while loop, the body for a do-while loop can be a single statement or a block of statements enclosed in braces. Also, notice that the expression is followed by a semicolon. do-while is the only looping statement that ends with a semicolon and executes at least once even if the condition evaluation is already false from the start.

**Example:**

public class EvenCounter2{

public static void main(String[] args) {

int number = 2;

do {

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

number += 2;

 } while (number <= 20);

System.out.println();

}}

Figure 5-2: The flowchart for a do-while loop.

**The For Loop**

In addition to while and do-while loops, Java offers the for loop. You may have noticed that many of the loops presented so far involve counting. It turns out that counting loops are quite common in computer programs, so the people who design computer programming languages (they’re called computer programming language designers) long ago concocted a special kind of looping mechanism that is designed just for counting. The basic principle behind a for loop is that the loop itself maintains a counter variable — that is, a variable whose value increases each time the body of the loop is executed. If you want a loop that counts from 1 to 10, you’d use a counter variable that starts with a value of 1 and is increased by 1 each time through the loop. Then you would use a test to end the loop when the counter variable reaches 10. The for loop lets you set all this up in one convenient statement

**Format of the for loop**

 for (initialization-expression; test-expression; count-expression) statement;

The three expressions in the parentheses following the keyword for control how the for loop works. The following paragraphs explain what these three expressions do:

1. The initialization expression is executed before the loop begins. Usually, you use this expression to initialize the counter variable. If you have not declared the counter variable before the for statement, you can declare it here too.
2. The test expression is evaluated each time the loop is executed to determine whether the loop should keep looping. Usually, this expression tests the counter variable to make sure that it is still less than or equal to the value you want to count to. The loop keeps executing as long as this expression evaluates to true. When the test expression evaluates to false, the loop ends.
3. The count expression is evaluated each time the loop executes. Its job is usually to increment the counter variable.

Here’s a simple for loop that displays the numbers 1 to 10 on the console:

public class CountToTen{

public static void main(String[] args) {

for (int i = 1; i <= 10; i++)

System.out.println(i);

}

}

Run this program, and here’s what you see on the console: 1 2 3 4 5 6 7 8 9 10

1. The initialization expression is int i = 1. This expression declares a variable named i of type int and assigns it an initial value of 1.
2. The test expression is i <= 10. As a result, the loop continues to execute as long as i is less than or equal to 10.
3. The count expression is i++. As a result, each time the loop executes, the variable i is incremented.
4. The body of the loop is the single statement System.out.println (i). As a result, each time the loop executes, the value of the i variable is printed to the console.

**Scope of the counter variable**

If you declare the counter variable in the initialization statement, the scope of the counter variable is limited to the for statement itself. Thus, you can use the variable in the other expressions that appear within the parentheses and in the body of the loop, but you can’t use it outside the loop.

**Nested Loops**

Loops can contain loops. The technical term for this is loop-de-loop. Just kid-ding. Actually, the technical term is nested loop, which is simply a loop that is completely contained inside another loop. The loop that’s inside is called the inner loop, and the loop that’s outside is called the outer loop.

**A simple nested for loop**

To demonstrate the basics of nesting, here’s a simple little program that uses a pair of nested for loops:

public class NestedLoop{

public static void main(String[] args) {

for(int x = 1; x < 10; x++) {

for (int y = 1; y < 10; y++)

System.out.print(x + "-" + y + " ");

System.out.println();

}

}

}