Lesson 9:

Introduction To Arrays

(Updated for Java 1.5 Modifications by Mr. Dave Clausen)

Lesson 9: Introduction To Arrays

Objectives:

- Write programs that handle collections of similar items.
- Declare array variables and instantiate array objects.
- Manipulate arrays with loops, **including the enhanced for loop**.
- Write methods to manipulate arrays.
- Create parallel arrays and two-dimensional arrays.

Lesson 9: Introduction To Arrays

Vocabulary:

- array
- element
- index
- initializer list
- logical size
- multi-dimensional array
- one-dimensional array
- structure chart
- parallel arrays
- physical size
- ragged array
- range bound error
- subscript
- two-dimensional array
- Enhanced for loop
- Procedural decomposition

9.1 Conceptual Overview

- An array consists of an ordered collection of similar items.
- An array has a single name, and the items in an array are referred to in terms of their position within the array.
- An array makes it is as easy to manipulate a million test scores as it is to manipulate three test scores.
- Once an array is declared, its size is fixed and cannot be changed.

9.1 Conceptual Overview

- Without arrays, a program with 20 test scores would look like this:

```
private String name;
private int test1,  test2,  test3,  test4,  test5,
            test6,  test7,  test8,  test9,  test10,
            test11, test12, test13, test14, test15,
            test16, test17, test18, test19, test20;
```

9.1 Conceptual Overview

- And the computation of the average score looks like this:

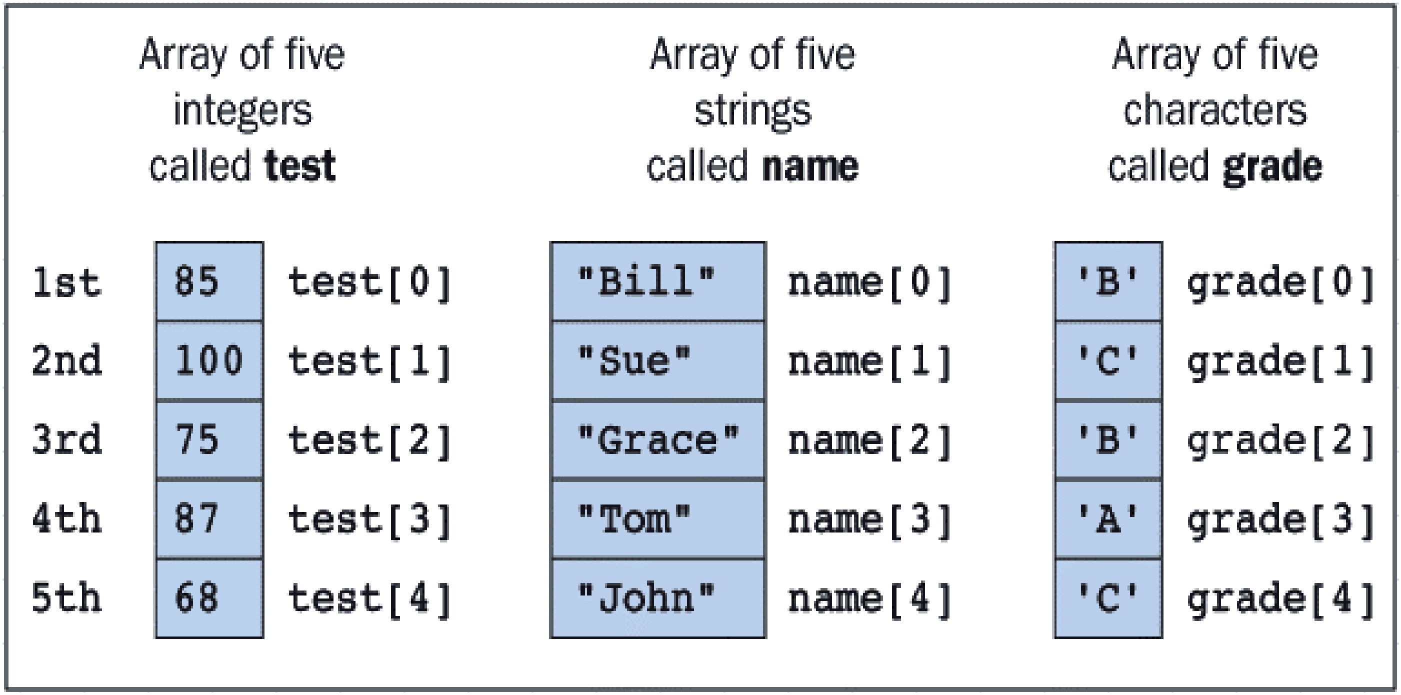
```
// Compute and return a student’s average
public int getAverage(){
    int average;
    average = (test1 + test2  + test3  + test4  + test5 +
              test6  + test7  + test8  + test9  + test10 +
              test11 + test12 + test13 + test14 + test15 +
              test16 + test17 + test18 + test19 + test20) / 20;
    return average;
}
```

9.1 Conceptual Overview

- The items in an array are called **elements**.
- For any particular array, all the elements must be of the same type.
- The type can be any primitive or reference type.
    - For instance, we can have an array of test scores (integers), an array of names (Strings), or even an array of student objects.
    - In figure 9-1 each array contains five elements, or has a length of five.
    - Remember that Java starts counting with 0 rather than 1

9.1 Conceptual Overview

- The first element in the array test is referred to as test[0], the second as test[1], and so on.
- An item's position within an array is called its index or subscript.
- Array indexes appear within square brackets [ ]



Array of five integers called **test**

Array of five strings called **name**

Array of five characters called **grade**

1st	85	test[0]	"Bill"	name[0]	'B'	grade[0]
2nd	100	test[1]	"Sue"	name[1]	'C'	grade[1]
3rd	75	test[2]	"Grace"	name[2]	'B'	grade[2]
4th	87	test[3]	"Tom"	name[3]	'A'	grade[3]
5th	68	test[4]	"John"	name[4]	'C'	grade[4]

9.2 Simple Array Manipulations

- First we **declare and instantiate** an array of 500 integer values.
- By default, all of the values are initialized to 0:

```
int[] abc = new int[500];
```

- Next, we declare some other variables:

```
int i = 3;
int temp;
double avFirstFive;
```

9.2 Simple Array Manipulations

- The basic syntax for referring to an array element has the form:

  <array name>[<index>]

- Where <index> must be between 0 and the array's length less 1.
- The subscript operator ([ ]) has the same precedence as the method selector (.).

