**Traversing Through**

A [traversal](javascript:void(0);) is simply an iteration through an array or an ArrayList. You can consider it similar to flipping through the pages of a book. While traversing the book, you can write down vocabulary words and their definitions to help build a glossary.

Since a traversal involves iteration, it naturally will involve a loop of some sort. If you think back to when you first worked with arrays and ArrayLists, this makes perfect sense. Loops and data structures go hand in hand.

By now, you have already traversed arrays many times. After all, the best way to manipulate an array is to iterate through it using a loop. In fact, you have already done the same thing with an ArrayList. What you need to understand now is that this process is actually a basic algorithm used by programmers in every programming language that uses data structures. You cannot make use of an array or ArrayList without traversing through it in one fashion or another.

### Part 1

Traversing should seem very familiar to you; it is the fundamental algorithm for traversing through data structures. After exploring some additional details of traversing in the eIMACS labs, you will explore its application in several simple demo programs.

We now turn our attention away from learning how to write programs in the Java language and focus instead on how the language may be used to help us work with data. In particular we consider the various strategies or [algorithms](javascript:void(0);) that may be used to inspect and manipulate data.

In this section, our study of algorithms will take place in the context of arrays of objects. These algorithms fall into three groups:

* basic operations (including traversals, insertions, and deletions),
* searching (including sequential and binary searches), and
* sorting (including selection, insertion, and merge sorts, and — optionally, for students who enjoy a challenge — quicksort and heapsort).

You are already familiar with several of the techniques we consider here, because we have been incorporating them into the programming exercises you have completed to this point. Our purpose now, though, is to turn the spotlight on the techniques themselves and to present them in the larger context of the study and analysis of algorithms.

We begin with an example aimed at clarifying what we mean by "algorithm" and at demonstrating how the careful selection of an algorithm may significantly affect the performance of a program.

Consider the following problem:

If m and n are integers such that m is less than or equal to n, then we use the phrase "the interval from m to n" to refer to the collection of all the integers from m through n, inclusive. For example, "the interval from 4 to 9" refers to this collection of integers: 4, 5, 6, 7, 8, and 9. Similarly, the interval from 112 to 199 consists of the integers 112, 113, 114, and so on all the way up through 197, 198, and 199. We would like to write a method that takes integers m and n as arguments (where m is less than or equal to n) and calculates the sum of all the integers in the interval from m to n.

Here is one possibility. The strategy (or algorithm) it uses is simply to take the integers in the interval one at a time and add each one onto a running total. Once the final integer has been added, the running total is the looked-for sum.

  private static int sumInterval( int m, int n )   
  {   
    int t = 0;   
    for ( int i = m ; i <= n ; i++ )   
      t += i;   
    return t;   
  }   
  
  public static void main( String[] args )   
  {   
    System.out.println( sumInterval( , ) );   
  }

However, there is a well-known formula for calculating the sum of the integers in an interval. It is this:

One-half of the result of multiplying the number of integers in the interval by the sum of the first and last integers.

In the case of the interval from m to n, the interval contains n – m + 1 integers, and the first and last integers are m and n, respectively. So, according to the formula,

sum = [(n – m + 1) \* (m + n)] / 2

In our particular case, it follows that the sum of the integers from 10 through 199 is:

[(199 – 10 + 1) \* (10 + 199)] / 2  
     = [190 \* 209] / 2 = 39710 / 2 = 19855.

This algorithm may be coded as follows:

  private static int sumInterval( int m, int n )   
  {   
    return ( (n - m + 1) \* (m + n) ) / 2;   
  }   
  
  public static void main( String[] args )   
  {   
    System.out.println( sumInterval( , ) );   
  }

Both algorithms achieve the desired result. How do they compare?

* The first algorithm is intuitively simple, and requires less knowledge of the mathematics involved. It is so simple in fact that it should be possible just to look at the method's definition and *see* that it correctly encodes the algorithm; we should be able to *see* that each integer in turn, from the first to the last, is added to the growing total and the final result reported. Because of this simplicity, it is very likely that a programmer, when faced with the task of coding the first algorithm, would be able to do so without making any mistakes.
* The second algorithm gives rise to a method that usually has much less work to do and is therefore probably much more efficient than the first. Provided the integers in question are not so large that an overflow occurs, the second method most likely executes very quickly because, regardless of the input integers, only five straightforward integer arithmetic operations are required. This means in particular that the execution time should be more or less constant, no matter how far apart *m* and *n* are. This contrasts with the first method, in which the further apart *m* and *n* are, the more additions have to be performed and therefore the longer the computation takes.

Finding the most appropriate algorithm for the given task is a vital step in designing a program. To get a sense of just how complex such a decision may be, visit the [Dictionary of Algorithms and Data Structures (DADS)](http://www.nist.gov/dads) internet web site provided by the [National Institute of Standards and Technology (NIST)](http://www.nist.gov). In the pages that follow, we study several of the most important algorithms described on the DADS web site.

### Part 2

Now that you have gained additional knowledge related to traversing, practice with an inventory sample using an array and an ArrayList.

* Open the [07.04 Virtual Lecture Notes](https://lti.flvsgl.com/flvs-cat-content/pf51e0c4chtrumk87a6ebdda66/flvs-cat-session/apcomputersciencea_v20/module07/lesson04/pop/16_01b/16_01_virtual_lecture_notes.htm).
* Create a new project called 07.04 Traverse in the Mod07 Lessons folder.
* Download the following Java files to the newly-created folder:
  + [InventoryItem.java](https://lti.flvsgl.com/flvs-cat-content/pf51e0c4chtrumk87a6ebdda66/flvs-cat-session/apcomputersciencea_v20/module07/lesson04/docs/16_01b/InventoryItem.java)
  + [TestInventory1.java](https://lti.flvsgl.com/flvs-cat-content/pf51e0c4chtrumk87a6ebdda66/flvs-cat-session/apcomputersciencea_v20/module07/lesson04/docs/16_01b/TestInventory1.java)
  + [TestInventory2.java](https://lti.flvsgl.com/flvs-cat-content/pf51e0c4chtrumk87a6ebdda66/flvs-cat-session/apcomputersciencea_v20/module07/lesson04/docs/16_01b/TestInventory2.java)
* Carefully read the discussion and carry out the instructions.

### Part 3

Before moving on, a reminder about the toString method may be helpful. Any class that does not explicitly extend another class automatically extends the Object class. In practical terms, this means that every object you create automatically has a toString method. The toString method returns a String representation of an object, which can be printed and is very useful when debugging programs.

* Download the [TestToString.java](https://lti.flvsgl.com/flvs-cat-content/pf51e0c4chtrumk87a6ebdda66/flvs-cat-session/apcomputersciencea_v20/module07/lesson04/docs/16_01b/TestToString.java) file to your Mod07 Lessons folder.
* Do a quick desk check of the program and specifically note the toString method.
* Run the program and observe the output.

In this program, the toString statement is actually invoking the toString method to print the contents of the objects. This is a very, very useful technique.