Hyperion Development

Task: Control Structures - While Loop

www.hyperiondev.com

Introduction

Welcome to the Control Structures - While 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 your previous task content! 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. This begins with the while loop, which is the simplest loop of the group.

-The Hyperion Team

What is a while?
A *while-loop* is the most general form of loop statements. The while-statement repeats its action until the controlling condition becomes false. In other words the statement repeatedly executes "while" the condition is true (hence the name). The while-statement begins with the keyword *while* followed by a boolean expression in parentheses. The expression is tested before beginning each iteration. If the test is true then the program passes control to the loop body; if false, the control passes to the statement after the body.

Syntax:

```
while ( logical expression )
{
        body
}
```

Example:
A while-statement sums successive even integers 2 + 4 + 6 + 8 + ... until the total is greater than 250. An update statement increments i by 2 so that it becomes the next even integer. This is an event-controlled loop (as opposed to counter-controlled loops like the for-loop) because iterations continue until some non-counter-related condition (event) stops the process.

```
int i, sum = 0;
i = 2; //initial even integer for the sum
while( sum <= 250 )     //loop test; check current value of sum
{
        sum += i;       //add integer to sum
        i += 2;         //update i to next even integer
        System.out.print( "+"+i );
}
System.out.println();
```

Compile and run the example.java file to see the execution output of the above program.

Get into the loop of things

Loops are a handy tool that enables programmers to do repetitive tasks with minimal effort. Say we want a program that can count from 1 to 10, we could write the following program:

Counting from 1 to 10

```
class Count {
   public static void main(String[] args) {
      System.out.println('1 ');
      System.out.println('2 ');
      System.out.println('3 ');
      System.out.println('4 ');
      System.out.println('5 ');
      System.out.println('6 ');
      System.out.println('7 ');
      System.out.println('8 ');
      System.out.println('9 ');
      System.out.println('10 ');
   }
}
```

The task will be completed just fine, the numbers 1 to 10 will be printed in the output, but there are a few problems with this solution:

- **Flexibility**: what if we wanted to change the start number or end number? We would have to go through and change them, adding extra lines of code where they're needed.
- **Scalability**: 10 repeats are trivial, but what if we wanted 100 or even 1000 repeats? The number of lines of code needed would be overwhelming and very tedious for a large number of iterations.
- **Maintenance**: where there is a large amount of code, one is more likely to make a mistake.
- **Feature**: the number of tasks is fixed and doesn't change at each execution.

Using loops we can solve all these problems. Once you get your head around them, they will be invaluable to solving many problems in programming.

Now let's consider the following code:

Counting from 1 to 10 with a while loop

```
public class loop {
   public static void main (String[] args) {
      int i = 1;
      while (i <= 10) {
         System.out.println(i);
         i++;
      }
   }
}
```

If we run the program, the same result is produced, but looking at the code, we immediately see the advantages of loops. Instead of executing 10 different lines of code, line 5 executes ten times. 10 lines of code have been reduced to just 4. Furthermore, we may change the number 10 to any number we like. Try it yourself, replace the 10 with your own number.

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:

- Create a new file called while.java
- Write a program that always asks the user to enter a number.
- When the user enters the negative number -1, the program should stop requesting the user to enter a number,
- The program must then calculate the average of the numbers entered exclusive of the -1.
- Make use of the while loop repetition structure to implement the program.
- Compile, save and run your file.

Optional Task

Follow these steps:

- Modify your while.java file to do the following:
  - Require the user to enter their name, with only a certain name being able to trigger the loop.
  - Print out the number of tries it took the user before inputting the correct number.
  - Add a conditional statement that will cause the user to exit the program without giving the average of the numbers entered if they enter a certain input.
- 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.]