9.2 Simple Array Manipulations

- For example we assign values to the first five elements:

```
abc[0] = 78;                        //1st element 78
abc[1] = 66;                        //2nd element 66
abc[2] = (abc[0] + abc[1]) / 2;     //3rd element average of first two
abc[i] = 82;                        //4th element 82 because i is 3
abc[i + 1] = 94;                    //5th element 94 because i + 1 is 4
```

- When assigning a value to the 500th element, we must remember that its index is 499, not 500:

```
abc[499] = 76;                  //500th element 76
```

9.2 Simple Array Manipulations

- The JVM checks the values of subscripts before using them and throws an ArrayIndexOutOfBoundsException if they are out of bounds (less than 0 or greater than the array length less 1).
- The detection of a ***range bound error*** is similar to the JVM's behavior when a program attempts to divide by 0.
- To compute the average of the first five elements, we could write:

```
avFirstFive = (abc[0] + abc[1] + abc[2] + abc[3] + abc[4])/5;
```

9.2 Simple Array Manipulations

- It often happens that we need to interchange elements in an array. (the basic idea behind a simple **sort**)

```
// Initializations
. . .
abc[3] = 82;
abc[4] = 95;
i = 3;
. . .

// Interchange adjacent elements
temp = abc[i];                     // temp       now equals 82
abc[i] = abc[i + 1];               // abc[i]     now equals 95
abc[i + 1] = temp;                // abc[i + 1]  now equals 82
```

9.2 Simple Array Manipulations

- We frequently need to know an array's length.
- The array itself makes this information available by means of a public instance variable called length:

```
System.out.println ("The size of abc is: " + abc.length);
//Just abc.length, no ( ) are used with the length variable
//In C++, length was a function and required ( )
```

9.3 Looping Through Arrays

Sum the Elements

- The following code sums the numbers in the array abc.
- Each time through the loop adds a different element to the sum. On the first iteration we add abc[0] and on the last abc[499].

```
int sum;
sum = 0;
for (int i = 0; i < 500; i++)
 sum += abc[i];
```

9.3 Looping Through Arrays

Count the Occurrences

- We can determine how many times a number x occurs in the array by comparing x to each element and incrementing count every time there is a match:

```
int x;
int count;
x = ...;                        //Assign some value to x
count = 0;
for (int i = 0; i < 500; i++){
   if (abc[i] == x)
      count++;                  //Found another element equal to x
}
```

9.3 Looping Through Arrays

Determine Presence or Absence

- To determine if a particular number is present in the array, programmers can end the loop as soon as the first match is found, using indefinite (while) loops.
- The Boolean variable found indicates the outcome of the search.

9.3 Looping Through Arrays

- Determine absence or presence

Source code w/o break statement as opposed to Page 309

```
int x, counter=0;
boolean notFound = true;
x = ...;  //number to search for
   while ((counter < list.length) && notFound )
   {
      if (list[counter] == x)
         notFound = false;
    counter++;
   }
   if (notFound)
     System.out.println (“Not Found”);
   else
     System.out.println (“Found”);
```

9.3 Looping Through Arrays

- Determine first location

Source code w/o break statement as opposed to Page 310

```
int x, counter=0,location=0;
boolean notFound = true;
x = ...;  //number to search for
   while ((counter < list.length) && notFound ){
      if (list[counter] == x)
      {
         location = counter;
         notFound = false;
      }
    counter++;
   }
if (notFound)
     System.out.println (“Not Found”);
   else
     System.out.println (“Found at index # ” + location);
```

9.3 Looping Through Arrays

Working With Arrays of Any Size

- It is possible and also desirable to write code that works with arrays of any size.
- Simply replace the literal 500 with a reference to the array's instance variable `length` in each of the loops.
- For example, this code would sum the integers in an array of any size:

```
int sum;
sum = 0;
for (int i = 0; i < abc.length; i++)
 sum += abc[i];
```

9.4 Declaring Arrays

- Arrays are objects and must be instantiated before being used.
- Several array variables can be declared in a single statement like this:

```
int[] abc, xyz;
abc = new int[500];
xyz = new int[10];
```

- **Or like this:**

```
int[] abc = new int[500];
int[] xyz = new int[10];
```

9.4 Declaring Arrays

- Array variables are null before they are assigned array objects.
- Failure to assign an array object can result in a null pointer exception.

```
int[] abc;
abc[1] = 10; // runtime error: null pointer exception
```

9.4 Declaring Arrays

- Because arrays are objects, all rules that apply to objects apply to arrays.
  - Two array variables may refer to same array.
  - Arrays may be garbage collected.
  - Array variables may be set to `null`.
  - Arrays are passed by reference to methods.

