Java Chapter 4: Loops

Contents

Time required: 60 minutes

DRY

Don't **R**epeat **Y**ourself

Learning Outcomes

Students will be able to:

- Write different types of loops, such as for and while
- Select the correct type of loop based on a given problem
- Use loops to solve common problems, such as, finding sum, finding max, etc.
- Write nested loops

Read: Think Java 2

- Chapter 6 Loops and Strings

Do: Java Tutorials

- Java While Loop
- Java For Loop
- Java For-Each Loop
- Java Break and Continue

Increment and Decrement Operators

For loops commonly use increment and decrement operators to help count through a loop.

increment operator: The increment operator (++) increases the value of a numeric variable by one.

decrement operator: The decrement operator (--) decreases the value of a numeric variable by one.

Pre increment

```
x = 1
// ++ happens first, then the x value is assigned to y
y = ++x
y = 2
x = 2
```

Post increment is the most used operator in a for loop. It is used when we want the values to count up.

```
X = 1
// x value is assigned to y, then increment x++
y = x++
y = 1
x = 2
```

Pre decrement

```
x = 1
// -- happens first, then x value is assigned to y
y = --x
y = 0
x = 0
```

Post decrement is the most used operator in a for loop. It is used when we want the values to count down.

```
x = 1
// x value is assigned to y, then decrement --
y = x--
y = 1
x = 0
```

For Loop

A for loop is typically used when you know exactly how many times you wish to loop.

In Java, a "**for**" loop is a control structure used for iterative tasks. It consists of three main parts:

- Initialization: You define and initialize a loop control variable.
- Condition: You specify a condition that must be true for the loop to continue.
- Iteration: You define how the loop control variable is updated after each iteration.

The syntax of a for loop.

```
for(initialization of counter; Boolean_expression; update counter) {
    // Statements
}
```

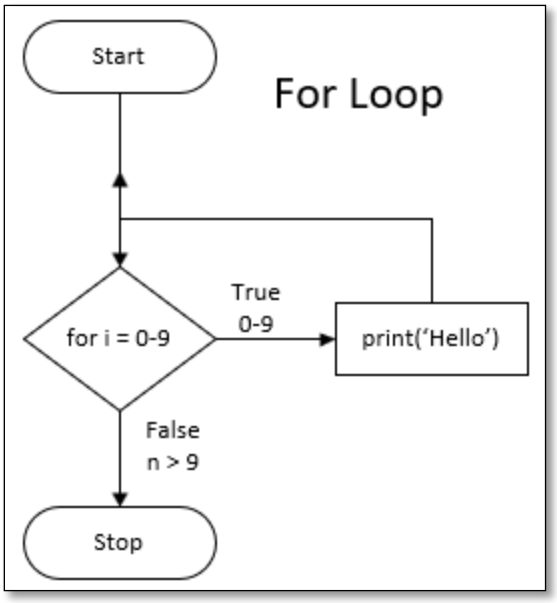
For loop example.

```
public class ForLoop {
    public static void main(String[] args) {
        for (int i = 0; i < 10; ++i) {
            System.out.println(i);
        } // end for

    }// end main
}
```

In this example, the loop initializes i to 0, continues as long as i is less than 10, and increments i by 1 in each iteration. The code inside the loop is executed repeatedly until the condition becomes false.

For Loop Diagram



Tutorial 4.1: For Loop Squares and Cubes

Create a Java program named: **SquaresAndCubes.java**

```java
/*
 * Name: SquaresAndCubes.java
 * Written by:
 * Written on:
 * Purpose: Use a for loop to print the squares
 * and cubes of the numbers 1 to 10
 */

public class SquaresAndCubes {
    public static void main(String[] args) {
        // Constant for dashes printed
        String DASHES = "---------------------";
        int square = 0;
        int cube = 0;

        // # Print the heading
        System.out.println(DASHES);
        System.out.println("Number\tSquare\tCube");
        System.out.println(DASHES);

        // A for loop from 1 - 10
        for (int i = 1; i < 11; i++) {
            // Calculate the square of the current number
            square = i * i;
            cube = i * i * i;
            System.out.println(i + "\t" + square + "\t" + cube);
        }
        System.out.println(DASHES);
    }
}
```

Example run:

```
---------------------
Number  Square  Cube
---------------------
1       1       1
2       4       8
3       9       27
4       16      64
5       25      125
6       36      216
7       49      343
8       64      512
9       81      729
10      100     1000
---------------------
```

Tutorial 4.2: For Loop Blast Off!

