Chapter 4: Loops and Files

CS 2070

February 5, 2018

Increment and Decrement Operators

Increment and Decrement Operators

There are numerous times where a variable must simply be incremented or decremented.

```
number = number + 1;
number = number - 1;
```

Also.

```
number += 1;
number -= 1;
```

Increment and Decrement Operators

- Java provide shortened ways to increment and decrement a variable's value.
- Using the $++$ or - - unary operators, this task can be completed quickly.

Code.

```
number++;   or   ++number;
number--;   or   --number;
```

Differences Between Prefix and Postfix

- When an increment or decrement are the only operations in a statement, there is no difference between prefix and postfix notation.
  - It is recommended that you use increment and decrement operators by themselves, because otherwise, this gets confusing.
- When used in an expression:
  - prefix notation (++number or - -number) indicates that the variable will be incremented or decremented **prior** to the rest of the equation being evaluated.
  - postfix notation (number++ or number- -) indicates that the variable will be incremented or decremented **after** the rest of the equation has been evaluated.

Let's go over some examples.

In each of these examples, we'll go over some code. You will evaluate the code and report what is displayed. You also need to report the final value of `number` (which may be different than what is displayed). `number` will always begin as 2. These will get tricky.

```
int number = 2;
number++;
System.out.println(number);
```

Let's go over some examples.

In each of these examples, we'll go over some code. You will evaluate the code and report what is displayed. You also need to report the final value of `number` (which may be different than what is displayed). `number` will always begin as 2. These will get tricky.

```
int number = 2;
number++;
System.out.println(number);
```

Both `number` and the output is 3. This is an example of the postfix increment.

Example

```
int number = 2;
++number;
System.out.println(number);
```

Example

```
int number = 2;
++number;
System.out.println(number);
```

Both `number` and the output is 3. This is an example of the prefix increment. Any use of the prefix increment or decrement beyond these two examples is **strongly discouraged**, but we are going to do more examples anyway, because programmers still use this in odd manners.

Example

```
int number = 2;
int value = 3 + number++;
System.out.println(value);
System.out.println(number);
```

Example

```
int number = 2;
int value = 3 + number++;
System.out.println(value);
System.out.println(number);
```

`value` is 5 and `number` is 3. When `number++` is evaluated, it's evaluated as 2, then it is incremented.

Example

```
int number = 2;
int value = 3 + ++number;
System.out.println(value);
System.out.println(number);
```

Example

```
int number = 2;
int value = 3 + ++number;
System.out.println(value);
System.out.println(number);
```

`value` is 6 and `number` is 3. When `++number` is evaluated, it's incremented first, then evaluated as 3.

Example

```
int number = 2;
int value = number++ + ++number;
System.out.println(value);
System.out.println(number);
```

Example

```
int number = 2;
int value = number++ + ++number;
System.out.println(value);
System.out.println(number);
```

Addition works from left to right, so `number++` is evaluated first. This instance is 2, after which it is incremented to 3. The second instance is `++number`. `number` is 3, which is first incremented to 4. The addition is now $2 + 4$, which is 6, and that is stored in `value`. `number` is 4.

Reminder

- Whether you use `number++` or `++number`, make sure that it is the only operation that you perform within that statement. This way the statement will not be confused.
- Clever code is also code that will be confused.
- Don't be clever. Be clear.

The while Loop

The while Loop

- Java provides three different looping structures.
  - Or more than three depending on how you define a looping structure.
- The while loop has the form:

Code.

```
while(condition) {
    statements;
}
```

- While the condition is true, the statements will execute repeatedly.
- The while loop is a pretest loop, which means that it will test the value of the condition prior to executing the loop.

Pretest Loop

Pretest loops work using the following steps.

1. Test the condition. If the condition is false, leave the loop.
2. Perform the statements associated with the loop.
3. Return to Step 1.

It is possible (and often) that a loop's statements will not be performed because the loop's conditions were initially false. In a pretest loop, the statements will be executed 0 or more times. Care must be taken to set the condition to false somewhere in the loop so the loop will end.

Infinite Loops

- In order for a while loop to end, the condition must become false. The following loop will not end:

Code.

```
int x = 20;
while(x > 0) {
    System.out.println("x is greater than 0");
}
```

- The variable x never gets decremented so it will always be greater than 0.
- Adding the x- - above fixes the problem.
- Infinite loops are bad because we don't have that much time.

Infinite Loops

This version of the loop decrements x during each iteration:

```
int x = 20;
while(x > 0) {
    System.out.println("x is greater than 0");
    x--;
}
```

Block Statements in Loops

Curly braces are required to enclose block statement while loops. (like block if statements)

```
while (condition) {
    statement;
    statement;
    statement;
}
```

Of course, I think you should use curly braces for all if statements and loops. They don't hurt. They make your code prettier.