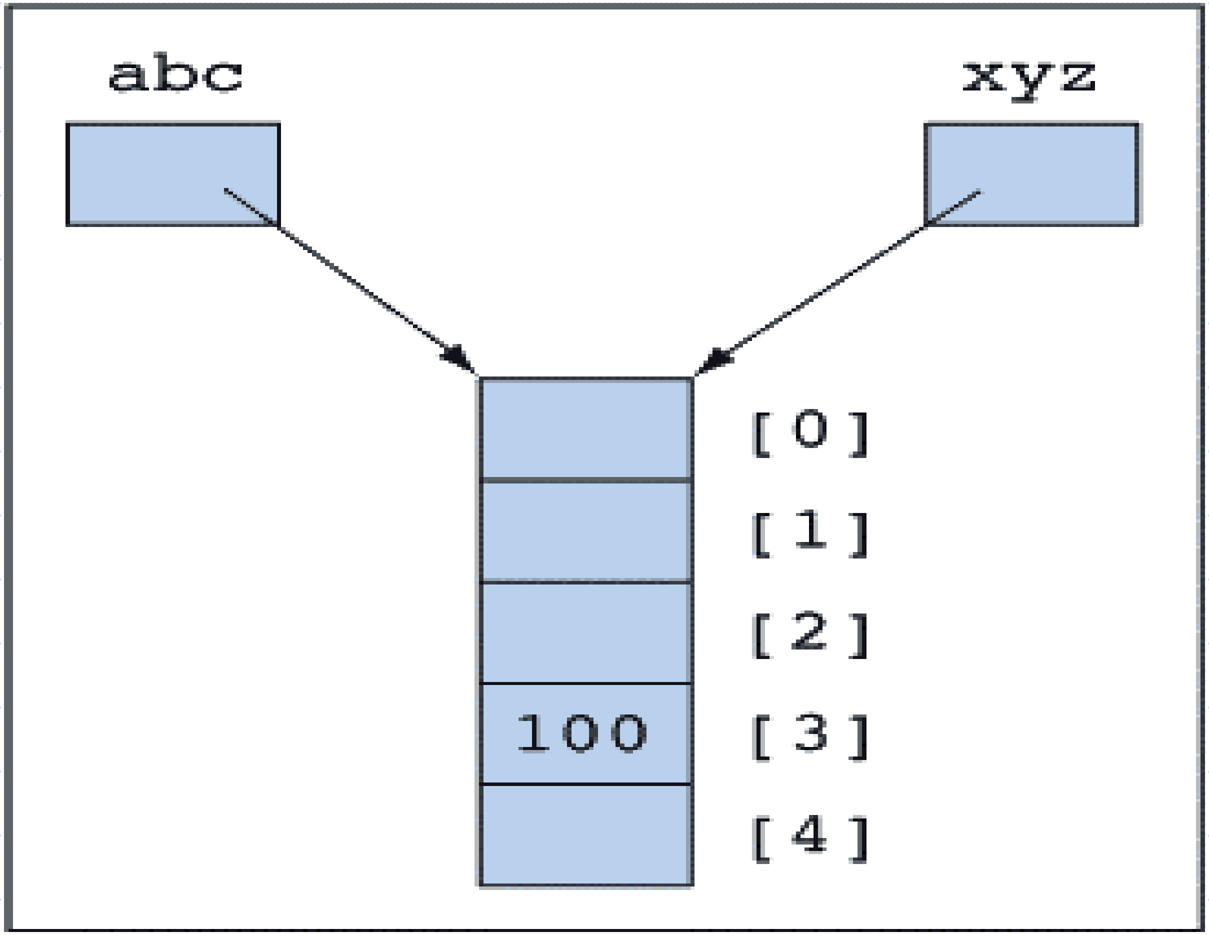
9.4 Declaring Arrays

- Because arrays are objects, two variables can refer to the same array as indicated in Figure 9-2 and the next segment of code:

```
int[] abc, xyz;
abc = new int[5]; // Instantiate an array of 5 integers
xyz = abc;         // xyz and abc refer (point) to the same
                   // array, they are not duplicate arrays
xyz[3] = 100;      // Changing xyz changes abc as well.
System.out.println (abc[3]);  // 100 is displayed
```

9.4 Declaring Arrays

Both variables point to the same array object



9.4 Declaring Arrays

- If you wish to make a copy of the array:

```
int [] abc, xyz;
abc = new int [10];
xyz = new int [10];
//fill the first array with numbers
 for (int counter = 0; counter < 10; counter ++)
    abc[counter] = counter * counter;

//copy the original array
 for (int counter = 0; counter < 10; counter ++)
    xyz[counter] = abc[counter];
```

9.4 Declaring Arrays

- Arrays can be declared, instantiated, and initialized in one step.
- The list of numbers between the braces is called an ***initializer list***.

```
int[] abc = {1,2,3,4,5} // abc now references an array of five integers.
```

- Here then are examples of arrays of doubles, Booleans, strings, and students:

9.4 Declaring Arrays

```
double[]   ddd = new double[10];
//error in book Page 313 array of char
boolean[]  bbb = new boolean[10];
String[]   ggg = new String[10];
Student[]  sss = new Student[10];
String     str;

ddd[5] = 3.14;

bbb[5] = true;
ggg[5] = "The cat sat on the mat.";
sss[5] = new Student();

sss[5].setName ("Bill");
str = sss[5].getName() + ggg[5].substring(7);
 // str now equals "Bill sat on the mat."
```

9.4 Declaring Arrays

- There is another way to declare array variables. but its use can be confusing.
- Here it is:

```
int aaa[];            // aaa is an array variable.
```

- Once an array is instantiated, its size cannot be changed. Make sure that the array is large enough when you instantiate it.

9.5 Working with Arrays That Are Not Full

- One might create an array of 20 ints but receive only 5 ints from interactive input.
- This array has a ***physical size*** of 20 cells but a ***logical size*** of 5 cells currently used by the application.
- From the application's perspective, the remaining 15 cells contain garbage.

9.5 Working with Arrays That Are Not Full

- It is possible to track the array's logical size with a separate integer variable.
- The following code segment shows the initial state of an array and its logical size:
  - Note that abc.length (the physical size) is 50, whereas size (the logical size) is 0

```
int[] abc = new int[50];
int logical_size = 10;
```

9.5 Working with Arrays That Are Not Full

- To work with arrays that are not full, the programmer must track the logical array size.
  - Declare an integer counter that will always indicate the number of elements.
  - Every time an element is added or removed, adjust the counter accordingly.
  - The counter indicates the logical size of the array and the next open position in the array.

