Arrays

Reading: Appendix A

• Declaring and Instantiating Arrays
• Accessing Array Elements
• Writing Methods.
• Aggregate Array Operations.
• Using Arrays in Classes.
• Searching Arrays.

Arrays:

• An **array** is a sequence of variables of the same data type.
• The data type can be any of Java’s **primitive types**:
  • `int`, `short`, `byte`, `long`, `float`, `double`, `boolean`, `char`
• The data type can be any **class**:
  • `String`, `SolidBoxes`, etc.
• Each variable in the array is an **element**.
• An **index** specifies the position of each element in the array.
• Useful for many applications:
  • Collecting statistics.
  • Representing the state of a game.
  • Etc.
Declaring and Instantiating Arrays:

- Arrays are objects.

- Creating an array requires two steps:
  1. Declaring the reference to the array
  2. Instantiating the array.

- To declare a reference to the array:
  ```
  datatype [ ] arrayName;
  ```

- To instantiate an array:
  ```
  arrayName = new datatype[ size ];
  ```

- `size` is an `int` and is the number of elements that will be in the array.

Declaring and Instantiating Arrays (continued):

- Examples:
  - Declaring and instantiating arrays of primitive types:
    ```
    double [ ] dailyTemps;       // elements are doubles
    dailyTemps = new double[ 365 ];  // 365 elements

    boolean [ ] answers;         // elements are booleans: true, false
    answers = new boolean[ 20 ];  // 20 elements

    int [ ] cs127A, cs252;       // two arrays, each containing integers
    cs127A = new int[ 310 ];     // 310 elements for cs127A
    cs252 = new int[ 108 ];      // 108 elements for cs252
    ```

  - Declaring and instantiating arrays of objects:
    ```
    String [ ] cdTracks;         // each element is a String
    cdTracks = new String[ 15 ]; // 15 elements to hold song names

    BaseballStats[] myTeam;     // each element is one player's statistics
    myTeam = new BaseballStats[ 25 ]; // 25 players on my team
    ```
Declaring and Instantiating Arrays (continued):

- The declaration and instantiation can be done in the same step.

```java
double [] dailyTemps = new double[ 365 ];
... can now use the dailyTemps array in my code ...
dailyTemps = new double[ 173 ];
... can now use the new size of dailyTemps array in my code ...
```

```java
int numberOfQuestions = 30;
boolean [] answers = new boolean[ numberOfQuestions ];
```

```java
int [] cs127 = new int[ 240 ], cs252 = new int[ 75 ];
String [] studentNames = new String[ 42 ];
```

```java
int numPlayers = 25;
int numTeams = 10;
BaseballStats [] myTeam = new BaseballStats[ numPlayers * numTeams ];
```

---

Declaring and Instantiating Arrays (continued):

- When an array is instantiated, the elements are assigned default values as follows:

<table>
<thead>
<tr>
<th>Array data type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte, short, int, long</td>
<td>0</td>
</tr>
<tr>
<td>float, double</td>
<td>0</td>
</tr>
<tr>
<td>char</td>
<td>nul character</td>
</tr>
<tr>
<td>boolean</td>
<td>false</td>
</tr>
<tr>
<td>Any object reference (for example, a String)</td>
<td>null</td>
</tr>
</tbody>
</table>
Declaring and Instantiating Arrays (continued):

- The declaration of an array creates an area of memory that holds the elements of the array.
- This area of memory has one name, the name of the array.

```java
double [] zapNums = new double[ 5 ];
```

```java
int numberOfQuestions = 4;
boolean [] answers = new boolean[ numberOfQuestions ];
```

Assigning initial values to arrays:

- Arrays can be instantiated by specifying a list of initial values.
- Syntax:

  ```java
datatype [] arrayName = { value0, value1, ...};
  ```

- where `valueN` is an expression evaluating to the data type of the array and is assigned to element at index `N`.

- Examples:

  - Create an array of integers. The array will have 11 elements, since there are 11 values supplied:
    ```java
    int magic = 13;
    int [] oddNumbers = { 1, 3, 5, 7, 9, magic, magic + 2, 17, 19, 21, magic + 10 };
    System.out.println( oddNumbers[7] );   // prints 17
    System.out.println( oddNumbers[4] );   // prints 9
    System.out.println( oddNumbers[magic - 3] ); // prints 23
    System.out.println( oddNumbers[5] - magic ); // prints 0
    ```

- Notes:
  - The `[]`’s are empty; do not put in the size of the array.
  - The `new` keyword is **not** used.
  - The Java compiler will count the number of elements inside the `{ }’s and use that as the size of the array.
Declaring and Instantiating Arrays (continued):

Assigning initial values to arrays (continued):

• Another Example:
  
  • Create an array of String’s. The array will have 3 elements, since there are 3 values supplied:
    
    ```
    String middleName = "Thomas";
    String [] myNames = { "Patrick", middleName, "Homer" };
    int [] oddNumbers = { 1, 3, 5, 7, 9, magic, magic + 2, 17, 19, 21, magic + 10 };
    ```
  
  • You can use this technique only when the array is first declared.
    
  • For example, the first line below is correct. The second is not!
    
