Today’s Topics

• Comments and/or Questions?
• Finish arrays
• Multidimensional arrays
• Math class methods
Arrays

- A data structure of related data items of the *same type*
- They store this data in contiguous memory locations
- Each data item in an array is an *element*.
- Arrays are denoted with square brackets.
- When you want to select one element of the array, use its *index* in square brackets. *Array indices start at 0.*
- When we speak of arrays: *array* means the full array, and *array element* is one element of the array.
Arrays

• Arrays can be of any type (user-defined class types, Java API class types, or primitive types.)

• Arrays of primitive types:
  – Might use an array of integers to store test scores. (better than creating individual variables of type int for reasons which should become clear.)
  – Might use an array of doubles to store a list of temperatures.

• Arrays of Java API class types:
  – Might use an array of Strings to store a list of people’s names.
Arrays (declaring and allocating)

- Arrays, like any variables need to be declared.
- Arrays also need to have memory allocated for a specific number of elements.

Example:
```java
int testScores[]; // declares an array of ints

testScores = new int[16]; // the line above allocates enough memory for 16 ints
```
Arrays (declaring and allocating)

- Arrays can be declared and allocated in one line.
- Example:

```
// declares an array of ints & allocates enough memory for 16 ints
int testScores[] = new int[16];
```
Arrays (allocating and initializing)

• When arrays are allocated, the array elements are all initialized to the following values:
  • Arrays of numeric primitive types (byte, short, long, int, float, double, char) have their elements initialized to 0.
  • Arrays of booleans (the only other primitive type) have their elements initialized to false.
  • Arrays of any non-primitive data type, (classes in the Java API or user-defined classes) have their elements initialized to null.
Arrays (declaring, allocating and initializing)

• Arrays can be declared, allocated and initialized to user defined values all in one line of code.

• Example:

```c
int testScores[] = {
    130, 120, 111, 44, 181,
    183, 160, 165, 148, 147,
    146, 153, 139, 150, 153,
    122
};
```

// declares an array of ints & allocates enough memory for 16 ints and // initializes each element to the value listed
Arrays

• In our example of the testScores array, each element of this array is an **int**. So, we can refer to say the 10th element of the array by `testScores[9]` and this is an **int**, so we can use it anywhere an **int** can be used.

• REMEMBER: Array indices start at 0 (zero) and end at the (size of the array – 1).
  – E.g. Indices for an array of size 36, go from 0 to 35.

• Array values can be set one element at a time by supplying the index like so:
  ```
  // stores the value 130 in the 0 element of the array testScores
  testScores[0] = 130;
  // stores the value 120 in the 1 element of the array testScores
  testScores[1] = 120;
  // etc …
  ```

Arrays

• Each array knows its own length, use the array name followed by `.length` to get access to its length. In our case for the testScores example, the length is 16 (0 to 15.)

• Arrays are easily manipulated with loops.

• Let’s look at an example.
Array code example

// Code to populate (store values into the elements of) an array

int testScores[] = new int[16];
String user_input;

for (int i=0; i < testScores.length; i++)
{
    user_input = JOptionPane.showInputDialog("Enter an integer");
    testScores[i] = Integer.parseInt(user_input);
}
// Code to find the lowest element of an array of ints.

int least = testScores[0];

for (int i=0; i < testScores.length; i++)
{
    if (testScores[i] < least)
        least = testScores[i];
}

System.out.println("The lowest value in the array is: " + least);
Array code example

• Let's rewrite that program from before, but use an array to store all the values
  – Compute least, greatest, sum and average.
String peoplesNames[] = new String[10];
// The above line declares and allocates space for 10 Strings.

// The Strings then can have their values set like:

peoplesNames[0] = "Jerry Garcia";

peoplesNames[1] = "Bob Weir";

// etc.
Example of an array of Strings

• Just like you'd expect, not only String literals are allowed to be assigned, we could assign another String variable value to one of the elements of the peoplesNames array.

```java
String peoplesNames[] = new String[10];

for (int i=0; i < peoplesNames.length; i++)
{
    String user_input_string =
        JOptionPane.showInputDialog("Enter a name");
    peoplesNames[i] = user_input_string;
}
```
.length of an array VS. .length() method of String class

• Understand the difference between
  
  – .length the variable accessible for Arrays to tell us how many elements the array has
  – .length() the method in class String to find out how many characters are in the String.

• e.g.
  String csCourseNames[] = new String[20];
  // csCourseNames.length --- this is the length of the array which is 20 here.
  csCourseNames[0] = “Introduction to computer science I”;
  // csCourseNames[0].length() --- this is how many characters are in
  // the csCourseNames[0] --- which is 34
.length of an array VS. .length() method of String class

String csCourseNames[] = new String[20];
csCourseNames[0] = “Introduction to computer science I”;

System.out.println(“The number of elements in the csCourseNames array is:” +
csCourseNames.length);

System.out.println(“The number of characters in the 0 element of the array is:” +
csCourseNames[0].length());

// so what #'s will these print?
Arrays of Strings

• So, let’s write a for loop that will populate an array of Strings based on user input.

• What will the for loop’s control variable be initialized to?