Here is an example program that counts down from 5 and then prints a message. We imported the **sleep** method from the **TimeUnit** library, to count down 1 second at a time.

```java
// Name: BlastoffForLoop.java
// Written by:
// Written on:
// Purpose: Count backwards with sleep to emulate
// a blastoff countdown

// Import TimeUnit library for sleep method
import java.util.concurrent.TimeUnit;

public class BlastoffForLoop {
    public static void main(String[] args) {
        // A for loop from 5 - 1
        for (int i = 5; i > 0; i--) {
            System.out.println(i);

            // Code block to sleep for 1 second
            try {
                TimeUnit.SECONDS.sleep(1);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        System.out.println("Blastoff!");
    }
}
```

Example program run:

```
5
4
3
2
1
Blastoff!
```

While Loop

in Java, a "**while**" loop is a control structure used for repetitive tasks. It continues executing a block of code as long as a specified condition is true. Here's how it works:

- Condition: You specify a condition that must be true for the loop to continue.
- Execution: The code block inside the loop is executed repeatedly as long as the condition remains true.
- Update: You must ensure that the condition eventually becomes false, or else the loop will run indefinitely.

Syntax of a while loop.

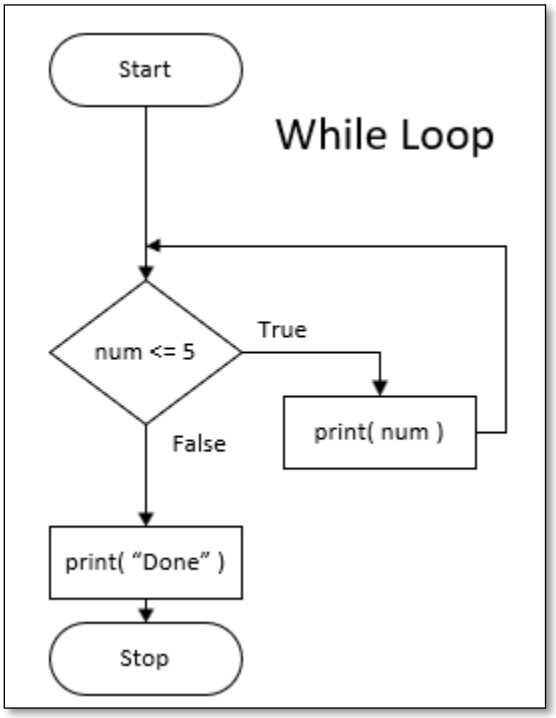
```
initialize counter
while(Boolean_expression) {
    // Statements
    update counter
    // Statements
}
```

A while loop is a pretest loop.

```
public class WhileLoop {
    public static void main(String[] args) {
        // Counter variable
        int count = 0;
        // While the condition is true
        // continue the loop
        while (count < 10) {
            System.out.println(count);
            // Increment counter variable
            count++;
        } // end while
    }
}
```

In this example, the loop starts with **count** as 0. As long as **count** is less than 10, the code inside the loop is executed. **`count`** is incremented by 1 in each iteration. The loop continues until **count** becomes equal to or greater than 10, at which point it exits.

While Loop Diagram



Break

Break causes the code to exit from the loop early. This is useful in a menu system.

