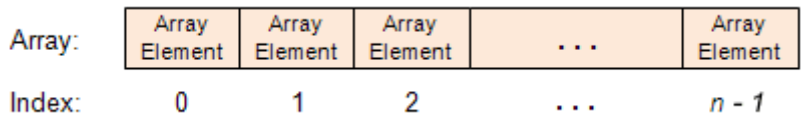
Chapter 2: Array Structure

Array Structure

We discussed arrays briefly in the first Java course and even more briefly in the first lesson of this course. It's time to settle down and see what makes arrays work (and how to make them work for us)! Almost every programming language has some form of this data structure, which is a good sign that it's a very useful structure.

First, let's go over the definition of an array. In Java, an *array* is a group of elements of the same type that are stored together in memory and accessed by using an index. So as we discussed in the first lesson, an array name by itself refers to the array, while the name followed by an index refers to just one element of an array.

Since a picture is worth a thousand words, let me draw one of a hypothetical array here:



This drawing represents the memory where an array with *n* elements is stored. The array's elements are stored contiguously, and we can use them individually via the index in square brackets. For example, if the array stores character values, and its name is *c*, then using `c` in a program will refer to the array, while using `c [0]` will refer to the first character in the array; `c [4]` will refer to the fifth character, and `c [n-1]` will refer to the last character in the array.

Since there's no better way to learn something than to practice, let's develop a program that uses an array. Along the way, we'll learn about some other aspects of Java.

We'll start out simple. Let's say I want to keep a basketball team roster. (I haven't done this since my daughter finished playing college ball, but it's something I'm familiar with and can develop pretty easily. So I'm going with it.) To start with, we'll just make a list of the players on the team. How can we do that? (Hint: This is a lesson on arrays. It would be a good guess to plan on using one.)

Since we want to store the names of the players, we'll make it an array of strings. Just to make sure we have enough room, our default constructor will allow for 20 players on our imaginary team. So we'll declare our array like this:

```
String[] teamArray = new String[20];
```

Now, just to get a feel for how to use an array, we're going to write a simple program to allow a user to enter player names and print a roster. Let's start by making a class called *Team* that will store and retrieve our roster. Let me show you my first version of the class, then we'll go over the details of what it does and how. (Note that I'm leaving out many of the comments I would normally include in the interest of saving space.)

```
public class Team

{

    private String[] roster; // array for roster

    private int teamSize;    // number of players in roster




    public Team()

    {

        roster = new String[20]; // create array

        teamSize = 0;            // initialize team size

    }




    public Team(int arraySize)

    {

        roster = new String[arraySize]; // create array

        teamSize = 0;     // initialize team size

    }




public void addPlayer(String playerName)

{

    if (teamSize < roster.length)

    {

        roster[teamSize] = playerName; // add player to roster

        teamSize++;                    // increment team size

    }

}




public String toString()

{

    String teamRoster = "Team Roster\n\n"; // output String

    int i = 0;                             // loop counter

    while (i < teamSize)                   // while more players

    {

        teamRoster = teamRoster + roster[i] + "\n"; // add name to roster

        i++;                               // increment loop counter

    }




    return teamRoster;                    // return roster
```

```
    }

}
```

Here's how the class works. The first two statements in the class declare a String array named *roster* to hold the team roster, and an int variable named *teamSize* to keep track of how many names are in the roster.

This is a good place to point out that once you create an array, its size is fixed. An array with 20 elements *always* has room for exactly 20 elements, whether you're using all or none of them. The only way to keep track of how many array elements actually contain information is to do it yourself. That's what we'll use the variable teamSize for.

Team Class Constructors

The default class constructor creates an array with room for 20 strings, then initializes the team size to zero. There's a second constructor that allows the client class to set the size of the array if 20 is too big or too small.

The `addPlayer()` Method

The *addPlayer()* method does two things if the array isn't already full. (The *if* statement checks for that by comparing the team size to the array size using a field that's built into every array in Java, the *length* field.) As long as there's room left in the array, the addPlayer() method puts the contents of the string argument *playerName* into the next available spot in the roster array. It uses the variable teamSize as the index pointing to the next spot.

Once the name is stored in the array, teamSize is incremented by one. The *++* operator in Java is called the *increment operator*, and it will add one to whatever variable you use it with.

The `toString()` Method

The *toString()* method formats the array's contents into an output string. This method is standard in many Java classes and is always designed to prepare the class's data for output. The toString() method does that for our Team class using a loop to process each name in the array with the index *i*, adding each name to the end of the output string named *teamRoster*. When used with strings, the plus sign becomes the *concatenation operator*, which *concatenates*, or links together, two strings by adding the second to the end of the first.

The other item that might be new to you is the *\n* in the string. That's the *newline* constant. Wherever it appears in a string is where a new line will begin when that string is displayed or printed.

If my explanation of this class leaves any questions in your mind, please let me know in the Discussion Area and I'll be glad to explain more.