**For Each Element, Do Something**

Arrays greatly expand your programming options because you are no longer limited to variables that can hold only one piece of information at a time. Since all of the elements are stored in consecutive memory locations, the array can be treated efficiently as a single unit.

Loops are indispensable tools for handling arrays because algorithms can be used to process the entire data structure. For example, finding the sum of 100 integers. Essentially, what we want to say when adding the elements in an array is "for each integer in the array, add it to the sum." The for loop is very handy for this task, but it can be improved upon. In programming (as you are discovering), it is often true that "for each" solution, there is often an even simpler solution! The for-each loop will be our next focus.

for(int index = 0; index < 10; index++)  
{  
     System.out.println(names[index]);  
}

The block of code inside the opening and closing curly braces is executed until the terminal condition is encountered. In this example, the loop counts from 0 to 9 and prints the information contained at the position in the **names** array indicated by the **index** variable. This **for** loop couldn't be any simpler, unless there was a shortcut.

The traditional **for** loop can very efficiently iterate through the index positions of an array; however, Java introduced an even more specialized **for** loop, specifically to deal with indexed data collections like arrays. This new loop is referred to as an enhanced for loop, or simply a **for-each** loop. The structure and syntax of a **for-each** loop is shown below.

for(String person : names)  
{  
     System.out.println(person);  
}

What happened to the index initialization, condition test, and incrementation? The strategy behind the **for-each** loop is that the index always starts at zero and increments by one, and that all elements in the array will be processed. The IMACS lesson will fully explain the details of the **for-each** loop; it is very useful, but there will still be certain situations in which the traditional **for** loop must be used with arrays.

**EIMACS**

A for-each loop allows a programmer to express the instruction, "give me each element of this array, one at a time" without having to deal with an index at all, and therefore not having to be concerned about declaring, initializing, or updating it. The following code, for example, uses a for-each loop to perform the same task as the above for loop:

String[] names = { "fred", "jane", "eric" };   
  
for ( String name : names )   
{   
  System.out.println( name );   
}

When reading the expression "for ( String name : names )" aloud, it is usual to add the word "each" after "for" and to read the colon as "in"; so the entire expression is read as "for each String name in names".

Since it doesn't involve an explicit index, there is little or no danger of committing an indexing error in a for-each loop. Furthermore, the for-each expression allows the programmer to specify a variable (in this case name) to which each element of the array is assigned in turn. Consequently, there is one less line of code in the body of the loop.

Every for-each loop has four components, none of which may be omitted, as follows:

    for ( declaration : expression )  
      body

|  |  |
| --- | --- |
| for | The Java keyword that, together with the presence of a colon (and the absence of semicolons) in the parenthesized code that follows, identifies this piece of code as a for-each loop. |
| body | **The *loop body*:** a statement or block of statements. This code is executed once during each iteration of the loop. |
| declaration | **The *loop variable declaration*:** the declaration of the variable to which each element of the array or collection that is the value of expression is assigned during successive iterations of the loop. (The declaration has the form of a data type — such as int, String, or boolean — followed by a variable, known as *the loop variable*.) |
| expression | **The *loop collection expression*:** a [statement expression](https://www.eimacs.com/eimacs/mainpage?epid=E201975702&cid=162149#StateExp) such that the value of the corresponding statement is either an array or some other collection that is "iterable". (This array or "iterable" collection is known as *the loop collection*.) |

(We remark that for-each loops are sometimes referred to as "*enhanced for loops*".)

For now, we limit ourselves to arrays when it comes to for-each loop collections. As we advance further in this course, we will meet other kinds of collections to which for-each loops may be applied, and will specify more precisely in what sense they are "iterable".

Warnings:

* The loop variable of a for-each loop is not visible once execution of the loop has terminated. (Recall that the same is true of any variable that is declared within the initialization code or the update code of a for loop.)
* The programmer has no way to control or change the order in which a for-each loop references the elements of the loop collection. For all "iterable" collections this is done with the help of the natural "iterator" for the particular type of collection in question. In the case of arrays, the elements will always be referenced in index order.

Using a for-each loop in place of a for loop results in shorter, more readable code and fewer errors. It is therefore both sensible and advisable to use a for-each loop in preference to a for loop whenever possible. "But when would it *not* be possible to use a for-each loop in place of a for loop," you may be wondering. Here are some of the situations in which a for-each loop is *not* appropriate:

When *Not* To Use A for-each Loop

When *not* iterating over an array (or other "iterable" collection).

For example, the following for loop cannot easily be replaced by a for-each loop:

for ( int i = 0 ; i < 10 ; i++ )   
  System.out.println( i );

When an index is needed for some purpose other than referencing the next element.