    ```
    double[] dailyMiles = { 175.3, 278.9, 0.0, 0.0, 0.0};  // correct
    dailyMiles = { 170.3, 278.9, 283.2, 158.0, 433.3};     // WRONG!!
    ```
  
  • The compiler will complain about the second line:
    
    ```
    zap.java:17: illegal start of expression
    dailyMiles = { 170.3, 278.9, 283.2, 158.0, 433.3};
    ^
    ```

• int [] wubble = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};

Accessing Array Elements:

• To access an element of an array, use:
  
  ```
  arrayName[exp]
  ```
  
  • where `exp` is evaluates to an `int` that is \( >= 0 \).
  
  • `exp` is the element’s `index`; it’s position within the array.
  
  • The index of the first element in an array is \( 0 \).
  
  • Each array has a read-only integer instance variable named `length`.
  
  • `length` holds the number of elements in the array.
  
  • Examples:
    
    ```
    int [] numbers = { 34, 42, 76, 98, -109, 10 };
    System.out.println( numbers[0] + " is the element at position 0");
    System.out.println("The array numbers has " + numbers.length + " elements");

    String myName = new String("Patrick Homer");
    System.out.println("My name has " + myName.length() + " characters.");
    ```
  
  • Note the difference between the length `instance variable`
    
    of an array and the length `method` of String.
Accessing Array Elements (continued):

- To access other elements, use either an `int`, or an expression that evaluates to an `int`:

  ```java
  int[] numbers = {34, 42, 76, 98, -109, 10};
  System.out.println("The element at position three is " + numbers[3]);
  int sample = 1;
  System.out.println("numbers[" + sample + "] is " + numbers[sample]);
  System.out.println("numbers[" + sample*2 + "] is " + numbers[sample * 2]);
  ```

- How to access the last element in the array?
  - The elements in the array are numbered starting at 0.
  - The `length` instance variable will tell us the number of elements.

  ```java
  int lastElement;
  lastElement = numbers.length - 1;
  System.out.println("last element of numbers is " + numbers[lastElement]);
  System.out.println("last element of numbers is " + numbers[numbers.length]);
  System.out.println("last element of numbers is " + numbers[-1]);
  System.out.println("last element of numbers is " + numbers[xray]);
  ```

Accessing Array Elements (continued):

- Trying to access an element at a position < 0, or at a position >= `arrayName.length` will generate an error:

  ```java
  System.out.println("Element at location -1 is " + numbers[-1]);
  ```

  Gives the following error at runtime:

  ```java
  Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -1
  at Sample.main(Sample.java:16)
  ```

- Trying to execute:

  ```java
  System.out.println("Element at location length is " + numbers[numbers.length]);
  ```

  Gives the following error at runtime:

  ```java
  Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
  at Sample.main(Sample.java:18)
  ```
Array Operations:

- Since the elements are indexed starting at 0 and going to `arrayName.length - 1`, we can print all the elements of an array using a for loop:

  ```java
  public class PrintArray
  {
    public static void main(String[] args)
    {
      double [] miles = {19.5, 16.7, 22.1, 4.5, 7.5, 10.5, 16.0, 42.0};
      int i;
      // other code...
      System.out.println("Element "+i+" is "+miles[i]);
    }
  }
  ``

- Gives the following output:

  Element 0 is 19.5
  Element 1 is 16.7
  Element 2 is 22.1
  Element 3 is 4.5
  Element 4 is 7.5
  Element 5 is 10.5
  Element 6 is 16.0
  Element 7 is 42.0

Array Operations (continued):

- The loop on the previous slide is typical of loops that access elements of an array:

  ```java
  for ( int i = 0; i < arrayName.length; i++)
  { // other code...
    System.out.println("Element "+i+" is "+miles[i]);
  }
  ``

- Start with `i = 0`, since the first element in any array is at index 0.
- Loop continues as long as `i < arrayName.length` is true.
  - Stop when `i` reaches `arrayName.length` (do not execute the body of the loop for this value of `i`).
  - Loop update is `i++` so we process each element in order.
- Variations:
  - Can print every other element in the array: Use `i += 2` to update the loop index.
  - How to print the array backwards?
    - What is the initial value of `i`? (Where is the last position of the array?)
      ```java
      for ( i = miles.length - 1;
      ```
    - How do we make `i` count backwards?
      ```java
      for ( i = miles.length - 1; ... ; i-- )
      ```
    - How do we stop the loop?
      ```java
      for ( i = miles.length - 1; i >= 0; i-- )
      ```
Array Operations (continued):

Reading data into an array:

- We can read data from the user to put into an array:

```java
import java.util.Scanner;
public class ReadDoubles {
    public static void main(String[] args) {
        Scanner inputScan = new Scanner(System.in);
        int i;
        double[] numbers = new double[10];
        for (i = 0; i < numbers.length; i++) {
            System.out.print("Enter a number: ");
            numbers[i] = inputScan.nextDouble();
        }
        System.out.println();
        for (i = 0; i < numbers.length; i++) {
            System.out.print("numbers[" + i + "] is ");
            System.out.println(numbers[i]);
        }
    }
}
```

Array Operations (continued):

Summing elements of an array:

- Once we have the values in an array, we can use loop(s) to perform calculations. For example, we can extend the previous example to find the average of the numbers the user enters:

```java
double sum;
double average;
double[] numbers = new double[<some value goes here>];
... read in the numbers from the user ...

sum = 0.0;
for (i = 0; i < numbers.length; i++) {
    sum = sum + numbers[i];
}
if (numbers.length != 0)
    average = sum / numbers.length;
else
    average = 0.0;
System.out.println("average is "+average);
```
Array Operations (continued):

Copying arrays:

• Suppose we want to copy the elements of one array to another array. We could try this code:

```java
double[] numbersBackup = new double[10];
namesBackup = numbers;
names[3] = 927.37;
System.out.println( numbersBackup[3] );
```

• This code will compile, and it will run without producing error messages.
• But, it is wrong!!!!
• We get two variables that both refer to the same array.
  • Two aliases for the same memory location.
• We wanted two arrays with the same contents.