9.5 Working with Arrays That Are Not Full

Processing Elements in an Array That Is Not Full

- When the array is not full, one must replace the array's length with its logical size in the loop.
- Here is the code for computing the sum of the integers currently available in the array abc:

9.5 Working with Arrays That Are Not Full

```
int[] abc = new int[50];
int logical_size = 10;

... code that puts values into some initial
portion of the array and sets
    the value of size ...

int sum = 0;
// logical_size contains the number items in the array
for (int i = 0; i < logical_size; i++)
 sum += abc[i];
```

9.5 Working with Arrays That Are Not Full

Adding Elements to an Array

- The simplest way to add a data element to an array is to place it after the last available item.
- One must first check to see if there is a cell available and then remember to increment the array's logical size.
- The following code shows how to add an integer to the end of array abc:

9.5 Working with Arrays That Are Not Full

```
if (logical_size < abc.length)
{
   abc[logical_size] = anInt;
   logical_size ++;
}
```

- When **logical_size** equals abc.length, the array is full.
- The if statement prevents a range error from occurring.
- Remember that Java arrays are of fixed size when they are instantiated, so eventually they become full.

9.5 Working with Arrays That Are Not Full

Removing Elements from an Array

- Removing a data element from the end of an array requires no change to the array itself.
- Simply decrement the logical size, thus preventing the application from accessing the garbage elements beyond that point.

9.5 Using an Array with a Text File

- Here is an example of an array that is saved to a text file:

ArrayToFile.java

ArrayToFile.txt

Remember, don’t use a break to escape from a loop!

ArrayToFileNoBreak.java

ArrayToFileNoBreak.txt

9.5 Using an Array with a Text File (cont.)

- Here is an example of an array that is read from a text file:

FileToArray.java

FileToArray.txt

numbers.txt

9.6 Parallel Arrays

- Suppose we want to keep a list of people's names and ages.
- This can be achieved by using two arrays in which corresponding elements are related.

```
String[] name = {"Bill", "Sue", "Shawn", "Mary", "Ann"};
int[]    age  = {20    , 21   , 19     , 24    , 20};
```

- Thus, Bill's age is 20 and Mary's is 24.

9.6 Parallel Arrays

- Here is a segment of code that finds the age of a particular person.
- In this example, the parallel arrays are both full and the loops use the instance variable length.
- When the arrays are not full, the code will need an extra variable to track their logical sizes.

9.6 Parallel Arrays

```
String searchName;
int correspondingAge = -1;
int i;

searchName = ...;                   // Set this to the desired name
for (i = 0; i < name.length; i++){  // name.length is the array’s size
   if (searchName.equals (name[i]){
      correspondingAge = age[i];
      break;  //Don’t use break to end a loop! (See next slide)
   }
}

if (correspondingAge == -1)
   System.out.println(searchName + " not found.");
else
 System.out.println("The age is " + correspondingAge);
```

9.6 Parallel Arrays

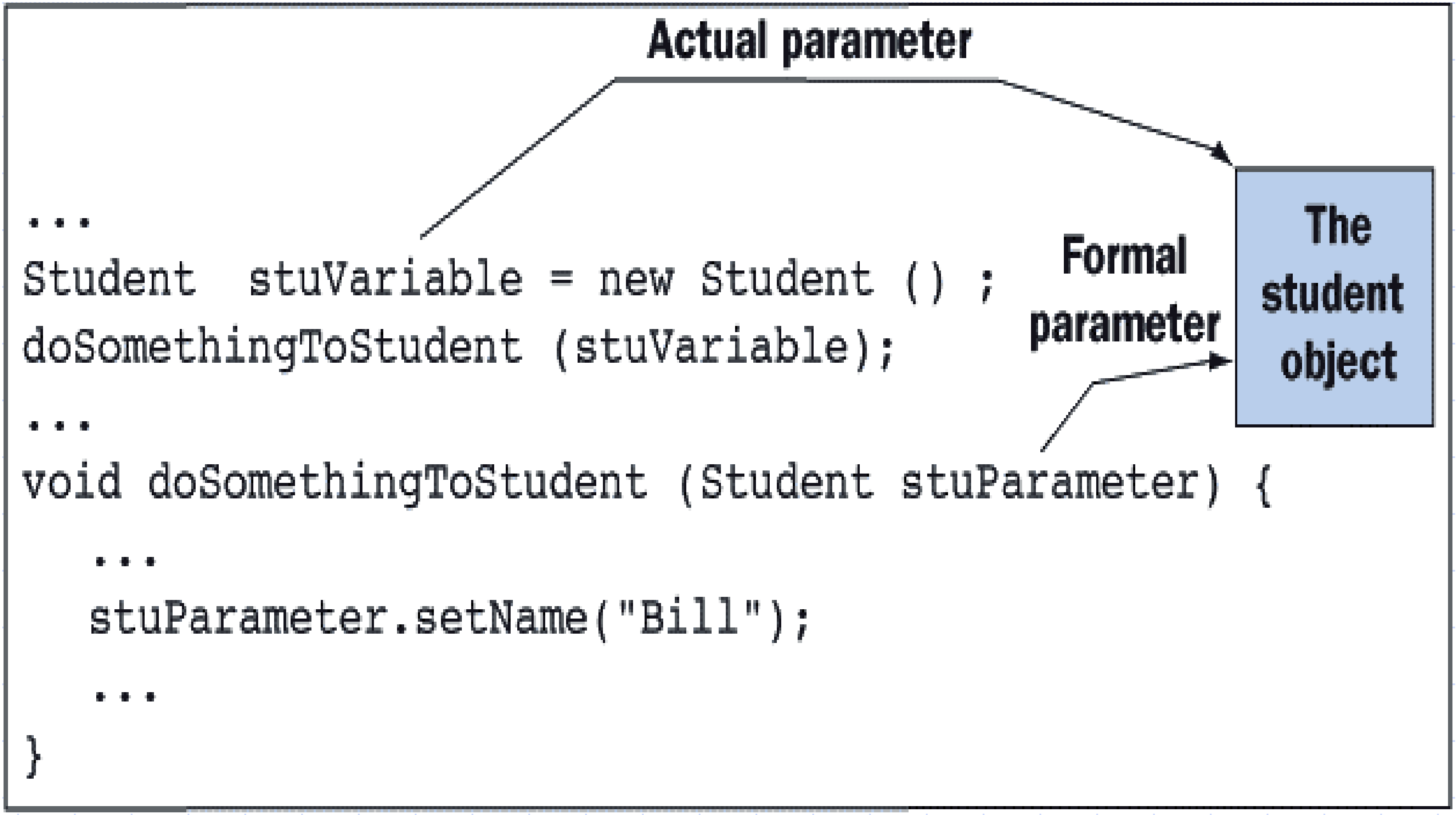
Code w/o break statement: Page 317

```
String searchName;
int correspondingAge = 0, counter =0;
boolean notFound = true;
searchName = ...;    // Set this to the desired name
while ((counter < name.length)&&(notFound))
{
   if (searchName.equals (name[counter]) //don’t use ==
   {
      correspondingAge = age[counter];
      notFound = false;
   }
   counter ++;
}

if (notFound)
   System.out.println(searchName + " not found.");
else
  System.out.println("The age is " + correspondingAge);
```