As, for example, in the following for loop:

int[] values = { 1, 1, 2, 4, 7, 6 };   
  
for ( int i = 0 ; i < values.length ; i++ )   
{   
  int v = values[ i ];   
  
  if ( v == i )   
    System.out.println( 0 );   
  else    
    System.out.println( v );   
}

When an index is used to reference the elements of a collection in a particular order or a particular selection from among those elements.

For example, the following for loop cannot easily be replaced by a for-each loop:

int[] values = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };   
  
for ( int i = values.length - 1 ; i > 0 ; i /= 2 )   
{   
  System.out.println( values[ i ] );   
}

When values are to be assigned to the elements of a collection.

As, for example, in this code:

int[] values = { 1, 1, 2, 4, 7, 6 };   
  
for ( int i = 0 ; i < values.length ; i++ )   
{   
  int v = values[ i ];   
  
  if ( v == i )   
    values[ i ] = 0;   
  
  System.out.println( values[ i ] );   
}

In the last of the above examples, where our intention is to assign values to elements of the array, you might be tempted to use a for-each loop and simply make assignments to the loop variable, as in the following exercise.

### Part 1

The **for-each** loop occupies a specialized niche among the loops; it specializes in iterating through data structures. While **for-each** does provide a shortcut, there are still situations when a traditional **for** loop is called for. Like any good programmer, you will develop habits based on the techniques you prefer. Whether you use the traditional **for** loop, or the new enhanced loop is up to you. Programmers like to have options.

### Part 2

The **for-each** loop is streamlined to do just one thing, iterate through a data structure. Just remember to read the colon (:) as "in" when you say the statement aloud. For example, the following line

for(String person : names)

would be spoken as, "for each person in names." Therefore, the meaning of this statement is that the loop will **traverse** the names array from beginning to end and assign each element one-at-a-time to a temporary **String** variable called **person**.

The context of each programming situation will determine which **for** statement is appropriate to use. The enhanced **for** statement has several advantages.

* No index variable is needed.
* No condition needs to be stated.
* The index variable does not need to be incremented.
* It works on any data structure, regardless of the length or size.
* An ArrayIndexOutOfBoundsException will never be thrown.

However, specialization often has consequences, which are not necessarily disadvantages of the **for-each** loop, just limitations to recognize. For example, enhanced **for** loops cannot be used in the following circumstances:

* When not iterating over an array (or other "iterable" collection).
* When an index is needed for some purpose other than referencing the next element.
* When an index is used to reference the elements of a collection in a particular order or a particular selection from among those elements.
* When values are to be assigned to the elements of a collection.

The traditional **for** loop can always replace an enhanced **for** loop and you may be tempted just to stick with them; however, using appropriate shortcuts distinguishes you from the novice programmer. Besides, the **for-each** loop is covered on the AP Exam, so you need to know how to use them.

#### Answer Key

Top of Form

#### Question 1 (Worth 4 points)

In reference to for-each loops, which of the following statements is true?

1. for-each loops can never be rewritten as for loops
2. an index variable is NOT explicitly coded
3. an ArrayIndexOutOfBoundsException will **never** be thrown

 I and II only

 I and III only

 II and III only

 I, II, and III

 None of the statements are true

Points earned on this question: **4**

#### Question 2 (Worth 4 points)

Which is an inappropriate use of a for-each loop?

1. when replacing values within an array
2. when all values in an array need to be traversed
3. when an index variable is required to complete a task

 I only

 II only

 III only

 I and III only

 II and III only

Points earned on this question: **0**

#### Question 3 (Worth 4 points)

Assume an array of integers named arrNum has been properly declared and initialized. Which of these code segments will result in displaying the same output?

1. int i = 0;  
   while (i <= arrNum.length)  
   {  
      System.out.println(arrNum[i]);  
      i++;  
   }
2. for (int i = 0; i < arrNum.length; i++)  
   {  
      System.out.println(arrNum[i]);  
   }
3. for (int a : arrNum)  
   {  
      System.out.println(a);  
   }

 I and II only

 II and III only

 I and III only

 All three print the same results.

 All three print different results.

Points earned on this question: **0**

#### Question 4 (Worth 4 points)

Consider the following code segment. What is the value of x after execution of the code?  
  
double [] numList = {4.17, 3.11, 2.46, 1.57};  
int x = 0;  
for(double n : numList)  
{  
   x += (int)n + 1;  
}

 2.57

 9

 14

 15

 15.3

Points earned on this question: **4**

#### Question 5 (Worth 4 points)

What is the output after the following code segment is executed?  
  
String[] mascots = { "Knights", "Seminoles", "Bulls", "Gators" };  
int n = −1;  
String str = "";  
  
for (String m : mascots)  
{  
   n = m.indexOf("l") + 2;  
   str += m.substring(n);  
}  
System.out.println(str);

 nssa

 ightstors

  nightsssators

 KnightseslsGators

 StringIndexOutOfBoundsException occurs

Points earned on this question: **4**

Bottom of Form