Array Operations (continued):

Copying arrays (continued):

• Suppose we want to copy the elements of one array to another array. We could try this code:

```java
double[] numbersBackup = new double[ numbers.length ];
for ( int i = 0; i < numbers.length; i++)
  numbersBackup[i] = numbers[i];
```

• This code makes a second array and puts a copy of each element from the first array into the second.
**Array Operations (continued):**

**Copying arrays (continued):**

- Now we can change the contents of one array without changing the other:

```java
for (int i = 5; i < numbers.length; i++)
    numbers[i] = numbers[i] + 2;
```

---

**Array Operations (continued):**

**Comparing Arrays for Equality:**

```java
if (numbers == numbersBackup) // does NOT work
```

- To compare whether the elements of two arrays are equal:
  1. Determine if both arrays have the same length.
  2. Compare each element in the first array with the corresponding element in the second array.

- To do this, use a flag variable and a for loop:
  - A flag variable:
    - A boolean.
  - Two ways to use it:
    - Set the flag to true, then test to determine if the flag should be false. OR
    - Set the flag to false, then test to determine if the flag should be true.
  - For this example, we will set the flag to true and then test to determine if it should be false.
Array Operations (continued):

Comparing Arrays for Equality (continued):

```java
public class CompareArrays
{
    public static void main(String[] args)
    {
        int[] alpha = {0, 1, 2, 43, 48, 59, 60, 70, 88, 90, 1000};
        int[] bravo = {0, 1, 2, 43, 48, 59, 60, 70, 88, 90, 1000};

        boolean equalFlag = true;   // Set the flag to true
        // Test if the flag should be false
        if (alpha.length != bravo.length)
            equalFlag = false;
        else
            for (int i = 0; i < alpha.length; i++)
                if (alpha[i] != bravo[i])
                    equalFlag = false;

        // Print the result
        if (equalFlag)
            System.out.println("alpha and bravo are equal");
        else
            System.out.println("alpha and bravo are NOT equal");
    }
} // end of class CompareArrays
```

Array Operations (continued):

Finding Maximum/Minimum Values:

- General idea:
  - Use a loop to examine each value in the array.
    - Compare each value with the largest value found so far.
  - But, what about the first element in the array? What is it compared to?
- **Key point**: Assume the first value is the largest.
  - Then, the loop tests that assumption, changing the largest as needed.
Array Operations (continued):
Finding Maximum/Minimum Values (continued):

```java
import java.util.Scanner;
public class LargestNumber{
    public static void main(String[] args){
        Scanner inputScan = new Scanner(System.in);
        int[] zap = new int[10];
        int i;
        int largest;
        for (i = 0; i < zap.length; i++) {
            System.out.print("Enter an integer: ");
            zap[i] = inputScan.nextInt();
        }
        largest = zap[0]; // assume zap[0] is the largest
        // test other values in zap to see if any are larger
        for (i = 1; i < zap.length; i++)
            if ( zap[i] > largest )
                largest = zap[i];
        System.out.println("The largest integer is " + largest);
    } // end of method main
} // end of class LargestNumber
```

What would you change to find the smallest instead of the largest?

Array Operations (continued):
Finding Maximum/Minimum Values (continued):

• What if the array is an array of String’s?

```java
public class CompareStrings{
    public static void main(String[] args){
        String[] words = {"hello", "c3p0", "out of here", "outward", "inward",
                          "onward", "r2d2", "water", "Zombie", "wombat");
        int i;
        String largest;
        largest = words[0];
        for (i = 1; i < words.length; i++)
            if ( words[i].compareTo(largest) > 0 )
                largest = words[i];
        System.out.println("largest word is " + largest);
    } // end of method main
} // end of class CompareStrings
```

How to find the smallest?
Using Arrays in Classes:

- In a user-defined class, an array can be:
  - An instance variable.
  - A parameter to a method.
  - A return value from a method.
  - A local variable in a method.

Using Arrays in Classes (continued):

Grades Class Example:

- Consider a class that maintains a set of grades for an assignment or exam.
- We need an instance variable that is an array to hold the grades:

```java
private double[] grades;

public void setGrades( double[] newGrades ) {
    int i;
    for (i = 0; i < newGrades.length; i++)
        grades[i] = newGrades[i];
} // setGrades method
```

- Every time that `setGrades` is called, it
  - creates a new array of `double`'s, and
  - copies in the new values.

```java
grades = newGrades;
```
Using Arrays in Classes (continued):
Grades Class Example (continued):

- The constructor will need grades to use as the initial set of values.
  - Will pass the Constructor an array of double's; for example, GradesClient.java might do in main:
    ```java
double[] someGrades = { 89.5, 95.5, 100.0, 64.5, 72.5, 73. };
Grades examScores, moreScores;
examScores = new Grades( someGrades );
moreScores = new Grades( someGrades );
```

- The constructor for Grades.java:
  ```java
  public Grades( double [] inputGrades ) {
    setGrades( inputGrades );
  } // Constructor with grades
  ```

- There are now two copies of the grades:
  - One in the client (inside main).
  - One inside the object.

