Loops Using Enhanced for (for each iteration)

by Sophia

WHAT'S COVERED

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

1. Enhanced for Loops

The **enhanced for loop**, also known as a **for-each loop** or **range-based** for loop in other programming languages, is a special version of the for loop that is designed to work with Java arrays and collections. It provides a simpler and safer way of iterating over these data structures than using a plain for loop.

Java's enhanced for loop uses the `for` keyword, but the contents of the parentheses are different.

The basic syntactic pattern is:

EXAMPLE

```
for(<variable> : <array/collection>){
<statements>
}
```

This style of for loop iterates the values in `<array/collection>`. Each value is accessible in turn via the `<variable>`. The statements in the body of the loop have access to the `<variable>` and can even change it

in the loop, but changes made in the body of the loop to `<variable>` don't affect the original array or collection.

TERM TO KNOW

Enhanced for Loop (also called a for-each Loop or Range-Based for Loop)
A special version of the Java for loop designed to work with arrays and collections that provides a simpler way to set up a for loop.

2. Enhanced for Loops With Arrays

The enhanced for loop works well with Java arrays because when working with this type of loop, the compiler works out the size of the array. This means the loop will run automatically once for each element in the array without needing to write code that checks the size of the array (the middle term in the three-part definition of a standard for loop).

Here is a sample program that uses an enhanced for loop to loop over the values in an array and carry out calculations using the values:

```
import java.util.Arrays;

class EnhancedForLoop {
  public static void main(String[] args) {
    int[] numbers = {1, 2, 3, 4, 5};
    System.out.println("Values in array: " + Arrays.toString(numbers));
    for(int number : numbers) {
      /* value can be used in statements in loop body */
      System.out.println(number + " * 2 = " + number * 2);
    }
  }
}
```

The program above should produce the following output:

```
Values in array: [1, 2, 3, 4, 5]
1 * 2 = 2
2 * 2 = 4
3 * 2 = 6
4 * 2 = 8
5 * 2 = 10
```

To see how the enhanced for loop makes the code a bit simpler, compare the code above with this version that uses a plain for loop:

```
import java.util.Arrays;

class EnhancedForLoop {
  public static void main(String[] args) {
    int[] numbers = {1, 2, 3, 4, 5};
    System.out.println("Values in array: " + Arrays.toString(numbers));
    for(int number = 0; number < numbers.length; number++) {
      System.out.println(numbers[number] + " * 2 = " + numbers[number] * 2);
    }
  }
}
```

Here is the start of the enhanced loop:

EXAMPLE

```
for(int number : numbers) {
```

Now compare it with the start of the plain for loop:

EXAMPLE

```
for(int number = 0; number < numbers.length; number++) {
```

Note, the following line from the enhanced for loop version simplifies access to the array contents:

EXAMPLE

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

Compare this to the corresponding line in the code using a plain for loop:

EXAMPLE

```
System.out.println(numbers[number] + " * 2 = " + numbers[number] * 2);
```

3. Enhanced for Loops With ArrayLists and HashSets

An enhanced for loop can also be used with some collections. A collection has to be **iterable** to be used with an enhanced for loop. A collection is iterable if the collection provides a mechanism for the items in the collection to be accessed in a fixed order.

An `ArrayList` is iterable, and here is some sample code using an `ArrayList`:

```
import java.util.ArrayList;

class EnhancedForCollection {
  public static void main(String[] args) {
    ArrayList<String> names = new ArrayList<>();
    names.add("Annette");
    names.add("John");
    names.add("Lee");
    System.out.println("ArrayList: " + names.toString());
    for(String name : names) {
      System.out.println(name + " has " + name.length() + " letters.");
    }
  }
}
```

When run, this program produces results like this:

```
ArrayList: [Annette, John, Lee]
Annette has 7 letters.
John has 4 letters.
Lee has 3 letters.
```

The enhanced for loop works with other collection types, too. Here is a version of the previous program using a `HashSet`:

```
import java.util.HashSet;

class EnhancedForCollection {
  public static void main(String[] args) {
    HashSet<String> names = new HashSet<>();
    names.add("Annette");
    names.add("John");
    names.add("Lee");
    System.out.println("HashSet: " + names.toString());
    for(String name : names) {
      System.out.println(name + " has " + name.length() + " letters.");
    }
```

```
  }
}
```

The output from this program should be nearly the same as the previous one showing Annette, John, Lee as the output.

```
HashSet: [Annette, John, Lee]
Annette has 7 letters.
John has 4 letters.
Lee has 3 letters.
```

TERM TO KNOW

Iterable

A collection is iterable if the collection provides a mechanism for the items in the collection to be accessed in a fixed order.

4. Enhanced for Loops With HashMaps

The `HashMap` collection type is not iterable. It cannot be directly looped over using an enhanced for loop. However, there is a workaround. The `HashMap` class has a **.keySet()** method that retrieves the `Set` of keys from the `HashMap`. This `Set` of keys is iterable, so it can be used with an enhanced for loop.

With a key value, the `HashMap`'s `get()` method can be used to get the value corresponding to the key:

```
import java.util.HashMap;

public class EnhancedForCollection {

  public static void main(String[] args) {
    // HashMap holds key-value pairs.
    // The key (user ID) is a String (case sensitive).
    // The value (score) is an Integer (int)
    HashMap<String, Integer> scores = new HashMap<>();
    scores.put("ssmith04", 88);
    scores.put("tlang01", 100);
    scores.put("glewis03", 99);
    System.out.println("HashMap: " + scores.toString());
    /* A HashMap isn't iterable, but the keySet() method returns
       a Set with the keys in the HashMap, which can be interated
       over. The value can be retrieved using the key value
    */
    for(var key : scores.keySet()) {
```

```
      // For each key retrieve the value (score) & print
      System.out.println(key + " has a score of " + scores.get(key) + ".");
    }
  }
}
```

The code above should produce this output displaying scores 88-100 as the output.

```
HashMap: {ssmith04=88, glewis03=99, tlang01=100}
ssmith04 has a score of 88.
glewis03 has a score of 99.
tlang01 has a score of 100.
```

TERM TO KNOW

keySet()

The keySet() method returns a Java Set collection containing the keys used in a HashMap. This Set of keys is iterable, unlike the HashSet itself, which is not.

SUMMARY

In this lesson, you learned about Java's **enhanced for loop**. You learned about the use of **enhanced for loops with arrays**. You also learned about using enhanced for loops to iterate over Java collections such as **ArrayLists, HashSets, and HashMaps**.

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

Enhanced for Loop (also called a For - Each loop or Range-Based for Loop)
A special version of the Java for loop designed to work with arrays and collections that provides a simpler way to set up a for loop.

Iterable
A collection is iterable if the collection provides a mechanism for the items in the collection to be accessed in a fixed order.

keySet()

The keySet() method returns a Java Set collection containing the keys used in a HashMap. This Set of keys is iterable, unlike the HashSet itself, which is not.