Object Oriented Programming

(CSC 302)

Lecture Note 4

How to work with Arrays

4th December, 2019

How to work with arrays

- An **array** is an object that contains one or more items called **elements**. Each element within an array can be a *primitive* type such as an **int** value or a **reference** type such as a **String** object or a **Student** object.
- To create an array, you must declare a variable of the correct type and instantiate an array object that the variable refers to. You can declare and instantiate the array in separate statements, or you can combine the declaration and instantiation into a single statement.
- To declare an array variable, you code a set of empty brackets after the type or the variable name.
- To instantiate an array, you use the **new** keyword and specify the *length*, or *size*, of the array in *brackets* following the array type.
- When you instantiate an array of primitive types, numeric types are set to **zeros** and boolean types to **false**. When you create an array of reference types, they are set to **nulls**.
- The syntax for declaring and instantiating an array

  type[] arrayName;

  arrayName = **new type[**length**];** or

  type[] arrayName = **new type[**length**];**

- An example that declares and instantiates an array of double values

```
double[] prices;
prices = new double[4]; or
double[] prices = new double[4];
```

- An array of String objects

```
String[] titles = new String[3];
```

- An array of Product objects

```
Product[] products = new Product[5];
```

- Code that uses a constant to specify the array length

```
final int TITLE_COUNT = 100;
String[] titles = new String[TITLE_COUNT];
```

- Code that uses a variable to specify the array length

```
System.out.print("Enter number of titles: ");
int titleCount = Integer.parseInt(sc.nextLine());
String[] titles = new String[titleCount];
```

- How to assign values to the elements of an array
  - To refer to the elements in an array, you use an **index** that ranges from zero (the first element in the array) to one less than the number of elements in the array.
  - If you specify an index that's less than zero or greater than the upper bound of the array, Java throws an **ArrayIndexOutOfBoundsException**.
  - You can instantiate an array and provide initial values in a single statement by listing the values in braces. The number of values you provide determines the size of the array.
- The syntax for referring to an element of an array

  arrayName[index]
- Code that assigns values to an array of double types

```
double[] prices = new double[4];
prices[0] = 14.95;
prices[1] = 12.95;
prices[2] = 11.95;
prices[3] = 9.95;
//prices[4] = 8.95;     // this would throw ArrayIndexOutOfBoundsException
```

- Code that assigns values to an array of String types

```
String[] names = new String[3];
names[0] = "Ted Lewis";
names[1] = "Sue Jones";
names[2] = "Ray Thomas";
```

- Code that assigns objects to an array of Product objects

```
Product[] products = new Product[2];
products[0] = ProductDB.getProduct("java");
products[1] = ProductDB.getProduct("jsp");
```

- The syntax for creating an array and assigning values in one statement

```
type[] arrayName = {value1, value2, value3, ...};
```

- Examples that create an array and assign values in one statement

```
double[] prices = {14.95, 12.95, 11.95, 9.95};
String[] names = {"Ted Lewis", "Sue Jones", "Ray Thomas"};
Product[] products = {
        ProductDB.getProduct("java"),
        ProductDB.getProduct("jsp")
};
```

- How to use for loops with arrays
  - You can use for loops to process each element in an array.
  - You can use the length field of an array to determine the number of elements in the array.
- The syntax for getting the length of an array

```
arrayName.length
```

- Code that puts the numbers 0 through 9 in an array

```
int[] values = new int[10];
for (int i = 0; i < values.length; i++) {
        values[i] = i;
}
```

- Code that prints an array of prices to the console

```
double[] prices = {14.95, 12.95, 11.95, 9.95};
for (int i = 0; i < prices.length; i++) {
        System.out.println(prices[i]);
}
```

- Code that computes the average of the array of prices

```
double[] prices = {14.95, 12.95, 11.95, 9.95};
double sum = 0.0;
for (int i = 0; i < prices.length; i++) {
        sum += prices[i];
}
double average = sum/prices.length;
```

- How to use enhanced for loops with arrays
  - Java 5 introduced the enhanced for loop. This loop simplifies the code required to loop through arrays. The enhanced for loop is sometimes called a **foreach** loop because it lets you process each element of an array.
  - Within the parentheses of an enhanced for loop, you declare a variable with the same type as the array followed by a colon and the name of the array.
  - With each iteration of the loop, the variable that's declared by the for loop is assigned the value of the next element in the array. You can also use enhanced for loops to work with collections.