9.9 Arrays and Methods

- When any object is used as a parameter to a method, what actually gets passed is a reference to the object and not the object itself.
- The actual and formal parameters refer to the same object, and changes the method makes to the object's state are still in effect after the method terminates.
- In the figure, the method changes the student's name to Bill, and after the method finishes executing the name is still Bill.

9.9 Arrays and Methods



```
...
Student  stuVariable = new Student () ;
doSomethingToStudent (stuVariable);
...
void doSomethingToStudent (Student stuParameter) {
   ...
   stuParameter.setName("Bill");
   ...
}
```

9.9 Arrays and Methods

- Arrays are objects, so the same rules apply.
- When an array is passed as a parameter to a method, the method manipulates the array itself and not a copy.
- Changes made to the array in the method are still in effect after the method has completed its execution.
- A method can also instantiate a new object or a new array and return it using the `return` statement.

9.9 Arrays and Methods

Sum the Elements

- The following method computes the sum of the numbers in an integer array.
- When the method is written, there is no need to know the array's size.
- The method works equally well with integer arrays of all sizes, as long as those arrays are full

9.9 Arrays and Methods

```
int Sum (int[] a)
{
   int i, result = 0;
   for (i = 0; i < a.length; i++)
      result += a[i];
   return result;
}
```

- Using the method is straightforward:

```
int[] array1 = {10, 24, 16, 78, -55, 89, 65};
int[] array2 = {4334, 22928, 33291};
...
if (Sum(array1) > Sum(array2)) ...
```

9.9 Arrays and Methods

Search for a Value

- The code to search an array for a value is used so frequently in programs that it is worth placing in a method.
- Here is a method to search an array of integers.
- The method returns the location of the first array element equal to the search value or -1 if the value is absent:

```
int search (int[] a, int searchValue){
   int location, i;
   location = -1;
   for (i = 0; i < a.length; i++){
      if (a[i] == searchValue){
         location = i;
         break; //DO NOT USE BREAK!(see next slide)
      }
   }
   return location;
}
```

9.9 Arrays and Methods

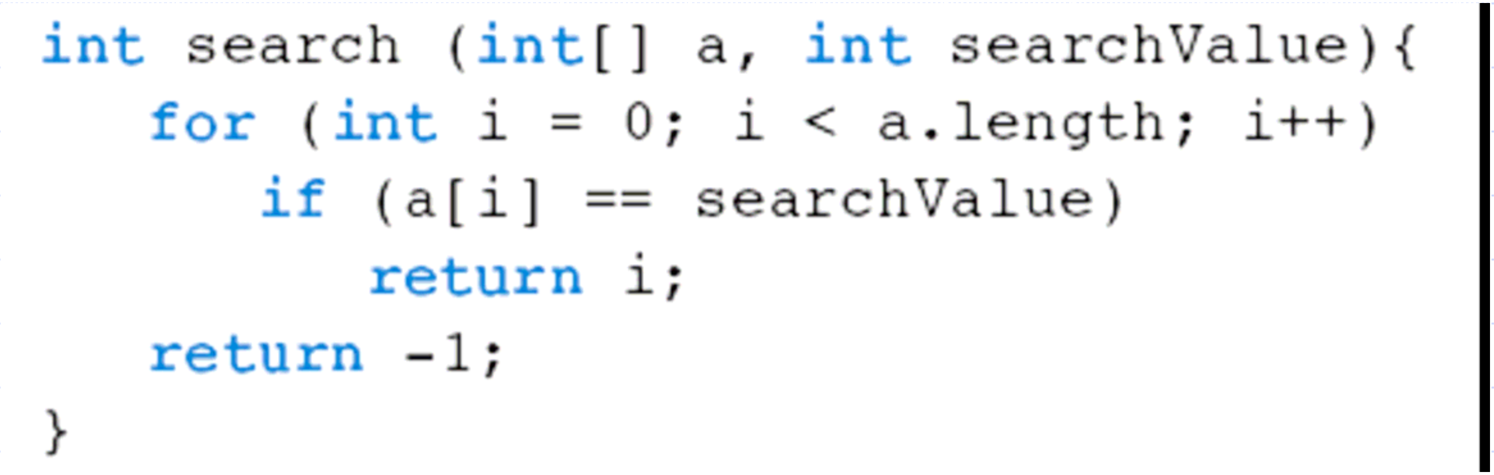
A Linear Search for a Value Without using a Break Statement

(Use this style) (LinearSearch.ppt)

```
int Linear_Search (int[] list, int searchValue)
{
   int location, counter=0;
   boolean notFound = true;
   location = -1;
   while ((counter < list.length) && notFound))
   {
      if (list[counter] == searchValue)
      {
         location = counter;
         notFound = false;
      }
    counter++;
   }
   return location;
}
```