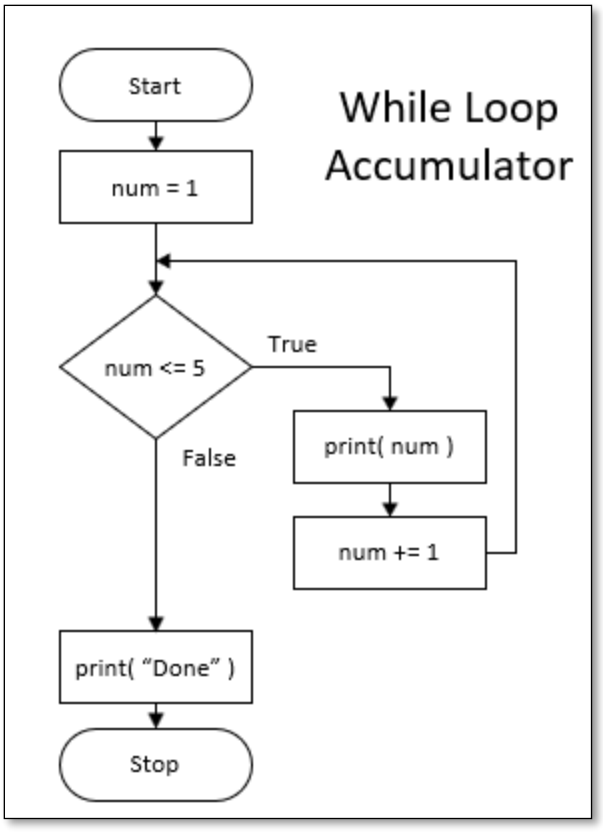
```
public class Break {
    public static void main(String[] args) {
        int count = 0;
        while(count < 10) {
            if(count == 5) {
                count++;
                // break exits the loop
                break;
            }
            System.out.print(count + "\t");
            count++;
        }//end while
    }//end main
}
```

Continue

Continue causes the code to go directly to the beginning of the loop.

```
public class Continue {
    public static void main(String[] args) {
        int count = 0;
        while (count < 10) {
            if (count % 2 != 0) {
                count++;
                // continue causes the current iteration to
                // go immediately to the beginning of the loop
                continue;
            }
            System.out.println(count);
            count++;
        } // end while
    }// end main
}
```

While Loop Accumulator Diagram



Tutorial 4.3: Running Total While Loop

The following program demonstrates how to keep a running total of numbers entered by the user.

Create a Java program named: **RunningTotal.java**

```java
// Name: RunningTotal.java
// Written by:
// Written on:
// Purpose: Sum a series of numbers entered by the user with a while loop

// Import Scanner library for input
import java.util.Scanner;

public class RunningTotal {
    public static void main(String[] args) {
        // Declare Scanner object and initialize with
        // predefined standard input object, System.in
        Scanner keyboard = new Scanner(System.in);

        // Declare variables for input and running total
        double runningTotal = 0;
        double number;

        // Print the heading and prompt
        System.out.println("+-------------------------------------+");
        System.out.println("|      Sum the entered numbers        |");
        System.out.println("+-------------------------------------+");

        while (true) {
            System.out.print("Enter a number (0 to quit): ");
            // Get double from the keyboard
            // Assign double to variable
            number = keyboard.nextDouble();
            // If the user types in the sentinel value 0
            // Break the loop
            if (number == 0) {
                break;
            }
            // Accumulate running total
            runningTotal += number;
        }

        // # Display the running total
        System.out.println("The total is: " + runningTotal);

        keyboard.close();
    }
}
```

Example run:

```
+----------------------------------+
|      Sum the entered numbers     |
+----------------------------------+
Enter a number (0 to quit): 2
Enter a number (0 to quit): 3
Enter a number (0 to quit): 100
Enter a number (0 to quit): 0
The total is: 105.0
```

Do While Loop

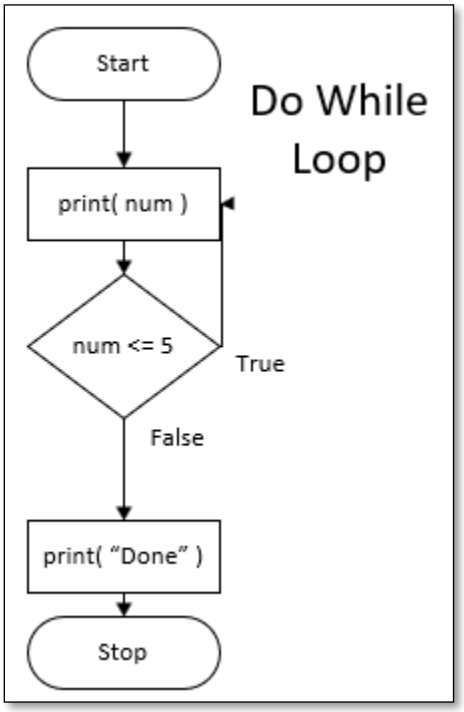
```
initialize counter
do{
    // Statements
    update counter
    // Statements
}while(Boolean_expression)
```

The do while loop is a post test loop. It will run at least once before it tests the condition.

```
/*
* Name: DoWhileLoop.java
* Written by:
* Written on:
* Purpose: Demo of do while loop
*/

public class DoWhileLoop {
    public static void main(String[] args) {
        int count = 15;
        // Posttest loop, runs at least once
        do {
            System.out.println(count);
            // Increment counter
            count++;
            System.out.println(count);
            // Continue loop while condition is true
        } while (count < 10);
    }// end main
}
```

Do While Loop Diagram



Input Validation and Exception Handling

Input validation ensures that the input is in the right range. An age should be positive. A bank withdrawal would be negative.