• What will the condition to stop the loop be?

• What will we increment the control variable by?
Two dimensional arrays

- If, instead of a list of items, you wanted to store a grid of items, you could use a 2d array. A grid has rows and columns.

- A two dimensional array is declared with two pairs of brackets.

```java
int array2d[][] = new int[2][3];
```

// This line creates a 2-by-3 array (2 rows, 3 columns)
Two dimensional arrays

int array2d [ ] [ ] = { { 5, 7, 11 }, { 62, 3, 48 } };

to access the element at row 0, column 1, we use:

array2d [ 0 ] [ 1 ] // this holds the value 7 in our example.
Two dimensional arrays

• A two dimensional array might be useful for something like keeping track of all the students in a class and their grades on each of the homeworks.
• So, each student could be represented by a number (0 to 16, for 17 students) and each homework represented by a number (0 to 9, for 10 hw's).
• Each student could be a row in the 2d array and each homework could be a column.
• The values stored in the array elements are the grade a particular student got on a particular homework.
Two dimensional arrays

```java
int student_hw_grades[][] = new int[17][10];

// The outer loop will be for the 17 students and the inner loop will be for the 10 hw's.

for (int row = 0; row < array2d.length; row++)
{
    for (int col = 0; col < array2d[row].length; col++)
    {
        user_str = JOptionPane.showInputDialog(
            "Enter student " + row + 
            "'s homework #" + col + "grade");

        student_hw_grades[ row ][ col ] = Integer.parseInt(user_str);
    }
}

// note the use of .length in the two loop conditions
```
Multi-dimensional arrays

- Not only can we have one dimensional, and two dimensional arrays but we can have \( n \) dimensional arrays, where \( n \) is any positive integer.

- Example when \( n=3 \):

  ```java
double temperatures[][][][] = new double [12][31][24];
```

  Take a guess as to what might be stored in this array and what the indices mean.
Multi-dimensional arrays

• Unfortunately the first index must go from 0 to 11, the second from 0 to 30 and the third from 0 to 23.
• What if we wanted the index to represent exactly the month (1 to 12), day (1 to 31) and hour (0 to 23)?
• Is there anything we could do to this line?

    double temperatures[][][] = new double [12][31][24];
Math class

• Let's look at the Java API for the Math class.
• Specifically these methods:

  • abs – guess what this does.
  • cos, sin, tan
  • ceil – returns smallest whole number $\geq$ parameter.
  • pow – takes two parameters – raises first to second and returns the result.
  • random – returns random # in the range: $[0.0, 1.0)$
  • sqrt
A few more methods in the Math class

**max** (x, y)
method that returns the larger of x and y

**min** (x, y)
method that returns the smaller of x and y

There are versions of these methods that work for x and y being floats, doubles, ints and longs and return a result that is same type.
example calls to static methods in the Math class

```java
double w = 5.1, z = 10.56, a, b, c;

a = Math.max ( w, z ); // what value would a have?

a = Math. max ( z, w ); // what value would a have, now?

b = Math. min ( z, w );

c = Math. sqrt ( z );
```
random( ) method in the Math class

double randNum;

randNum = Math.random ( );
// what value might randNum have after this line of code?

// is 0.34452 a possible value?

// is 2 a possible value?

// is -14.555423 a possible value?
random( ) method in the Math class

• `random( )` returns a `double` whose value is \( \geq 0 \) and \(< 1 \), but sometimes we want a random integer

• How might we do that?
random( ) method in the Math class

• **random ( )** returns a **double** whose value is $\geq 0$ and $< 1$, but sometimes we want a random integer.

• One way to do that is to first multiply the result by some integer to get a value that isn’t necessarily between 0 and 1. Then, cast this new value to an **int** by using the **(int)** cast operator.
random( ) method in the Math class

// example:
int someRandomInt;
double someRandomDbl;

someRandomDbl = Math.random ( ) * 25;
// this will result in a value >= 0 and < 25.

someRandomInt = (int) (Math.random ( ) * 25);

// what is the range of values for someRandomInt here?
random( ) method in the Math class

```java
int randomCardValue;
int randomCardSuit;

randomCardValue = 1 + (int) (Math.random ( ) * 13);
randomCardSuit = (int) (Math.random ( ) * 4);
```

• Let’s put this code in a program and execute it.
random( ) method in the Math class

• What if I put the cast to int without using parentheses around the rest of the expression?

• e.g.

    randomCardSuit = (int) Math.random ( ) * 4;
random( ) method in the Math class

• What if I put the cast to int without using parentheses around the rest of the expression?

• e.g.
  
  randomCardSuit = (int) Math.random ( ) * 4;

• since the cast operator (int) has higher precedence than the multiplication operator *, it would be done first, which means what?
random( ) method in the Math class

- random_card_suit = (int) Math.random ( ) * 4;

- the **Math.random()** method call would return a double value and immediately this value would be cast to an int. Casting a double to an int causes the truncation of any decimal portion. Recall that the double that is returned by **Math.random()** is >= 0.0 and < 1.0

- So, what's the possible values of (int) Math.random() ?
random( ) method in the Math class

- (int) Math.random( ) would always be zero.