Hyperion Development

Task: Control Structures - For Loop

www.hyperiondev.com

Introduction

Welcome to the Control Structures - For Loop Task
Overview:

You should currently be very comfortable with the understanding and implementation of different conditional statements - if not, be sure to go through the content of your previous tasks! You'll now be sequentially exposed to loop statements to understand how they can be utilised in reducing lengthy code, preventing coding errors, as well as paving the way towards code reusability. The next statement you'll be exposed to is the for loop, which is essentially a different variation of the while loop.

-The Hyperion Team

What is a for-loop?
The for-statement is an alternative loop structure for a while-statement. This means that a for-loop is equivalent to a while-loop and one can apply a for-loop where a while-loop could be applied and vice-versa. The main difference between a while-loop and a for-loop is syntax. A for-loop allows for counter controlled repetition to written more compactly, clearer and thus easier to read. The format of the for-loop has three separate fields, separated by semicolons.
The three fields of a for-loop enable the programmer to initialise variables, specify a loop test, and update values for control variables.
Syntax:

```
for ( initialise statement; loop test; update statement )
        body
```

Initialise statement

The initialise statement consists of assignment statements separated by commas. Typically there is one assignment statement that initialises a counter-control variable. A for statement executes the initialisation only once. The initialise statement is the entry point of the for-loop, that is, when a for-loop executes it begins its execution by carrying-out this initialise statement first. Also, the loop entry is the only time the initialise statement is executed.

Loop test

The loop test is next to be executed after the initialise statement. The loop test is a boolean expression, that is, the loop test is a Java expression such that when it is evaluated the value of the expression is a boolean. The loop test expression is evaluated prior to any iteration of the for-loop. If the condition is true then the program control is passed to the loop body; if false, control passes to the first statement after the loop body.

Update statement

Update statements assign new values to the loop control variables. The statements typically use the increment (++) or decrement (--) operator to update the control variable if it changes by 1 or a compound assignment statement if it changes by some other amount. An update statement is always and only executed next after the body has been executed. After the update statement has been executed, control passes to the loop test to mark the beginning of the next iteration.

Example 1:

This example aims to show the syntax of a for-loop and how a for-loop is a while-loop expressed differently. This is best done by contrasting a for-loop with a while-loop.
Look at the problem of computing the sum of the first 10 integers: 1 + 2 + ... + 10. The variable is the counter. For each loop, use the following declaration.

```
int i, sum = 0;
```

	while-loop	for-loop
init:	i = 1;	for(i = 1; i <= 10; i++)
test:	while(i <= 10)	sum += i;
	{	
	sum += i;	
update:	i++;	
	}	

Example 2:

A for-statement simulates a countdown to blastoff. The integer control variable is declared within the initialise statement and so has scope only within the for-loop.

```
for( int count = 10; count > 0; count --)
        System.out.print(count + " ");      //loop body
System.out.println("Blast Off!!!");         //after execution of for loop
```

Break Statement

Within a loop body, a break statement causes an immediate exit from the loop to the first statement after the loop body. The break allows for an exit at any intermediate statement in the loop.

Syntax:

```
break;
```

Using a break statement to exit a loop has limited, but important applications. Let us describe one of these situations. A program may use a loop to input data from a file. The number of iterations depends on the amount of data in the file. The task of reading from the file is part of the loop body which thus becomes the place where the program discovers that data is exhausted. When the end-of-file condition becomes true, a break statement exits the loop. In selecting a loop construct (either while-loop, do-while-loop, or for-loop) to read from a file, we recognise that the test for end-of-file occurs within the loop body. The loop statement has the form of an infinite loop: one that runs forever. The assumption is that we do not know how much data is in the file. Versions of the for-loop and the while-loop permit a programmer to create an infinite loop. In the for-loop each field of the loop is empty. There are no control variables and no loop test. The equivalent while-loop uses the constant true as the logical expression.

```
for(;;)                 while(true)
  loop block              loop block
```

Let us look at a code structure that reads data from a file. An infinite while-loop ensures that the input repeats. The loop terminates when the loop body recognises that an input request was not carried out because the process reached the end of the file. The condition end-of-file (eof) is true:

```
while( true )
{
        <read data from the file>
        if( eof )
                break;
        <process data from this input>
}
```

Instructions

First read example.java, open it using jGRASP (Right click the file and select 'Open with jGRASP").

- In this folder there is a file called example.java
- Open this folder using the JGRASP program. You can either do this by right clicking on the example.java file, going to the 'Open with' menu, and selecting JGrasp.exe. Alternatively, you can run the JGRASP program on your computer, go to the top left corner of the program, select File->Open and navigate to example.java on your hard drive and double click on it to open it.
- Once example.java is open in JGRASP please read all of its content very carefully. This will help you understand the basic structure of a Java program.
- There is a compulsory task at the end of the example.java document. The instructions of what to do to complete the task can be found here. You must complete this exercise to continue onto the next task of this course.

Compulsory Task 1

Follow these steps:

- In the while.java program you wrote in the Control Structures - While Loop task, modify the program to use a for-loop repetition construct where the while-loop was used.
- This should be the only modification made to your source code.
- Save this new version as for.java.
- Place comments that describe the modifications you have had to make to use a for-loop.
- Compile, save and run your file.

Compulsory Task 2

Follow these steps:

- Modify do_whilePassword.java from the Control Structures - Do While Loop task to use for-loops to achieve the same computations being done by the while-loop(s) of that source code.
- Let that be the only change in the new source code.
- Place comments that describe the modifications you have had to make to use a for-loop.
- Compile, save and run your file.

Things to look out for

1. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **jGRASP or Java** may not be installed correctly.
2. If you are not using Windows, please ask your tutor for alternative instructions.

Still need help?

Just write your queries in your comments.txt file and your tutor will respond.

Task Statistics

Last update to task: 20/01/2016.
Authors: Riaz Moola and Jared Ping.
Main trainer: Umar Randeree.
Task Feedback link: [Hyperion Development Feedback.]