Loops Using While & Do...While

by Sophia

WHAT'S COVERED

In this lesson, you will learn about patterns that generate while loops when coding an algorithm. Specifically, this lesson covers:

1. Getting Started With While

In the previous tutorial, you briefly learned about the while loop.

As you recall, the structure of the while loop looks like this:

EXAMPLE

```
while( <expression>) {
 <statement(s)>
}
```

HINT

In the example above, the `<expression>` and `<statement(s)>` terms with outside arrows are just for information purposes; these are not keywords or actual code. These are just to explain what goes into each of the parts of a while loop.

The `<statements(s)>` represents the block of code that should be executed repeatedly. This is also called the body of the loop. The `<expression>` typically is based on one or more variables that are initialized outside of the loop and then modified within the body of the loop. The while loop continually evaluates the `<expression>` (the condition) looking for a True or False value. It keeps going (looping) as long as the evaluated condition is True. Once it is False, it exits the loop.

KEY CONCEPT

You also learned about the use of iteration variables. The body of the loop should change the value of one or more variables (in the <expression>) so that eventually the condition becomes false and the loop terminates. The iteration variable is what changes each time the loop executes and controls when the loop finishes. Without an iteration variable to change the condition, the loop would never finish.

Here is a simple program that keeps prompting the user to enter an even number. If the entry is not even, the loop ends:

```
import java.util.Scanner;

class WhileEven {
  public static void main(String[] args) {
    System.out.println("This program keeps prompting the user to enter numbers ");
    System.out.println("as long as the entries are even. When the user enters an ");
    System.out.println("odd number.\n");
    Scanner input = new Scanner(System.in);

    int number = 0;
    while(number % 2 == 0) {
      System.out.print("Enter a whole number: ");
      number = input.nextInt();
    }
    System.out.println("The loop is done.");
  }
}
```

THINK ABOUT IT

The variable number is initialized with a value of 0. Java treats 0 as an even number, so the loop starts running. The variable number is the iteration variable (or loop variable). This variable will change with each loop (iteration) to the latest entry by the user.

Here is a sample run of the program that shows the loop running until the user enters an odd number)(replace with code:

```
This program keeps prompting the user to enter numbers
as long as the entries are even. When the user enters an
odd number.

Enter a whole number: 2
Enter a whole number: 6
Enter a whole number: 18
```

```
Enter a whole number: 5
The loop is done.
```

KEY CONCEPT

More formally, here is the flow of execution for a while loop:

1. Evaluate the expression, yielding a boolean value of true or false.
2. If the condition is false, exit the while loop and continue execution at the next statement after the loop.
3. If the condition is true, execute the body of the loop and then go and check the condition again.

THINK ABOUT IT

Think back to a past tutorial. Remember that each time the body of the loop is executed is an iteration. For the above while loop, we would say, “It had four iterations,” which means that the body of the loop was executed four times.

Here is an example where a text entry is required to exit the loop:

```
import java.util.Scanner;

class AppendWhile {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    String textToAdd = "";
    String text = "";

    // Can't use == to check for equality of Strings. Need to use
    // the equals() or equalsIgnoreCase() to compare Strings.
    // Remember that ! means "not"
    while(!textToAdd.equalsIgnoreCase("quit")) {
      System.out.print("Enter a string, enter quit to exit the loop: ");
      textToAdd = input.nextLine();
      // Append input to text with space after it.
      text += textToAdd + " ";
    }
    System.out.println("\n" + text);
  }
}
```

Now, consider the while loop. It loops as long as the entry is not set to the word “quit.” Start by prompting the user for a string and then store it in textEntered. Check if the textEntered is not equal to “quit.” If it isn’t, then the stringBuilder adds the textEntered with a space. Once the user enters the word “quit,” the loop ends, and the text is output to the screen.

EXAMPLE The output screen shows the string "this is a test" being built in a while loop as the output.

```
Enter a string, enter quit to exit the loop: This
Enter a string, enter quit to exit the loop: is
Enter a string, enter quit to exit the loop: just
Enter a string, enter quit to exit the loop: a test.
Enter a string, enter quit to exit the loop: quit

This is just a test. quit
```

2. Break and Continue

When using while loops so far, the entire body of the loop was executed on each iteration. Java has two reserved keywords that allow a loop to end execution early. The **break statement** can immediately terminate a loop. The program goes to the first statement after the loop. There is also the **continue statement** which ends the current loop iteration and returns to the top of the loop (starting a new iteration). The expression is then evaluated to determine if the loop will execute again, or end there.

Let's see how the break statement works. The next version of the program adds words up to four letters in length until the user enters quit. However, if the user enters a word longer than four letters, the loop ends (using `break`) and the text is displayed.

Enter the following code in the IDE in a file named Break.java:

```
import java.util.Scanner;

class Break {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    String textToAdd = "";
    String text = "";

    // Can't use == to check for equality of Strings. Need to use
    // the equals() or equalsIgnoreCase() to compare Strings.
    // Remember that ! means "not"
    // If the user enters a word longer than 4 letters, the loop ends.
    while(!textToAdd.equalsIgnoreCase("quit")) {
      System.out.println("Enter a string, enter quit or a word of more than ");
      System.out.print("letters to exit the loop: ");
      textToAdd = input.nextLine();
      if(textToAdd.length() > 4) {
        break;
```

```
      }
      // Append input to text with space after it.
      text += textToAdd + " ";
    }
    System.out.println("\n" + text);
  }
}
```

The results should look like this:

```
Enter a string, enter quit or a word of more than
letters to exit the loop: This
Enter a string, enter quit or a word of more than
letters to exit the loop: is
Enter a string, enter quit or a word of more than
letters to exit the loop: just
Enter a string, enter quit or a word of more than
letters to exit the loop: a
Enter a string, enter quit or a word of more than
letters to exit the loop: test
Enter a string, enter quit or a word of more than
letters to exit the loop: terminate

This is just a test
```

Let's try the same program, but this time, we'll add the `continue` keyword to skip a word of more than four letters but not quit until the user enters quit.