9.9 Arrays and Methods

- Method to search for a value in an array:



```
int search (int[] a, int searchValue){
    for (int i = 0; i < a.length; i++)
        if (a[i] == searchValue)
            return i;
    return -1;
}
```

9.9 Arrays and Methods

Sum the Rows

- Here is a method that instantiates a new array and returns it. The method computes the sum of each row in a two-dimensional array and returns a one-dimensional array of row sums. The method works even if the rows are not all the same size.

```
int[] Sum_Rows (int[][] list){
   int i, j;
   int[] rowSum = new int[list.length];
   for (i = 0; i < list.length; i++){
      for (j = 0; j < list[i].length; j++){
         rowSum[i] += list[i][j];
      }
   }
   return rowSum;
}
```

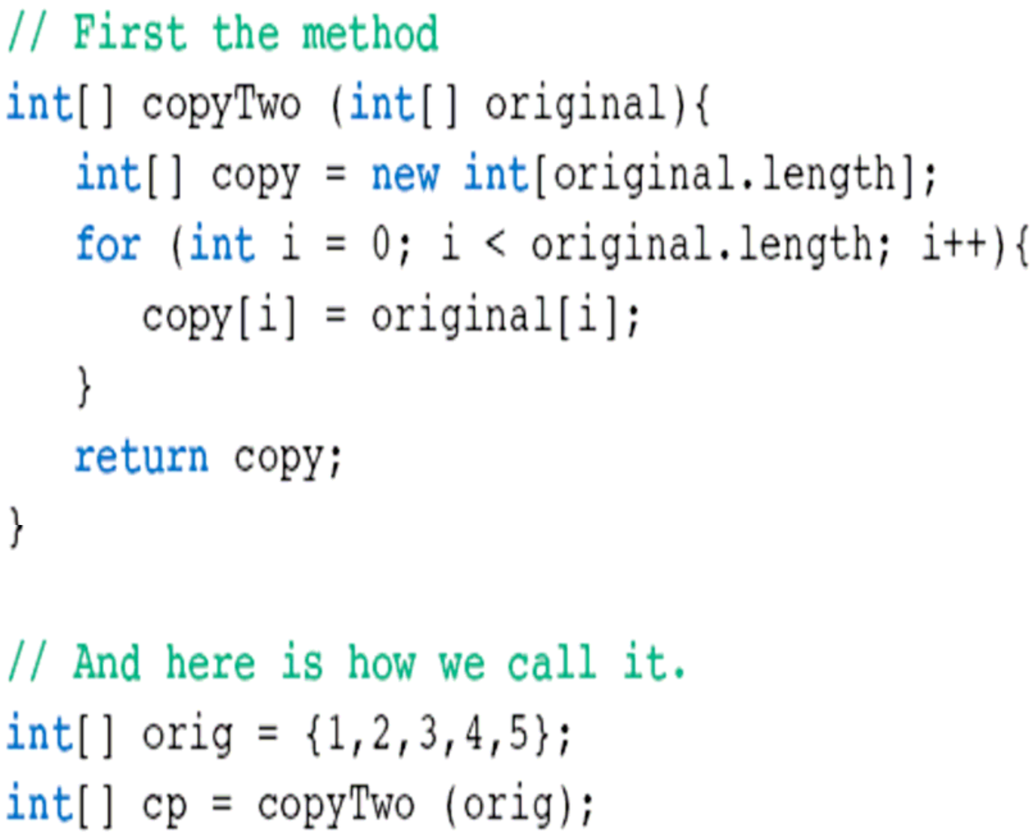
9.9 Arrays and Methods

- Here is code that uses the method.
- We do not have to instantiate the array oneD because that task is done in the method sumRows.

```
int[][] twoD = {{1,2,3,4}, {5,6}, {7,8,9}};
int[] oneD;

oneD = sumRows (twoD); // oneD now references the array created and returned
                      // by the method sumRows. It equals {10, 11, 24}
```

9.9 Arrays and Methods

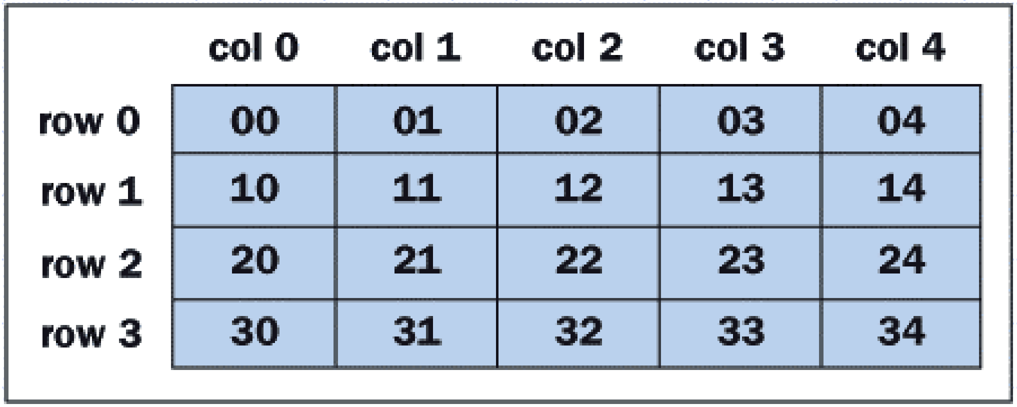
- Method to make a copy of an array and return it:



```
// First the method
int[] copyTwo (int[] original){
   int[] copy = new int[original.length];
   for (int i = 0; i < original.length; i++){
      copy[i] = original[i];
   }
   return copy;
}

// And here is how we call it.
int[] orig = {1,2,3,4,5};
int[] cp = copyTwo (orig);
```

9.7 Two-Dimensional Arrays

- A table of numbers, for instance, can be implemented as a ***two-dimensional array***. Figure 9-3 shows a two-dimensional array with four rows and five columns.



	col 0	col 1	col 2	col 3	col 4
row 0	00	01	02	03	04
row 1	10	11	12	13	14
row 2	20	21	22	23	24
row 3	30	31	32	33	34

- Suppose we call the array `table`; then to indicate an element in `table`, we specify its row and column position, remembering that indexes start at 0:

  x = table[2][3]; // Set x to 23, the value in (row 2, column 3)

9.7 Two-Dimensional Arrays

Sum the Elements

- Here is code that sums all the numbers in table.
- The outer loop iterates four times and moves down the rows.
- Each time through the outer loop, the inner loop iterates five times and moves across a different row.