```java
double[] anotherGrades = { 42.6, -62.7, 99.0 };
examScores.setGrades( anotherGrades );
```

- Not visible to the outside.

```java
private
 grades

89.5
95.5
100.0
64.5
72.5
73.0
```

- Not visible from the outside.
Using Arrays in Classes (continued):
Grades Class Example (continued):

• Want an accessor, get, method for the grades.
  • This method will need to return an array of double’s.
  • Should not return a reference to the internal private instance variable.
    • Use a local variable inside the get method.
    • Create an array for this local variable.
    • Copy contents of the instance array into this local array.

```java
public double[] getGrades() {
    double[] tempGrades = new double[grades.length];
    int i;
    for (i = 0; i < grades.length; i++)
        tempGrades[i] = grades[i];
    return tempGrades;
} // getGrades method
```

From GradesClient.java:
```java
double[] someGrades = { <values here> };
Grades examScores;

examScores = new Grades( someGrades );
double[] answers;
answers = examScores.getGrades();
answers[1] = 100.0;
for (int i = 0; i < answers.length; i++)
    System.out.println( answers[i] );
```

Using Arrays in Classes (continued):
Grades Class Example (continued):

• What would a default constructor do? (A constructor that takes no arguments.)
  • What should the array it creates look like?
  • Here is one possibility:

```java
public Grades() {
    grades = new double[0];
} // default Constructor
```
Using Arrays in Classes (continued):

Grades Class Example (continued):

• If the object was created by calling the default constructor, what does the following code do?

  Grades noScores = new Grades();
  double[] anyScores = noScores.getGrades();
  System.out.println("Array size is " + anyScores.length);
  for (int i = 0; i < anyScores.length; i++)
    System.out.println( anyScores[i] );

Partially-filled Arrays:

• Arrays are not always full.

• Consider a shopping list class. We need an array of String’s to hold the items in the shopping list.
  • But, we do not buy the same number of items each time we go to the store.

• The constructor(s) for the class need to know the capacity of the array.

  • Use two local int’s:
    • The capacity of the array, how large it was declared.
    • The number of items currently in the array. Will be a value between 0 and the capacity.

public class ShoppingList
{
  private String[] list;
  private int capacity;
  private int itemCount;
Partially-filled Arrays (continued):

• Constructor(s) need to specify a size for the array, the capacity.
  
• And, set the count of items in the list to zero.

  // default constructor: 15 items for express lane use
  public ShoppingList() {
    list = new String[15];
    capacity = 15;
    itemCount = 0;
  } // end of constructor

  public ShoppingList(int numItems) {
    list = new String[numItems];
    capacity = numItems;
    itemCount = 0;
  } // end of constructor

• When we add items to the array, we also need to increment the count of the items.

  public void addItem(String anItem) {
    list[itemCount] = new String(anItem);
    itemCount++;
  } // end of method addItem

  • Note the order:
    • Add element to the array.
    • Increment the count of items.

• But, what if the array is already full?
  
• The code above will produce a run-time error!!
  
• Need to determine if there is, or is not, any room left in the array.

  public boolean isFull() {
    boolean answer;
    if ( itemCount >= capacity )
      answer = true;
    else
      answer = false;
    return answer;
  } // end of method isFull
Partially-filled Arrays (continued):

- Now, when we add items to the array, we can check to make sure there is room!

```java
public boolean addItem(String anItem) {
    if ( ! isFull() ) {
        list[itemCount] = new String(anItem);
        itemCount++;
        return true; // added new item to list
    }
    else
        return false; // no room for new item
} // end of method addItem
```

- `addItem()` returns a `boolean` to indicate the item was, or was not, added to the array.

- The `itemCount` instance variable becomes the limit on loops. For example, in the `toString()` method:

```java
public String toString() {
    String result = new String();
    // String result = "";
    int i;
    for (i = 0; i < itemCount; i++)
        result = result + list[i] + "\n";
    return result;
} // end of method toString
```
Partially-filled Arrays (continued):

- ShoppingList.java
  ```java
  public class ShoppingList {
    private String[] list; // max number of elements in the list
    private int capacity;  // current number of items in the list
    private int itemCount; // default constructor: 15 items for express lane use
    public ShoppingList() {
      list = new String[15];
      capacity = 15;
      itemCount = 0;
    } // end of constructor
    public ShoppingList(int numItems) {
      list = new String[numItems];
      capacity = numItems;
      itemCount = 0;
    } // end of constructor
    public boolean isFull() {
      if (itemCount == capacity)
        return true;
      else
        return false;
    } // end of method isFull
    public boolean addItem(String anItem) {
      if (!isFull()) {
        list[itemCount] = new String(anItem);
        itemCount++;
        return true; // added new item to list
      }
      else
        return false;  // no room for new item
    } // end of method addItem
    public String toString() {
      String result = new String();
      int i;
      for (i = 0; i < itemCount; i++)
        result = result + list[i] + "\n";
      return result;
    } // end of method toString
  } // end of class ShoppingList
  ```
Partially-filled Arrays (continued):

- How to use the `ShoppingList` class, two techniques:
  
  - If we know we won’t use more than capacity items:
    ```java
    ShoppingList myList;
    myList = new ShoppingList();
    myList.addItem("rice");
    myList.addItem("tortilla chips");
    myList.addItem("salsa");
    myList.addItem("lemonade");
    System.out.println("My shopping list:");
    System.out.println( myList.toString() );
    ```

  We ignore the boolean returned by the `addItem` method.

  What happens if we try to add more than 15 items?

- Use the return value from `addItem()` to halt the loop if the user tries to enter too many items:
  ```java
  ShoppingList myList;
  String item;
  boolean okay = false;
  myList = new ShoppingList(10);
  do {
    System.out.print("Enter an item or 'stop': ");
    item = inputScan.next();
    if ( ! item.equalsIgnoreCase("stop") )
      okay = myList.addItem(item);
  } while ( ! item.equalsIgnoreCase("stop") && ! myList.isFull() );
  System.out.println();
  System.out.println("My shopping list:");
  System.out.println( myList.toString() );
  ```

Stop if the user says to stop
Stop if the list fills up.
Grades Source Code:

```java
public class Grades {
    private double[] grades;

    public Grades() {
        grades = new double[0];
    } // default Constructor

    public Grades(double[] inputGrades) {
        setGrades(inputGrades);
    } // Constructor with grades

    public Grades(String fileName) {
        Scanner.
    }

    public void setGrades(double[] newGrades) {
        grades = new double[newGrades.length];
        int i;

        for (i = 0; i < newGrades.length; i++)
            grades[i] = newGrades[i];
    } // setGrades method

    public double[] getGrades() {
        double[] tempGrades = new double[grades.length];
        int i;

        for (i = 0; i < grades.length; i++)
            tempGrades[i] = grades[i];

        return tempGrades;
    } // getGrades method

    public String toString() {
        String temp = new String("elements: ");
        temp = "elements: ";
        temp = new String();
        temp = "";
        int i;

        for (i = 0; i < grades.length - 1; i++)
            temp = temp + grades[i] + ", ";
        temp = temp + grades[grades.length - 1];
        return temp;
    } // toString method
```
Grades Source Code (continued):

    public boolean equals(Grades obj) {
        boolean result;
        int i;
        double [] otherGrades;

        otherGrades = obj.getGrades();

        result = true;
        if ( otherGrades.length != grades.length )
            result = false;
        else {
            for ( i = 0; i < grades.length; i++ )
                if ( Math.abs( grades[i] - otherGrades[i] ) > .001 )
                    result = false;
        }
        return result;
    } // equals method

    public double highestGrade() {
        int i;
        double highest = grades[0];
        for (i = 1; i < grades.length; i++)
            if ( grades[i] > highest )
                highest = grades[i];
        return highest;
    } // highestGrade method

    public double lowestGrade() {
        int i;
        double lowest = grades[0];
        for (i = 1; i < grades.length; i++)
            if ( grades[i] < lowest )
                lowest = grades[i];
        return lowest;
    } // lowestGrade method

    public double average() {
        int i;
        double average, sum;

        sum = 0.0;
        for ( i = 0; i < grades.length; i++ )
            sum = sum + grades[i];

        if ( grades.length == 0 )
            return 0.0;
        else
            return sum / grades.length;
    } // average method

} // end of class Grades
Grades Client Source Code:

```java
public class GradesClient
{
    public static void main(String[] args)
    {
        double[] someGrades = { 89.5, 95.5, 100.0, 64.5, 72.5, 73.0 }; 
        Grades examScores;

        examScores = new Grades( someGrades );
        System.out.println("examScores contains:");
        System.out.println(examScores.toString());
        System.out.println();
        System.out.println("Highest grade = "+ examScores.highestGrade());
        System.out.println();
        System.out.println("Lowest grade = "+ examScores.lowestGrade());
        System.out.println();
        System.out.println("Average = "+ examScores.average());

        double[] answers;
        answers = examScores.getGrades();
        for (int i = 0; i < answers.length; i++)
            System.out.println("answers["+i+"] is "+answers[i]);

        Grades noScores = new Grades();
        System.out.println("Average for noScores: "+noScores.average());
        double[] anyScores = noScores.getGrades();
        System.out.println("Grades that are in noScores:");
        for (int i = 0; i < anyScores.length; i++)
            System.out.println( anyScores[i] );
    }
}
```
Multi-Dimensional Arrays

Reading: Appendix A

- Declaring and Instantiating 2-D Arrays
- Aggregate Two-dimensional Array Operations.
- Other Multi-dimensional Arrays.

Two-Dimensional Arrays:
- An array that organizes data into rows and columns.
- Example:
  - Grades for a course:
    - Each row represents one student.
    - Each column represents one assignment/exam/homework/etc:

<table>
<thead>
<tr>
<th></th>
<th>Hw1</th>
<th>Hw2</th>
<th>Hw3</th>
<th>Exam</th>
<th>Exam</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dave</td>
<td>98</td>
<td>95</td>
<td>87.5</td>
<td>92.5</td>
<td>83</td>
<td>91.2</td>
</tr>
<tr>
<td>Mary</td>
<td>100</td>
<td>97.5</td>
<td>92</td>
<td>95.5</td>
<td>98.5</td>
<td>96.7</td>
</tr>
<tr>
<td>Pete</td>
<td>99</td>
<td>85</td>
<td>87.5</td>
<td>82.5</td>
<td>89</td>
<td>88.6</td>
</tr>
</tbody>
</table>
Declaring and Instantiating 2-D Arrays:

- Creating an array requires two steps (for both 1-D and 2-D arrays):
  1. Declaring the reference to the array
  2. Instantiating the array.

- To declare a reference to a 2-D array:
  ```
  datatype [][] arrayName;
  ```

- To instantiate an array:
  ```
  arrayName = new datatype[ rowSize ] [ colSize ];
  ```
  - `rowSize` is an `int` and is the number of rows that will be in the array.
  - `colSize` is an `int` and is the number of columns that will be in the array.

Declaring and Instantiating 2-D Arrays (continued):

- Examples:
  - Declaring and instantiating 2-D arrays of primitive types:
    ```
    double [][] grades;              // elements are doubles representing grades
    grades = new double[ 320 ][ 10 ]; // 320 rows and 10 columns: 320 students, 10 grades each
    ```
  
  - Declaring and instantiating 2-D arrays of objects:
    ```
    String [][] studentNames;       // each element is the name of a student
    studentNames = new String[25][13]; // 25 rows, 13 columns: 1 column per section
    ```
    ```
    String [][] monthlySchedule;    // each element is one daily activity
    monthlySchedule = new String[6][7]; // 6 rows, one row for each week
    ```
Declaring and Instantiating 2-D Arrays (continued):

- The declaration and instantiation can be done in the same step.