The while Loop for Input Validation

Input validation is the process of ensuring that user input is valid.

```
System.out.print("Enter a number in the " +
                 "range of 1 through 100: ");
number = keyboard.nextInt();
while (number < 1 || number > 100) {
    System.out.println("That number is invalid.");
    System.out.print("Enter a number in the " +
                     "range of 1 through 100: ");
    number = keyboard.nextInt();
}
```

The do-while Loop

The do-while Loop

- The do-while loop is a post-test loop, which means it will execute the loop prior to testing the condition.
- The do-while loop (sometimes called called a do loop) takes the form:

Code.

```
do {
    statements;
} while (condition);
```

Post-test Loops

Post-test loops work using the following steps.

1. Perform the statements associated with the loop.
2. Test the condition. If the condition is false, leave the loop.
3. Return to Step 1.

Notice the difference between this set of steps to that of a pretest loop. Steps 1 and 2 are swapped. In a posttest loop, the statements will be executed 1 or more times. Infinite loops can still happen in a post-test loop. Care must be taken to set the condition to false somewhere in the loop so the loop will end.

The for Loop

- The for loop is a pre-test loop.
- The for loop allows the programmer to initialize a control variable, test a condition, and modify the control variable all in one line of code.
- The for loop takes the form:

Code.

```
for (initialization; test; update) {
    statement(s);
}
```

The Sections of The for Loop

- The initialization section of the for loop allows the loop to initialize its own control variable.
- The test section of the for statement acts in the same manner as the condition section of a while loop.
- The update section of the for loop is the last thing to execute at the end of each loop.

The for Loop Initialization

- The initialization section of a for loop is optional; however, it is usually provided.
- Typically, for loops initialize a counter variable that will be tested by the test section of the loop and updated by the update section.
- The initialization section can initialize multiple variables.
- Variables declared in this section have scope only for the for loop.

The Update Expression

- The update expression is usually used to increment or decrement the counter variable(s) declared in the initialization section of the for loop.
- The update section of the loop executes last in the loop.
- The update section may update multiple variables.
- Each variable updated is executed as if it were on a line by itself.

Modifying The Control Variable

- You should avoid updating the control variable of a for loop within the body of the loop.
- The update section should be used to update the control variable.
- Updating the control variable in the for loop body leads to hard to maintain code and difficult debugging.

Pretest For loops.

Under Java's hood, there is absolutely no difference between a for loop and a while loop. Both are pretest loops. **However,** there are still syntactic differences and usage differences. The usage differences will be described later.

1. Initialize the control variable.
2. Test the condition. Often, we are testing the control variable that we initialized. If the condition is false, leave the loop.
3. Perform the statements associated with the loop.
4. Perform whatever code is in the update step.
5. Return to Step 2.

Notice that Step 1 is performed exactly once.

Differences between for and while

- for loops are often called “counting loops”, since they are often used for counting.
- Each of the essential parts of a for loop are encouraged by Java.
  - However, they can be ignored.
- while loops are designed for processes in which we don’t know how many loops are needed (for example, input validation).
- for loops are designed for processes in which we know exactly how many loops are needed.

Multiple Initializations and Updates

The for loop may initialize and update multiple variables.

```
for(int i = 5, int j = 0;
    i < 10 || j < 20;
    i++, j+=2) {
   statements;
}
```

I consider this to be a bad practice. Your for loops should use one and exactly one control variable.

All parts are optional.

Note that the only parts of a for loop that are mandatory are the semicolons.

```
for(;;) {
    statements;
} // infinite loop
```

If left out, the test section defaults to true. Again, you should be using exactly one control variable.

Running Totals

- Loops allow the program to keep running totals while evaluating data.
- Imagine needing to keep a running total of user input. (Such as a cash register system.)

Running Totals

```
int total = 0;
int numberOfEntries = keyboard.nextInt();
for (int i = 0; i < numberOfEntries; i++) {
    int value = keyboard.nextInt();
    total += value;
}
System.out.println("Total: "+total);
```

Sentinel Values

- Sometimes the end point of input data is not known.
- A sentinel value can be used to notify the program to stop acquiring input.
- If it is a user input, the user could be prompted to input data that is not normally in the input data range (i.e. -1 where normal input would be positive.)
- Programs that get file input typically use the end-of-file marker to stop acquiring input data.

Sentinel Values

```
final int SENTINEL = -1;
double testScoreTotal = 0;
int testScores = 0;
int score = 0;
do {
    System.out.print(
        "Enter test score: (or -1): ");
    score = keyboard.nextDouble();
    if (score >= 0) {
        testScoreTotal += score;
        testScores++;
    }
} while (score != SENTINEL);
```

Nested Loops

Nested Loops

Like if statements, loops can be nested. If a loop is nested, the inner loop will execute all of its iterations for each time the outer loop executes once.

```
for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 10; j++) {
        loop statements;
    }
}
```

The loop statements in this example will execute 100 times. Many algorithms use nested for loops, which means they are critical to learn.

The break Statement

- The break statement can be used to abnormally terminate a loop.
- The use of the break statement in loops bypasses the normal mechanisms and makes the code hard to read and maintain.
- It is considered bad form to use the break statement in this manner.

The continue Statement

- The continue statement will cause the currently executing iteration of a loop to terminate and the next iteration will begin.
- The continue statement will cause the evaluation of the condition in while and for loops.
- Like the break statement, the continue statement should be avoided because it makes the code hard to read and debug.