9.7 Two-Dimensional Arrays

The sum of **all** of the elements in the table (two-dimensional array).

```
int i, j;
int sum = 0;
for (i = 0; i < 4; i++)
{     // There are four rows: i = 0,1,2,3
   for (j = 0; j < 5; j++)
   { // There are five columns: j = 0,1,2,3,4
      sum += table[i][j];
   }
}
```

9.7 Two-Dimensional Arrays

- This segment of code can be rewritten without using the numbers 4 and 5.
  - The value table.length equals the number of rows,
  - Table[i].length is the number of columns in row i.

```
int i, j;
int sum = 0;
for (i = 0; i < table.length; i++){
   for (j = 0; j < table[i].length; j++){
      sum += table[i][j];
   }
}
```

9.7 Two-Dimensional Arrays

Sum the Rows

- We now compute the sum of each row separately and place the results in a one-dimensional array called `rowSum`.
- This array has four elements, one for each row of the table.
- The elements in `rowSum` are initialized to 0 automatically by virtue of the declaration.

9.7 Two-Dimensional Arrays

Sum the Rows

```
int i, j;
int[] rowSum = new int[4];
for (i = 0; i < table.length; i++)
{
   for (j = 0; j < table[i].length; j++)
   {
      rowSum[i] += table[i][j];
   }
}
```

9.7 Two-Dimensional Arrays

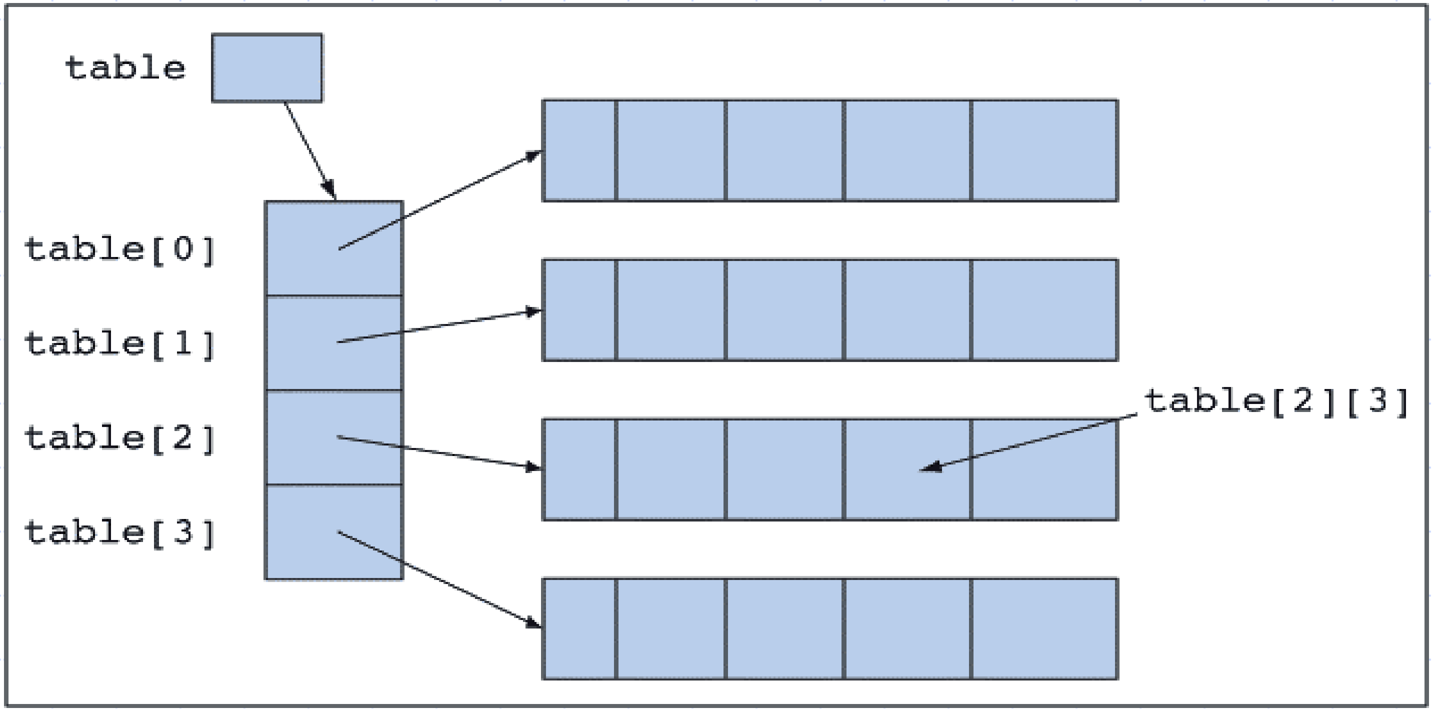
Declare and Instantiate

- Declaring and instantiating two-dimensional arrays are accomplished by extending the processes used for one-dimensional arrays:

```
int[][] table;          // The variable table can reference a
                        // two-dimensional array of integers
table = new int[4][5];  // Instantiate table as an array of size 4,
                        // each of whose elements will reference an array
                        // of 5 integers.
```

9.7 Two-Dimensional Arrays

- The variable `table` references an array of four elements.
- Each of these elements in turn references an array of five integers.