Exception handling ensures that the correct data type is input by the user. If the user enters a string when an integer is what the program wants, this will generate a program exception.

For example, if you want users to enter their ages, your code shouldn't accept nonsensical answers such as negative numbers (which are outside the range of acceptable integers) or words (which are the wrong data type). Input validation can also prevent bugs or security vulnerabilities.

If you implement a **WithdrawFromAccount()** method that takes an argument for the amount to subtract from an account, you need to ensure the amount is a positive number. If the **WithdrawFromAccount()** function subtracts a negative number from the account, the "withdrawal" will end up adding money!

Typically, we perform input validation by repeatedly asking the user for input until they enter valid input, as in the following example:

Tutorial 4.4: Input Validation

```java
/**
 * Name: InputValidation.java
 * Written by:
 * Written on:
 * Purpose: Validate postive integer input
 */

// Import Scanner library for user input
import java.util.Scanner;

public class InputValidation {
    public static void main(String[] args) {
        // Declare Scanner object and initialize with
        // predefined standard input object, System.in
        Scanner keyboard = new Scanner(System.in);

        int age = 0;
        String input;
```

```
20          while (true) {
21              // Try to get valid integer input
22              try {
23                  System.out.print("Enter your age: ");
24                  // Get String from the keyboard
25                  input = keyboard.nextLine();
26                  // Parse input into integer, assign to variable
27                  age = Integer.parseInt(input);
28              } catch (Exception e) {
29                  // Handle exception if input is not an integer
30                  System.out.println(e);
31                  System.out.println("Please use a whole number.");
32                  // Start the loop over
33                  continue;
34              }
35              // Is the integer a positive number
36              if (age < 1) {
37                  System.out.println("Please enter a positive number.");
38                  // Start the loop over
39                  continue;
40              } else {
41                  // Break out the loop with valid input
42                  break;
43              }
44          }
45          // Input is valid
46          System.out.println("Your age is: " + age);
47          keyboard.close();
48      }
49  }
```

Example run:

```
Enter your age: bi
Please use a whole number.
Enter your age: -25
Please enter a positive number.
Enter your age: 66
Your age is : 66
```

When you run this code, you'll be prompted for your age until you enter a valid one. This ensures that when the execution leaves the while loop, the age variable will contain a valid value that won't crash the program.

Tutorial 4.5: Nested For Loops

You can place a loop inside a for loop.

Create a Java program named **NestedLoopsFor.java**

```java
/*
 * Name: NestedLoopsFor.java
 * Written by:
 * Written on:
 * Purpose: Use a nested loop
 */

public class NestedLoopsFor {

   public static void main(String[] args) {
      System.out.println("======= Nested Loop Demo ========");
      // Exterior loop
      for (int i = 0; i < 3; i++) {
         System.out.println("-------- Exterior Loop " + i + " --------");
         System.out.println("-------- Interior Loop ----------");
         // Interior loop
         for (int j = 0; j < 5; j++) {
            System.out.print(j + "\t");
         } // End interior loop
         System.out.println("\n");
      } // End exterior loop
   }
}
```

Nested for loop

Example run:

```
======= Nested Loop Demo ========
-------- Exterior Loop 0 --------
-------- Interior Loop ----------
0       1       2       3       4

-------- Exterior Loop 1 --------
-------- Interior Loop ----------
0       1       2       3       4

-------- Exterior Loop 2 --------
-------- Interior Loop ----------
0       1       2       3       4
```

Tutorial 6: Nested While Loops

You can place a loop inside a while loop. An example of use for this would be a game loop nested inside a menu loop.