- The syntax of the enhanced for loop

```
for (type variableName : arrayName) {
        statements
}
```

- Code that prints an array of prices to the console

```
double[] prices = {14.95, 12.95, 11.95, 9.95};
for (double price : prices) {
        System.out.println(price);
}
```

- Code that computes the average of the array of prices

```
double[] prices = {14.95, 12.95, 11.95, 9.95};
double sum = 0.0;
for (double price : prices) {
        sum += price;
}
double average = sum / prices.length;
```

- How to work with two-dimensional arrays
  - Two-dimensional arrays use two indexes and allow data to be stored in a table that consists of **rows** and **columns**. This can also be thought of as an array of arrays where each row is a separate array of columns.
  - Most two-dimensional arrays are **rectangular** arrays where each row has the same number of columns. However, it's also possible to create **jagged** arrays where each row may have a different number of columns.
  - If necessary, you can extend this two-dimensional syntax to work with arrays that have more than two dimensions.
- How to create a rectangular array
  - The syntax for creating a rectangular array

    ```
    type[][] arrayName = new type[rowCount][columnCount];
    ```
- A statement that creates a 3x2 array

  ```
  int[][] numbersTable = new int[3][2];
  ```
- How to assign values to a rectangular array
  - The syntax for referring to an element of a rectangular array

    ```
    arrayName[rowIndex][columnIndex]
    ```

- The indexes for a 3x2 array

  [0][0] [1][0] [2][0]

  [0][1] [1][1] [2][1]

- Code that assigns values to the array

```
numbersTable[0][0] = 1; numbersTable[0][1] = 2;
numbersTable[1][0] = 3; numbersTable[1][1] = 4;
numbersTable[2][0] = 5; numbersTable[2][1] = 6;
```

- Code that creates a 3x2 array and initializes it in one statement

```
int[][]numbersTable = { {1,2} {3,4} {5,6} };
```

- How to use nested for loops to process a rectangular array

```
int[][] numbersTable = { {1,2}, {3,4}, {5,6} };
for (int[] row : numbersTable) {
    for (int column : row) {
        System.out.print(column + " ");
    }
    System.out.print("\n");
}
```

- How to use the Arrays class
- The Arrays class
  - java.util.Arrays
  - **fill(array, value)** - Fills all elements of the specified array with the specified value.
  - **sort(array)** - Sorts the elements of an array into ascending order.
  - **binarySearch(array, value)** - Returns an *int* value for the index of the specified value in the specified array. Returns **-1** if the specified value is not found in the array. For this method to work correctly, the array must first be sorted by the sort method.
- All of these methods accept arrays of primitive data types and arrays of objects for the array argument, and they all accept primitive types and objects for the value argument.
- For the sort method to work correctly with an array of objects created from a user-defined class, such as the Student class, the class must implement the Comparable interface.

- How to fill an array

```
int[] quantities = new int[5];
Arrays.fill(quantities, 1);
```

- How to sort an array

```
int[] numbers = {2,6,4,1,8,5,9,3,7,0};
Arrays.sort(numbers);
for (int number : numbers) {
        System.out.print(number + " ");
}
```

- How to search an array

```
String[] productCodes = {"mysql", "jsp", "java"};
Arrays.sort(productCodes);
int index = Arrays.binarySearch(productCodes, "mysql");
```

- How to copy an array
  - Like a string, an array is a **reference** type.
  - To create a reference to an existing array, you can use the assignment operator **(=)** to assign a variable that points to an existing array to another variable. Then, both variables point to the same array.
  - To check if two array variables refer to the same array, you can use the equality operator **(==)**.
  - The **copyOf** method was introduced with Java 6. As a result, it only works with Java 6 and later. Prior to Java 6, you can use the **arraycopy** method of the **System** class.
  - When you copy an array, the new array must be the same type as the source array.
- More static methods of the Arrays class

copyOf(array, length) -Copies the specified array, truncating or padding with default values as necessary so the copy has the specified length.

equals(array1, array2) - Returns a *boolean* **true** value if both arrays are of the same type and all of the elements within the arrays are equal to each other.

- How to create a reference to an array

```
double[] grades = {92.3, 88.0, 95.2, 90.5};
double[] percentages = grades;
percentages[1] = 70.2;
System.out.println("grades[1]=" + grades[1]);
```

- How to create a shallow copy of an array (Java 6 or later)

```
double[] grades = {92.3, 88.0, 95.2, 90.5};
double[] percentages = Arrays.copyOf(grades, grades.length);
percentages[1] = 70.2;
System.out.println("grades[1]=" + grades[1]);
```

- How to determine if two variables refer to the same array

```
if (grades == percentages) {
        System.out.println("Both variables refer to the same array.");
} else {
        System.out.println("Each variable refers to a different array.");
        System.out.println("However, these arrays may contain the same data.");
}
```

- How to determine if two variables contain the same data

```
if (Arrays.equals(grades, percentages)) {
        System.out.println("Both variables refer to the same data.");
} else {
        System.out.println("Both variables do not refer to the same data.");
}
```