```java
// 320 rows and 10 columns: 320 students, 10 grades each
double[][] grades = new double[320][10];

// each element is the name of a student, 1 column per section
String[][] studentNames = new String[25][13];

// each element represents one day in the month
String[][] monthlySchedule = new String[6][7];
```

Declaring and Instantiating 2-D Arrays (continued):

- When an array is instantiated, the elements are assigned default values as follows:

<table>
<thead>
<tr>
<th>Array data type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte, short, int, long</td>
<td>0</td>
</tr>
<tr>
<td>float, double</td>
<td>0</td>
</tr>
<tr>
<td>char</td>
<td>nul char</td>
</tr>
<tr>
<td>boolean</td>
<td>false</td>
</tr>
<tr>
<td>Any object reference (for example, a String)</td>
<td>null</td>
</tr>
</tbody>
</table>
Declaring and Instantiating 2-D Arrays (continued):

- The declaration of an array creates an area of memory that holds the elements of the array.
- This area of memory has one name, the name of the array.

```java
double[][] zapNums = new double[3][5];
```

- The declaration creates 3 arrays, each containing 5 doubles.
- The array has one name, `zapNums` in this case.

Accessing 2-D Array Elements:

- Can access individual elements of the array:
Accessing 2-D Array Elements (continued):

- Loops are the best way to print the contents of any array.
- For a 2-D array, you most often need two nested loops.
- Example:

```java
for (int row = 0; row < 3; row++) {
    for (int col = 0; col < 5; col++)
        System.out.print( zapNums[row][col] + " ");
    System.out.println();
}
```

Accessing 2-D Array Elements (continued):

- The size of an array:
  - It is best to have code that automatically knows how many rows and columns are present.
  - With 1-D arrays, we did this as: `arrayName.length`
  - For 2-D arrays, we need both the number of rows and the number of columns!

- For the number of rows, use: `arrayName.length`

- For the number of columns, first specify which row, then ask for the length: `arrayName[2].length`
Accessing 2-D Array Elements (continued):

- The printing example then becomes:

```java
for (int row = 0; row < zapNums.length; row++ ) {
    for (int col = 0; col < zapNums[row].length; col++ )
        System.out.print( zapNums[row][col] + " ");
    System.out.println();
}
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>-6.2</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>42.0</td>
<td>0.0</td>
<td></td>
<td>0.0</td>
<td>99.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>87.5</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>
```

Tells us how many rows in the array.

Tells us how many columns in the current row.

- We can initialize values in a 2-D array by specifying a list of initial values.
  - Example:
    ```java
double [][] grades = { { 91.0, 92, 93, 99.0, 85.5 },  
                        { 87.5, 100, 92.5, 75.0, 62.0 },  
                        { 90,  69,  73,  82.5, 90.5 } };
```
  - A comma separates rows.
  - No comma after last row.
  - Each row gets its own set of { }’s.
  - Outer set of { }’s define the beginning and the end of the array’s values.

Neatness counts!
Line things up when you enter the values.
**Aggregate Operations on 2-D Arrays:**

Computing averages:

- Want to compute the average for each row. For a grade book, the average is usually stored as the last column:

  ```
  double [][] grades = { { 91.0, 92, 93, 99.0, 85.5, 0 },
                        { 87.5, 98, 92.5, 75.0, 62.0, 0 },
                        { 90, 69, 73, 82.5, 90.5, 0 } };
  ```

- To print the grades without the average column:

  ```
  System.out.println("The grades without the averages:");

  for ( row = 0; row < grades.length; row++ ) {
    for ( col = 0; col < grades[row].length - 1; col++)
      System.out.print( grades[row][col] + " ");
    System.out.println();
  }
  ```

  ```
  91.0 92.0 93.0 99.0 85.5
  87.5 98.0 92.5 75.0 62.0
  90.0 69.0 73.0 82.5 90.5
  ```

**Aggregate Operations on 2-D Arrays** (continued):

Computing averages (continued):

