# CIS 3145 Class Notes: Text Chapter 10

## Arrays with Loops

**Objectives**

* Create and assign values to arrays
* Use array methods

Arrays allow for the **Dynamic** **Declaration** of Variables: 1 to many variables can be declared dynamically

* Use the Comparable interface to Sort an array
* Be able to reference and copy arrays
* Use two dimensional arrays.

**Arrays**

Arrays are just like regular variables but they are a group of many variables. All of the variables in the array are of the same **data** or **object** type. Each variable in the array is called an ‘***Element*’**. The location in the array is defined by the ‘***Index*’**. The first element of an array has an index value of zero, thus arrays in java are said to be ***zero-based***. The index is an **Integer** data type, and is represented by a literal value (‘0’) or a variable (int i = 0).

The value of the Index can’t be less than 0 **or** equal to or larger than the **length** of the array. This would cause an *ArrayIndexOutOfBoundsException* to be thrown. The length is a property of the array; it is not a method.

**Declare and Instantiate an array**

Use ‘square brackets’ to show that the variable is an array. Name an array with a **plural** name since it represents **many** values.

Just like objects, arrays

1. need a *variable* declared to hold the array,

String [] **myArray**;

double **myDoubleArray** [];

1. need to be *instantiated* with the new method and

**new** String [2]

**new** double [Size]

1. need the array object to be *assigned* to the variable with the equal symbol.

myArray = new String [2];

double myDoubleArray [] = new double [Size];

The length of the array is an **integer** value or **variable** used when the array is instantiated. The “Size” integer variable above sets the length of the array.

**Assigning values to an array**

Values in an array can be assigned **one at a time**:

String[] mySuits = new String [4];

mySuits[0] = "Spade";

mySuits[1] = "Heart";

mySuits[2] = "Diamond";

mySuits[3] = "Club";

Or **all in one statement** when the array is first declared:

String[] suits = {"S", "H","D","C"};

When data is input from a user in a while loop a ‘**counter’** index variable is used to count the number of inputs that are put into the array.

String itemName = sc.next();

itemArray[counter] = itemName;

counter++;

**Loops (Regular and Enhanced)**

Loops allow us to **process** every element in an array. This is a wonderful thing that loops can do. We can print each element or sum up all of the values for numeric arrays.

Use the **Length** of the array to find the upper limit. The last index value is **less than** the length.

// loop through the array and print each value

for (int i = 0; i < mySuits.length; i++){

System.out.print (mySuits[i] + ", ");

}

**Enhanced Loops (a.k.a. foreach loop)**

An ‘**enhanced**’ loop is a simpler version of a **For loop** because we do not need an **index** variable. Instead we create a single **block level** variable that represents each element of the array, one element at a time. We also do not have to worry about the upper limit of the array. In this example ‘**oneSuit**’ is the block level variable. Each element of the ‘mySuits’ array is represented in the print statement each iteration through the loop.

for (String **oneSuit**:mySuits){

System.out.print (**oneSuit** + ", ");

}

**Array Class Methods:** Methods that can be applied to any array

**fill** (arrayName, value) -> good for setting all values in a array to a default value.

**fill** (arrayName, index1, index2, value) -> sets some elements to a single value.

**equals** (arrayName1, arrayName2) -> true if both arrays are the same type and have the same values.

Using the condition, (Array1 == Array2), will give a true value ONLY if the **objects** point to the same memory location.

The ‘equals’ method will be true if the **values** in the arrays are the same.

**copyOf** (arrayName, length) -> returns a copy of an array with the specified length

**copyOfRange** (arrayName, index1, index2)

**sort** (arrayName) -> in ascending order

**sort** (arrayName, index1, index2) -> in ascending order

**binarySearch** (arrayName, value) -> finds the location of the ‘value’ & returns the index

**Using the Comparable Interface for sorting user defined objects.**

Arrays of a **data type** can only be sorted if the objects of that data type **implement** the *Comparable* interface. The interface defines a single method “compareTo” which returns an *int* variable and takes an object as a single parameter.