Create a Java program named **NestedLoopsWhile.java**

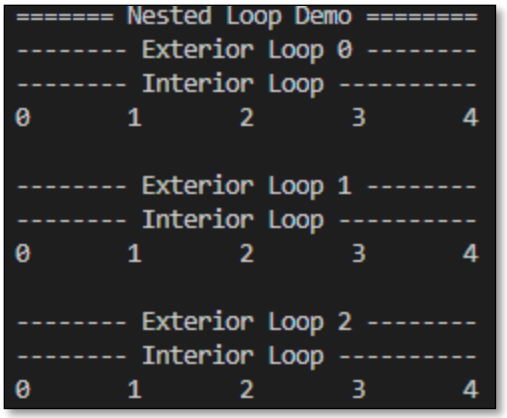
```java
/*
 * Name: NestedLoopsWhile.java
 * Written by:
 * Written on:
 * Purpose: Use a nested loop
 */

public class NestedLoopsWhile {

   public static void main(String[] args) {
      System.out.println("======= Nested Loop Demo ========");
      // Initialize exterior loop counter
      int i = 0;
      // Exterior loop
      while (i < 3) {
         System.out.println("-------- Exterior Loop " + i + " --------");
         System.out.println("-------- Interior Loop ----------");
         i++;
         // Initialize interior loop counter
         int j = 0;
         // Interior loop
         while (j < 5) {
            System.out.print(j + "\t");
            j++;
         } // End interior loop
         System.out.println("\n");
      } // End exterior loop
   }
}
```

Nested while loop

Example run:



Tutorial 7: Menu Loops with char

Menu loops can be done with integers or chars in Java. **char** is a character data type.

Java char Data Type

Python makes it very easy to compare strings using the standard relational operators. Strings in Java are a bit more complicated. As we are only using one letter, we are going to use the **char** data type for our menu.

You can see in the example below how to create and compare char variables.

```
char ch1 = 'A';
char ch2 = 'B';
char ch3 = 'A';
System.out.println(ch1 == ch2);
System.out.println(ch1 < ch2);
System.out.println(ch3 == ch1);
```

Getting char as input is a bit different as shown in the following example.

```java
// Java program to read character using Scanner class
import java.util.Scanner;

public class MenuChoice {
    public static void main(String[] args) {
        // Declare character variable
        char menuChoice;
        // Declare Scanner object for input
        Scanner keyboard = new Scanner(System.in);

        // Prompt user
        System.out.print("Please enter a menu choice (Y/N): ");

        // Character input
        menuChoice = keyboard.next().charAt(0);

        // Convert char to lowercase for easier comparison
        // You can also use toUpperCase()
        menuChoice = Character.toLowerCase(menuChoice);

        // Print the Menu Choice value
        System.out.println("Menu Choice: " + menuChoice);

        // Close Scanner OS resource
        keyboard.close();
    }
}
```

Replit MenuChoice

Debugging

As you start writing bigger programs, you might find yourself spending more time debugging. More code means more chances to make an error and more places for bugs to hide.

One way to cut your debugging time is "debugging by bisection." For example, if there are 100 lines in your program and you check them one at a time, it would take 100 steps.

Instead, try to break the problem in half. Look at the middle of the program, or near it, for an intermediate value you can check. Add a print statement (or something else that has a verifiable effect) and run the program.

If the mid-point check is incorrect, the problem must be in the first half of the program. If it is correct, the problem is in the second half.

Every time you perform a check like this, you halve the number of lines you must search. After six steps (which is much less than 100), you would be down to one or two lines of code, at least in theory.

In practice it is not always clear what the "middle of the program" is and not always possible to check it. It doesn't make sense to count lines and find the exact midpoint. Instead, think about places in the program where there might be errors and places where it is easy to put a check. Then choose a spot where you think the chances are about the same that the bug is before or after the check.

Assignment 1: Sum of Natural Numbers

Requirement: Use a for loop to calculate the sum of the first n natural numbers.

1. Write a Java program that asks the user to input a positive integer n.
2. Use a for loop to calculate the sum of all natural numbers from 1 to n.
3. Display the result.

Example run:

```
Enter a positive integer: 5
The sum of the first 5 natural numbers is: 15
```

Assignment Submission

1. Use pseudocode or TODO.
2. Comment your code to show understanding.
3. Attach the program files.
4. Attach screenshots showing the successful operation of the program.
5. Submit in Blackboard.