- Nested loops to compute the average.
  - The outer loop takes us through each row:
    ```
    for ( row = 0; row < grades.length; row++ )
    ```
  - The inner loop computes the sum of the grades for one row:
    ```
    for ( col = 0; col < grades[row].length - 1; col++)
    ```
  - To compute the average, we need to divide by the number of grades.
  - How many grades are there?
Aggregate Operations on 2-D Arrays (continued):

Computing averages (continued):

```java
public class Grades3
{
    public static void main(String[] args)
    {
        double [][] grades = { { 91.0, 92, 93, 99.0, 85.5, 0 },
            { 87.5, 100, 92.5, 75.0, 62.0, 0 },
            { 90, 69, 73, 82.5, 90.5, 0 } );

        int row, col;

        // Last column is for the average. Here's how to print
        // grades without printing the average column.
        System.out.println("The grades without the averages:");
        for ( row = 0; row < grades.length; row++ ) {
            for ( col = 0; col < grades[row].length - 1; col++)
                System.out.print( grades[row][col] + " ");
            System.out.println();
        }
        System.out.println();  // print a blank line

        // Compute the average:
        double sum;
        for ( row = 0; row < grades.length; row++ ) {
            sum = 0.0;
            for ( col = 0; col < grades[row].length - 1; col++)
                sum = sum + grades[row][col];
        grades[row][grades[row].length - 1] = sum / (grades[row].length - 1);
        }

        // Print grades again with the average
        System.out.println("The grades and the averages:");
        for ( row = 0; row < grades.length; row++ ) {
            for ( col = 0; col < grades[row].length; col++)
                System.out.print( grades[row][col] + " ");
        System.out.println();
        }
    } // end of method main
} // end of class Grades3
```

Aggregate Operations on 2-D Arrays (continued):

Computing averages (continued):

// Compute the average:
double sum;
for ( row = 0; row < grades.length; row++ ) {
    sum = 0.0;
    for ( col = 0; col < grades[row].length - 1; col++)
        sum = sum + grades[row][col];
    grades[row][grades[row].length - 1] = sum / (grades[row].length - 1);
}

// Print grades again with the average
System.out.println("The grades and the averages:");
for ( row = 0; row < grades.length; row++ ) {
    for ( col = 0; col < grades[row].length; col++)
        System.out.print( grades[row][col] + " ");
    System.out.println();
} // end of method main
} // end of class Grades3
### Accessing 2-D Array Elements (continued):

- Another way to initialize an array: Have the user type in the values.
  - Want to enter grades into the array.
  - Want to print the grades.

```java
import java.util.Scanner;

public class Grades1 {
    public static void main(String[] args) {
        Scanner inputScan = new Scanner(System.in);
        double[][] grades = new double[3][5];
        int row, col;
        for (row = 0; row < grades.length; row++) {
            for (col = 0; col < grades[row].length; col++) {
                System.out.print("Enter a grade for row "+ row + " col "+ col + ": ");
                grades[row][col] = inputScan.nextDouble();
            }
        }
        System.out.println(); // print a blank line
        System.out.println("The grades are: ");
        for (row = 0; row < grades.length; row++) {
            for (col = 0; col < grades[row].length; col++) {
                System.out.print(grades[row][col] + " ");
            }
            System.out.println();
        }
    } // end of method main
} // end of class Grades1
```
Aggregate Operations on 2-D Arrays (continued):

Computing average for each column:

- Nested loops to compute the average. But, this time:
  - The outer loop takes us through each column:
    ```java
    for (col = 0; col < grades[0].length; col++)
    ```
  - The inner loop computes the sum of the grades for one column:
    ```java
    System.out.println("The grades:");
    for (row = 0; row < grades.length; row++) {
        for (col = 0; col < grades[row].length; col++)
            System.out.print(grades[row][col] + " ");
        System.out.println();
    }
    System.out.println(); // print a blank line
    ```

- To compute the average, we need to divide by the number of grades.
- How many grades are there?
Aggregate Operations on 2-D Arrays (continued):

Computing average for each column (continued):

```java
// Compute the average for each column
double sum;
for (col = 0; col < grades[0].length; col++) {
    sum = 0.0;
    for (row = 0; row < grades.length; row++)
        sum = sum + grades[row][col];
    System.out.print("Average for column "+col+" is ");
    System.out.println(sum / grades.length);
}
} // end of method main
} // end of class Grades4
```

### Uneven rows:

- Consider the following declaration:
  ```java
double [][] grades = { { 91.0, 92, 93, 85.5 },
                      { 87.5, 100, 92.5 },
                      { 90, 69, 73, 82.5, 90.5 } };  
  ```

- This does compile(!!)
- It is **not** a requirement that each row have the same length.

```
grades
+----------------+
| 91  92  93  85.5 |
| 87.5 100  92.5 |
| 90  69  73  82.5  90.5 |
+----------------+
```
Aggregate Operations on 2-D Arrays (continued):

Uneven rows:

- Consider the following declaration:
  ```java
double [][] grades = { { 91.0, 92, 93, 85.5 },
                      { 87.5, 100, 92.5 },
                      { 90, 69, 73, 82.5, 90.5 } };
  ```

- How to print the elements of the array?
  - Use the length of each row to terminate the inner-loop:
    ```java
    for ( row = 0; row < grades.length; row++ ) {
        for ( col = 0; col < grades[row].length; col++)
            System.out.print( grades[row][col] + " ");
        System.out.println();
    }
    ```

Aggregate Operations on 2-D Arrays (continued):

Uneven rows (continued):