The Comparable interface is:

Public interface Comparable{

int compareTo(Object obj);

}

The compareTo method will:

return a -1 when the current object is less than the other object,

return a 0 if they are equal, and

return a +1 if the current object is greater than the other object.

**Array Reference and Copy**

The arrays are objects. When we assign **one array** variable to **another array** variable they will point to the same underlying memory location.

String[] oldSuits = suits; 🡨 the two array variables point to the same object

To create two **separate** arrays we will use the ‘copyOf’ method.

String[] newSuits = Arrays.copyOf(suits, suits.length); 🡨 The newSuits array variable holds a different object than the suits object.

**Two Dimensional Arrays**

In Java a two dimensional array is created by making an array of array objects. There are two flavors of two-dimensional arrays: **Rectangular** and **Jagged**. The first dimension is a **row** and the second dimension is a **column**.

Like with one-dimensional arrays values can be *assigned* **one at a time:**

String[ ][ ] twoHandsIndividual = new String [2][5];

twoHandsIndividual[0][0] = "1C";

twoHandsIndividual[0][1] = "2C";

twoHandsIndividual[0][2] = "3C";

twoHandsIndividual[0][3] = "4C";

twoHandsIndividual[0][4] = "5C"; ...and so on

or *assigned* when the array is **first declared:**

String[][] twoHandsDeclared = {{"1H", "2H", "3H", "4H", "5H"},

{"1S", "2S", "3S", "4S", "5S"}};

Note that each row is enclosed in a set of **inner** curly brackets and that there are two sets of rows in the **outer** curly brackets for the entire array. This array has 2 rows and 5 columns. In this example each row represents a single hand of cards.

*Reading* a value from the arrays requires two index values.

String singleCard = twoHandsDeclared [1][4]; // “singleCard” will be “5S”.

This code will print out the first hand only.

for (int i = 0; i < twoHandsDeclared[0].length; i++){

System.out.print (twoHandsDeclared[0][i] + ", ");

}

This code will only print the two of spades if it is in the second hand.

for (int i = 0; i < twoHandsDeclared[1].length; i++){

if (twoHandsDeclared[1][i].equals("1S"))

System.out.print (twoHandsDeclared[1][i]);

}

For the **jagged** array each row can have its own array size. This *numbers* array was three rows but the number of columns in each row is not defined. We define each individual row array separately because they are of different size (5, 10, or 15 elements).

int [ ] [ ] numbers = new int [3] [ ];

numbers [0] = new int[5];

numbers [1] = new int[10];

numbers [2] = new int[15];

**What Are Arrays Used For?**

Arrays can be used to **store data**. If there are 84 majors, we do not want to create 84 variables to hold the number of student in a major (count) for each one. A one dimensional array can hold the count of students in each major. We can ‘**lookup**’ values in the array if we know the index for the array element. If the first major is accounting then the following line of code will look up the number of accounting majors, and the next line finds the count for CIS majors if they are the 12th major in the array.

count = majorsCount[0];

count = majorsCount[11];

A two dimensional array is sometimes called a ‘**lookup table’**. If the elements of the array are the prices of a widget, and one dimension is the size and the second dimension is the type of widget (basic, standard, deluxe), then given a certain size and type index values we can loop up the price.

price = widgetPrices[2][1];

If we ask a user for a specific widget size and type, we can translate the choices into index values and dynamically look up the price of a widget.

price = widgetPrices[rowIndex][columnIndex];

All elements of an array must be of the same data type so we can’t mix integer and string values, but an object can be designed to hold a mixture of data types. Thus an **array of objects** can hold different data types.

Arrays can easily be **searched**. Using a ***loop***, we can search every element of an array. An ***if statement*** inside the loop is then used to determine if the element matches a **target** value. If a major is represented by an object that holds the major name, college, and number of students then an array of major objects can be searched for all majors that are part of the college of business. If a specific major object is part of the college of business, then the true branch can accumulate the total number of business students.

for (int i = 0; i < majorsArray[].length; i++){

if (majorsArray [i].getCollegeName().equals("CBUS”))

CBUScount += majorsArray[i].getCount();;

}