Virtual Lecture Notes (Part 2)

Numeric Array Basics

The syntax for declaring and initializing numeric arrays is the same as for **`String`** arrays.

```
int [] intValues;
intValues = new int[10];
```

or

```
int maxIndex = 100;
double [] doubleValues = new double[maxIndex];
```

The number of positions in an array can be assigned with an actual value (e.g., 10) or with a variable (e.g., **`maxIndex`**). The context of each program will indicate whether to use values or variables, and whether to declare and initialize an array with one statement or two. Before anything is assigned to an array, each index position is initialized with a placeholder indicating it is empty. Copy the following program into the 6.01 Numeric Arrays project, to determine what value is assigned to each index position when a numeric array is created. Part of the code has been deleted to make it a little more challenging, so decide how many index positions the array will contain. Copy this program into BlueJ and fill in the missing pieces, which are blacked out below. (Use the String array example as a model)

```
public class InitializingNumericgArray
{
    public static void main(String [] args)
    {
        int ■■ intValues;
        intValues = ■■■ int[10];

        for(int n = 0; n <= ■; n++)
        {
            System.out.println("index position " + n + " = "
                               + intValues[■]);
        }
    }
}
```

After you have filled in the missing pieces for an **`int`** array, change it to a **`double`** array. What do you predict will be the initial value of the index positions in a **`double`** array? Find out if you are correct.

During the design of a program, pay close attention to how an array will be used. It is possible that the initial content of some array positions may never change. For example, if you were finding the average of the values in an array, depending on the purpose of the program, counting the zeros in any unused positions may not be appropriate.

Once a numeric array is declared and initialized, values can be assigned to index positions using the same four techniques you learned about with **`String`** arrays.

Direct Assignment within the Source Code

Create a class called NumericArrayDemo1 of length 10 (modeled after StringArrayDemo1, but with an **`int`** array) and directly assign 10 integers of your choice to the array. Print the value of each index position to verify the program works correctly. Run the program and observe the output. Once it works for integers, change the program so it will work with **`doubles`**. Experiment with the program on your own.

- What is the biggest numeric array you can create? Since the index variable is an **`int`**, look back in an earlier lesson for the table showing the range of values for each primitive data type. Try creating the largest array possible, but don't bother to try and put in that many numbers!

Direct Assignment during Array Initialization

Create a class called NumericArrayDemo2 class with 20 indexed positions (modeled after StringArrayDemo2, but with a **`double`** array) and directly assign 20 decimal values during array initialization. Print the value of each index position to verify the program works correctly. Run the program and observe the output. Once it works for decimals, change the program so it will work with **`ints`**. Experiment with the program on your own.

- What happens if you try to assign **`doubles`** to an **`int`** array and vice versa?

User Assignment from the Keyboard

Create a class called NumericArrayDemo3 class with five indexed positions (modeled after StringArrayDemo3, but with a **`double`** array) and assign five decimal values to the array. Print the value of each index position to verify the program works correctly. Run the program and observe the output. Once it works for decimals, change the program so it will work with **`ints`**. Experiment with the program on your own.

- What happens if you try to assign **`doubles`** to an **`int`** array and vice versa?

Read Array Data from a File

Create a class called NumericArrayDemo4 class (modeled after StringArrayDemo4, but with an **`int`** array) and read the numbers in the **`int.txt`** file into the array. Print the value of each index position to verify the program works correctly. Run the program and observe the output. Once it works for integers, change the program so that it will read the **`doubles.txt`** file. Experiment with the program on your own.

- What happens if you try to read **`Strings`** into a numeric array?
- Determine the number of index positions with the array's length property.

Finding the Sum of Array Elements

One of the most common programming tasks is calculating the sum of the elements in an array. You will use this over and over again throughout the course. In earlier programs, simple variables that could only hold one value at a time were used, so you calculated sums by adding values as they were read in from a file or entered from the keyboard. With arrays, tasks can be broken down into smaller tasks, because all of the data is simultaneously stored in the memory. Consequently, the array can be traversed from beginning to end, adding each element to the sum. Take a close look at the following code segment. Can you complete the statement that accumulates the sum of the numbers assigned the **`intValues[]`** array?

```
…
int [] intValues = {5, 6, -12, 73, 4, -25, 15, 8, 0, -2};

int sum = 0;
for (int index = 0; index <= 9; index++)
     sum += _________;
System.out.println("Sum = " + sum);
…
```

What variable contains the value the integers in the array? Try writing a quick demo program to check your understanding of how to find the sum of array elements.

Finding the Maximum and Minimum Values in an Array

Another common array processing task is to determine the maximum and minimum values of an array. For example, simply by visually scanning the array in the previous example, you can instantly tell that 73 is the maximum value and -25 is the minimum value. But, how can you write the Java code to select the maximum and minimum values? Pretend you are the computer, traversing the array from beginning to end. What decisions do you have to make in order to identify the largest number and the smallest number? What do you with a number when it is the maximum or minimum? Can you translate your ideas into code? Give it a try using the short program you wrote to calculate the sum of the array. It only involves a few simple statements. You know how to do it, you just need to translate the algorithm into Java.