- How to compute the average from each column?
  ```java
  double [][] grades = { { 91.0, 92, 93, 85.5 },
                      { 87.5, 100, 92.5 },
                      { 90, 69, 73, 82.5, 90.5 } };
  ```

- Use the length of each row to check:
  ```java
  double [] [] grades = { { 91.0, 92, 93, 85.5 },
                         { 87.5, 100, 92.5 },
                         { 90, 69, 73, 82.5, 90.5 } };
  ```

  ```java
  for ( col = 0; col < ________________________________; col++ ) {
      sum = 0.0;
      count = 0;
      for ( row = 0; row < __________________________; row++ ) {
          if ( _______________________________________ ) {
              sum = sum + grades[row][col];
              count++;
          }
      }
      average = sum / count;
      System.out.println("Average for column " + col + " = " + average);
  }
  ```
Aggregate Operations on 2-D Arrays (continued):

• Uneven rows example code:

```java
class Grades5 {
    public static void main(String[] args) {
        double[][] grades = {{91.0, 92, 93, 85.5},
                             {87.5, 100, 92.5},
                             {90, 69, 73, 82.5, 90.5}};
        int row, col;
        System.out.println("The grades are:");
        for (row = 0; row < grades.length; row++) {
            for (col = 0; col < grades[row].length; col++)
                System.out.print(grades[row][col] + " ");
            System.out.println();
        }
        System.out.println();
        double sum;
        int count;
        double average;
        for (col = 0; col < 5; col++) {
            sum = 0.0;
            count = 0;
            for (row = 0; row < grades.length; row++)
                if (col < grades[row].length)
                    sum = sum + grades[row][col];
            average = sum / count;
            System.out.println("Average for column " + col + " = " + average);
        }
        } // end of method main
    } // end of class Grades5
```

Aggregate Operations on 2-D Arrays (continued):

• Uneven rows example code (continued):

```java
    double sum;
    int count;
    double average;
    for (col = 0; col < 5; col++) {
        sum = 0.0;
        count = 0;
        for (row = 0; row < grades.length; row++)
            if (col < grades[row].length)
                sum = sum + grades[row][col];
        average = sum / count;
        System.out.println("Average for column " + col + " = " + average);
    }
    } // end of method main
```
Aggregate Operations on 2-D Arrays (continued):

Uneven rows (continued):

- Consider a 2-D array of String’s:
  ```java
  private String[][] presidents = {
      { "Thomas", "Jefferson" },
      { "William", "Henry", "Harrison" },
      { "Abraham", "Lincoln" },
      { "Hiram", "Ulysses", "Simpson", "Grant" },
      { "Rutherford", "Birchard", "Hayes" },
      { "Calvin", "Coolidge" },
      { "George", "Herbert", "Walker", "Bush" },
      { "William", "(Bill)", "Jefferson", "Clinton" } };
  ```

- Write a method that returns a String containing the third name of each President:
  ```java
  public String thirdNames() {
      String result = new String("\n");
      int row;
      for (row = 0; row < presidents.length; row++)
          if ( presidents[row][2] != null )
              result = result + presidents[row][2] + "\n";
      return result;
  }
  ```

- Write a method that takes an int argument that specifies which name is wanted:
  ```java
  public String names(int position) {
      String result = new String("\n");
      int row;
      for (row = 0; row < presidents.length; row++)
          if ( presidents[row][position] != null )
              result = result + presidents[row][position] + "\n";
      return result;
  }
  ```

- Call this method by doing:
  ```java
  for (int i = 0; i < 4; i++) {
      System.out.println("Names at position " + i + ":");
      System.out.println(presNames.names(i));
      System.out.println();
  }
  ```

- What happens if position has a value larger than any of the names in the array?
Aggregate Operations on 2-D Arrays (continued):

- Uneven rows example code (continued):
  - How to determine the largest number of columns without looking?
    - Becomes a problem of finding the largest value.
    - Each row “knows” how many columns it has.

```java
// Find the row with the most columns
int mostColumns = _________________________;   // assume first row has the most

// Check the rest of the rows for a row with more columns
for ( row = 1; row < grades.length; row++ )
  if ( ___________________________________________________ )
    mostColumns = grades[row].length;
```

```java
public class Grades6
{
  public static void main(String[] args)
  {
    double [][] grades = {
      { 91.0, 92, 93, 85.5 },
      { 87.5, 100, 92.5 },
      { 90, 69, 73, 82.5, 90.5 },
      { 82, 19, 100, 67.5, 98.5, 93.5 },
      { 75, 76, 72, 83.5 }
    };

    int row, col;
    double sum;
    int count;
    double average;

    // Find the row with the most columns
    int mostColumns = grades[0].length;   // assume first row has the most

    // Check the rest of the rows for a row with more columns
    for ( row = 1; row < grades.length; row++ )
      if ( mostColumns < grades[row].length )
        mostColumns = grades[row].length;
  }
}
```
Aggregate Operations on 2-D Arrays (continued):

- How to determine the largest number of columns without looking?

```java
// Compute and print average for each column
for ( col = 0; col < mostColumns; col++ ) {
    sum = 0.0;
    count = 0;
    for ( row = 0; row < grades.length; row++ )
        if ( col < grades[row].length ) {
            sum = sum + grades[row][col];
            count++;
        }
    average = sum / count;
    System.out.println("Average for column "+col+"= "+average);
}
```

} // end of method main

} // end of class Grades6