The break and continue statement.

- Use of these commands is considered a bad practice because they make code hard to read and hard to debug.
- They were born out of C, which continued on to C++, then Java, and even languages like Python (one of my favorites).
- Scala, a language built upon and influenced by Java, does not have a break or continue statement.
- I wish more languages followed this path by not letting the programmer make poor choices.

Deciding Which Loops to Use

- The while loop (pretest)
  - Use it when you need to loop an unknown number of times.
- The do-while loop (post-test)
  - Use it when you need to loop an unknown number of times (but at least once).
- The for loop (pretest)
  - Use it when you know exactly how many times you need to loop.
  - Use it when you have a counting loop that updates on a particular interval.

File Input and Output

- Reentering data all the time could get tedious for the user.
- The data can be saved to a file.
  - Files can be input files or output files.
- Files:
  - Files have to be opened.
  - Data is then written to the file.
  - The file must be closed prior to program termination.
- In general, there are two types of files:
  - binary
  - text

Writing Text To a File

The PrintWriter Class

To open a file for text output you create an instance of the PrintWriter class.

```
PrintWriter outputFile
    = new PrintWriter("StudentData.txt");
```

The PrintWriter Class

- The PrintWriter class allows you to write data to a file using the print and println methods, as you have been using to display data on the screen.
- Just as with the System.out object, the println method of the PrintWriter class will place a newline character after the written data.
- The print method writes data without writing the newline character.

Code.

```
PrintWriter outputFile = new PrintWriter("Names.txt");
outputFile.println("Chris");
outputFile.println("Kathryn");
outputFile.println("Jean");
outputFile.close();
```

Exceptions

- When something unexpected happens in a Java program, an exception is thrown.
- The method that is executing when the exception is thrown must either handle the exception or pass it up the line.
- Handling the exception will be discussed later.
- To pass it up the line, the method needs a throws clause in the method header.

Exceptions

- To insert a throws clause in a method header, simply add the word throws and the name of the expected exception.
- PrintWriter objects can throw an IOException, so we write the throws clause like this:

Code.

```
public static void main(String[] args)
    throws IOException
```

Appending Text to a File

To avoid erasing a file that already exists, create a FileWriter object in this manner:

```
FileWriter fw =
    new FileWriter("names.txt", true);
```

Then, create a PrintWriter object in this manner:

```
PrintWriter fw = new PrintWriter(fw);
```

Specifying a File Location

- On a Windows computer, paths contain backslash () characters.
- Remember, if the backslash is used in a string literal, it is the escape character so you must use two of them:

Code.

```
PrintWriter outFile =
      new PrintWriter("A:\\PriceList.txt");
```

Specifying a File Location

- This is only necessary if the backslash is in a string literal.
- If the backslash is in a String object then it will be handled properly.
- Fortunately, Java allows Unix style filenames using the forward slash (/) to separate directories:

Code.

```
PrintWriter outFile = new
    PrintWriter("/home/rharrison/names.txt");
```

Reading Data From a File

Reading Data From a File

If you can use the Scanner class, you can read in data from a file.

```
Scanner keyboard = new Scanner(System.in);
System.out.print("Enter the filename: ");
String filename = keyboard.nextLine();

File file = new File(filename);
Scanner inputFile = new Scanner(file);
```

After that, the input file behaves similar to a person typing into the console.

Reading Data From a File

- The lines above:
  - Creates an instance of the Scanner class to read from the keyboard
  - Prompt the user for a filename
  - Get the filename from the user
  - Create an instance of the File class to represent the file
  - Create an instance of the Scanner class that reads from the file
- Once an instance of Scanner is created, data can be read using the same methods that you have used to read keyboard input (nextLine, nextInt, nextDouble, etc).
  - That's it. So much easier than C++.

Exceptions

- The Scanner class can throw an **IOException** when a File object is passed to its constructor.
- So, we put a throws **IOException** clause in the header of the method that instantiates the Scanner class.

Detecting The End of a File

The Scanner class's hasNext() method will return true if another item can be read from the file.

```
// Open the file.
File file = new File(filename);
Scanner inputFile = new Scanner(file);
// Read until the end of the file.
while (inputFile.hasNext()) {
   String str = inputFile.nextLine();
   System.out.println(str);
}
inputFile.close();// close the file when done.
```

The Random Class

The Random Class

- Some applications, such as games and simulations, require the use of randomly generated numbers.
- The Java API has a class, Random, for this purpose. To use the Random class, use the following import statement and create an instance of the class.

Code.

```
import java.util.Random;

Random randomNumbers = new Random();
```

Some Methods of the Random Class

- nextDouble()
  - Returns the next random number as a double. The number will be within the range of 0.0 and 1.0.
- nextFloat()
  - Returns the next random number as a float. The number will be within the range of 0.0 and 1.0.
- nextInt()
  - Returns the next random number as an int. The number will be within the range of an int, which is -2,147,483,648 to +2,147,483,648.
- nextInt(int n)
  - This method accepts an integer argument, n. It returns a random number as an int. The number will be within the range of 0 to n.