TRY IT

Directions: Enter the code in the IDE in a file named BreakContinue.java:

```
import java.util.Scanner;

class BreakContinue {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    String textToAdd = "";
    String text = "";

    // Can't use == to check for equality of Strings. Need to use
    // the equals() or equalsIgnoreCase() to compare Strings.
    // Remember that ! means "not"
    // If the user enters a word longer than 4 letters, the loop ends.
```

```
    while(!textToAdd.equalsIgnoreCase("quit")) {
      System.out.println("Enter a string or enter quit. A word of more than ");
      System.out.print("letters will be ignored: ");
      textToAdd = input.nextLine();
      if(textToAdd.length() > 4) {
        continue;
      }
      if(textToAdd.equalsIgnoreCase("quit")) {
        break;
      }
      // Append input to text with space after it.
      text += textToAdd + " ";
    }
    System.out.println("\n" + text);
  }
}
```

The output screen shows the appended text using continue to skip a long word:

```
Enter a string or enter quit. A word of more than
letters will be ignored: This
Enter a string or enter quit. A word of more than
letters will be ignored: is
Enter a string or enter quit. A word of more than
letters will be ignored: just
Enter a string or enter quit. A word of more than
letters will be ignored: a
Enter a string or enter quit. A word of more than
letters will be ignored: test
Enter a string or enter quit. A word of more than
letters will be ignored: terminate
Enter a string or enter quit. A word of more than
letters will be ignored: quit

This is just a test
```

REFLECT

Looking at the code above, think about how the keywords `break` and `continue` are used to modify how the steps in the loop are carried out. The statements in the loop are always the same, but `break` and `continue` can modify the flow through the loop when needed.

TERMS TO KNOW

break
This is a reserved keyword that creates a break statement for loops. The break statement can immediately terminate a loop entirely and disregard the execution of the loop. When this occurs, the program goes to the first statement/line of code after the loop.

continue
This is a reserved keyword that creates a continue statement for loops. The continue statement will end the current loop iteration, meaning that the execution jumps back to the top of the loop. The expression is then evaluated to determine if the loop will execute again or end there.

---

3. do...while Loops

In addition to the basic while loop that has been covered, there is an alternative loop for indefinite iteration. It is referred to as the **do...while loop**. A plain while loop has the condition for the loop at the top of the loop, and the condition is evaluated before each iteration, including the first. Depending on how the variable has been declared and initialized, there is a possibility that the loop may not run at all. The do...while loop puts the condition at the button of the loop, so the loop is guaranteed to run at least once before exiting the loop.

The basic pattern for the do...while loop is:

EXAMPLE

```
do {
 <statement(s)>
} while( <expression>)
```

Here is a revised version of the first program in this lesson. Instead of initializing the loop variable before the while loop, this version uses do...while to assign the user's input to the variable. Since the condition is evaluated after the input, the loop is guaranteed to run at least once.

TRY IT

Directions: Try this version of the code in the IDE, saved in a file named DoWhile.java:

```
import java.util.Scanner;

class DoWhile {
  public static void main(String[] args) {
    System.out.println("This program keeps prompting the user to enter numbers ");
    System.out.println("as long as the entries are even. When the user enters an ");
    System.out.println("odd number.\n");
    Scanner input = new Scanner(System.in);
```

```
    // number not initialized since it will get a value in the body of the loop
    int number;
    do{
      System.out.print("Enter a whole number: ");
      number = input.nextInt();
    }
    while(number % 2 == 0);

    System.out.println("The loop is done.");
  }
}
```

A sample run of the program shows the DoWhile.java as the output.

```
This program keeps prompting the user to enter numbers
as long as the entries are even. When the user enters an
odd number.

Enter a whole number: 2
Enter a whole number: 4
Enter a whole number: 5
The loop is done.
```

REFLECT

When thinking about using a do...while loop, it is important to note how its structure differs from other loop types. With other types of loops, the definition of the condition that controls the loop is in the first line, but where is the key condition for a do...while loop?

As the first code in this lesson shows, it's possible to do the same thing using a plain while loop with a properly initialized variable. A plain while loop also has the advantage of placing the condition at the top of the loop, where it's easily visible, rather than at the bottom of the loop.

TERM TO KNOW

do...while Loop

A do...while loop is a specialized type of while loop where the condition is evaluated at the bottom of the loop rather than the top of the loop, so the loop will always run at least one time.

SUMMARY

In this lesson, you learned about the **while loop** in more detail. You also learned about the **break statement**, which allows us to exit out of a loop, and the **continue statement**, which allows us to jump past the end of the loop of the current iteration and continues back at the loop's conditional check.

Finally, you learned about the **do...while version of the loop** as an alternative that guarantees the body of the loop will run at least once.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

break

This is a reserved keyword that creates a break statement for loops. The break statement can immediately terminate a loop entirely and disregard the execution of the loop. When this occurs, the program goes to the first statement/line of code after the loop.

continue

This is a reserved keyword that creates a continue statement for loops. The continue statement will end the current loop iteration, meaning that the execution jumps back to the top of the loop. The expression is then evaluated to determine if the loop will execute again or end there.

do...while Loop

A do...while loop is a specialized type of while loop where the condition is evaluated at the bottom of the loop rather than the top of the loop, so the loop will always run at least one time.