Lecture 18
Introduction to \textit{array}: why use arrays?

- **Problem:**
  
  - Write a program that reads in and stores away \textit{5} \textit{double} numbers.
  
  - After reading in the numbers, the program computes the \textit{average} of the numbers.

- **Solution:** (Java program)

```java
import java.util.Scanner;

public class Avg1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        double a, b, c, d, e;   // Hold 5 numbers
        double avg;

        System.out.println("Enter a number: ");
        a = in.nextDouble();   // Read in number

        System.out.println("Enter a number: ");
        b = in.nextDouble();   // Read in number

        System.out.println("Enter a number: ");
        c = in.nextDouble();   // Read in number

        System.out.println("Enter a number: ");
        d = in.nextDouble();   // Read in number

        System.out.println("Enter a number: ");
        e = in.nextDouble();   // Read in number

        avg = (a + b + c + d + e)/5;    // Print average
        System.out.println(avg);
    }
}
Problem with this solution:

- It is not scalable

The program becomes unmanageably large if the input size is large

Solution:

- Organize the storage of the information in such a way that you can access the information through an index:

```
Array:

23  4  6  15  5  7

0  1  2  3  4  5
```

This kind of organized data is called an array data structure
Java program using an array:

```java
import java.util.Scanner;

public class Avg2 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        double[] a;
        a = new double[5]; // Define an array of 5 elements

        double sum, avg;
        int i; // index

        for (i = 0; i <= 4; i++) {
            System.out.print("Enter a number: ");
            a[i] = in.nextDouble(); // Read in number
        }

        /* --------------------------------------------------------
        Use the "running sum" algorithm to compute total
        -------------------------------------------------------- */
        sum = 0.0;

        for (i = 0; i <= 4; i++) {
            sum = sum + a[i];
        }
        avg = sum/5;
        System.out.println(avg); // Print average
    }
}
```
The **array** data structure

- An array is a *collection (multiple) of variables* where:
  - Each variable in the collection is of the *same data type*
  - Note: that means that the *size (number of bytes)* of each variable is *the same*
  - The variables are placed (stored) *consecutively in memory*

Example:

![Array Diagram](image)
Declaring Array Variables

- `elementType[] arrayRefVar;`
- Or
- `elementType arrayRefVar[];`
Creating Arrays

• Unlike declarations for primitive data type variables, the declaration of an array variable does not allocate any space in memory for the array. It creates only a storage location for the reference to an array. If a variable does not contain a reference to an array, the value of the variable is null. You cannot assign elements to an array unless it has already been created. After an array variable is declared, you can create an array by using the new operator with the following syntax:
  – arrayRefVar = new elementType[arraySize];
• Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement as:

– `elementType[] arrayRefVar = new elementType[arraySize];`

or

– `elementType arrayRefVar[] = new elementType[arraySize];`
• To assign values to the elements, use the syntax:
  – arrayRefVar[index] = value;
• For example, the following code initializes the array.
• myList[0] = 5.6;
• myList[1] = 4.5;
• myList[2] = 3.3;
• myList[3] = 13.2;
• myList[4] = 4.0;
• myList[5] = 34.33;
• myList[6] = 34.0;
• myList[7] = 45.45;
• myList[8] = 99.993;
• myList[9] = 11123;
Array Size and Default Values

• When space for an array is allocated, the array size must be given, specifying the number of elements that can be stored in it. The size of an array cannot be changed after the array is created. Size can be obtained using `arrayRefVar.length`. For example, `myList.length` is 10.

• When an array is created, its elements are assigned the default value of 0 for the numeric primitive data types, \u0000 for `char` types, and `false` for `boolean` types.
Array Indexed Variables

• The array elements are accessed through the index. Array indices are 0 based; that is, they range from 0 to `arrayRefVar.length-1`. In the example in Figure 6.1, `myList` holds ten `double` values, and the indices are from 0 to 9.
Array Initializers

- Java has a shorthand notation, known as the *array initializer*, which combines the declaration, creation, and initialization of an array in one statement using the following syntax:

  ```java
  elementType[] arrayRefVar = {value0, value1, ..., valuek};
  ```

  For example, the statement

  ```java
  double[] myList = {1.9, 2.9, 3.4, 3.5};
  ```

  declares, creates, and initializes the array `myList` with four elements, which is equivalent to

  - the following statements:

  ```java
  double[] myList = new double[4];
  myList[0] = 1.9;
  myList[1] = 2.9;
  myList[2] = 3.4;
  myList[3] = 3.5;
  ```

  **Caution**

  The `new` operator is not used in the array-initializer syntax. Using an array initializer, you have to declare, create, and initialize the array all in one statement. Splitting it would cause a syntax error. Thus, the next statement is wrong:

  ```java
  double[] myList;
  myList = {1.9, 2.9, 3.4, 3.5};
  ```
For loop

• **for** (int i = 0; i < myList.length; i++){
  //TODO
}

• }
For-each loop

```javascript
for (elementType element: arrayRefVar) {
    // Process the element
}
```
• char array and String.
• char[] v.s. String