9.7 Two-Dimensional Arrays

- Initializer lists can be used with two-dimensional arrays. This requires a list of lists.
- The number of inner lists determines the number of rows, and the size of each inner list determines the size of the corresponding row.
- The rows do not have to be the same size, but they are in this example:

```
int[][] table = {{ 0, 1, 2, 3, 4},     // row 0
                 {10,11,12,13,14},     // row 1
                 {20,21,22,23,24},     // row 2
                 {30,31,32,33,34}};    // row 3
```

9.7 Two-Dimensional Arrays

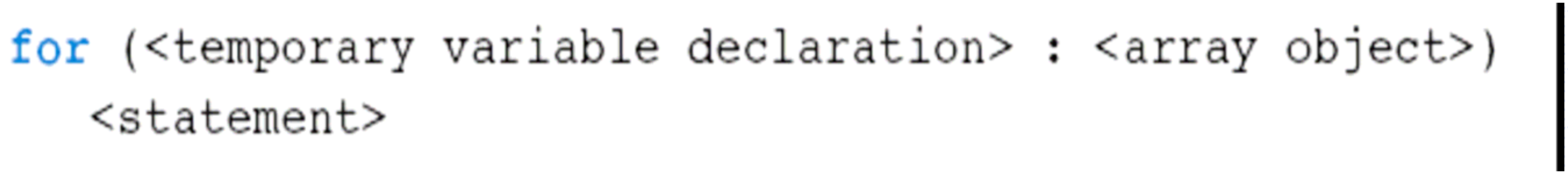
Variable Length Rows

- ***Ragged arrays*** are rows of a two-dimensional arrays that are not all the same length.

```
int[][] table;
table    = new int[4][];  // table has 4 rows
table[0] = new int[6];    // row 0 has 6   elements
table[1] = new int[10];   // row 1 has 10  elements
table[2] = new int[100];  // row 2 has 100 elements
table[3] = new int[1];    // row 3 has 1   element
```

9.8 The Enhanced for Loop

- Provided in Java 5.0 to simplify loops in which each element in an array is accessed
  - From the first index to the last
  - Frees programmer from having to manage and use loop counters
- Syntax:

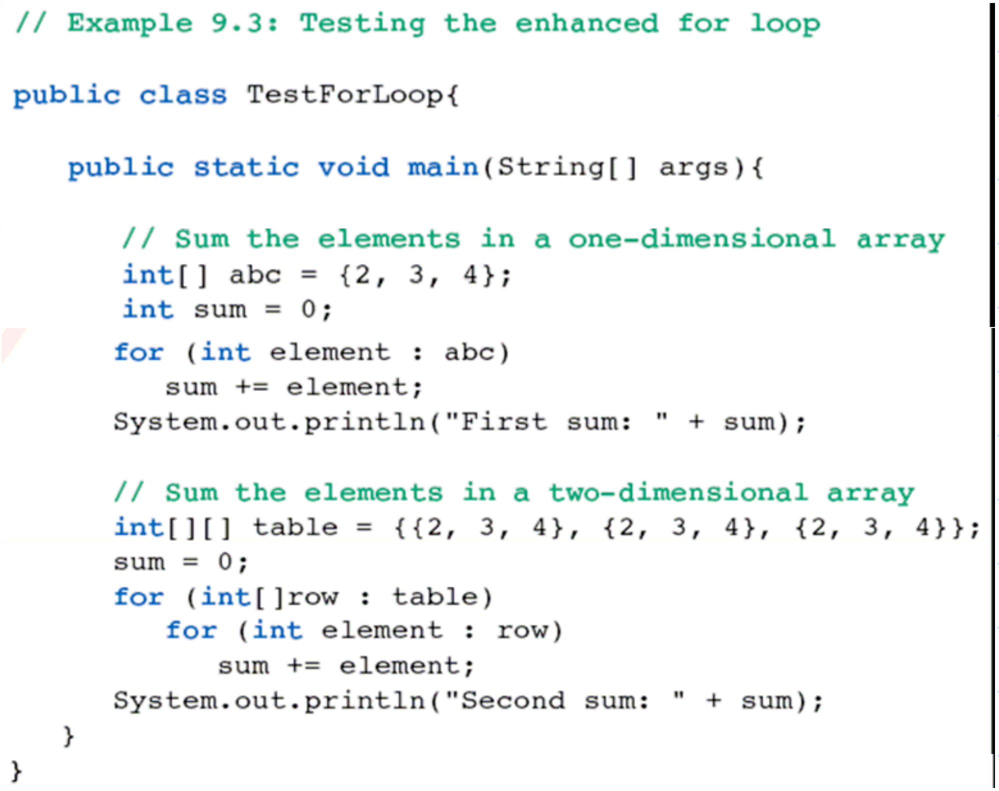


```
for (<temporary variable declaration> : <array object>)
   <statement>
```

9.8 The Enhanced for Loop

- Example 9.3: Testing the enhanced `for` loop

  [TestForLoop.java] [TestForLoop.txt]



```
// Example 9.3: Testing the enhanced for loop

public class TestForLoop{

   public static void main(String[] args){

      // Sum the elements in a one-dimensional array
      int[] abc = {2, 3, 4};
      int sum = 0;
     for (int element : abc)
        sum += element;
     System.out.println("First sum:  " + sum);

     // Sum the elements in a two-dimensional array
     int[][] table = {{2, 3, 4}, {2, 3, 4}, {2, 3, 4}};
     sum = 0;
     for (int[]row : table)
        for (int element : row)
           sum += element;
     System.out.println("Second sum:  " + sum);
   }
}
```

9.8 The Enhanced for Loop

- Cannot be used to:
  - Move through an array in reverse, from the last position to the first position
  - Assign elements to positions in an array
  - Track the index position of the current element in an array
  - Access any element other than the current element on each pass

9.10 Arrays of Objects

- Arrays can hold objects of any type, or more accurately, references to objects.
- For example, one can declare, instantiate and fill an array of students as follows:

```
// Declare and reserve 10 cells for student objects
Student[] studentArray = new Student[10];

// Fill array with students
for (int i = 0; i < studentArray.length; i++)
 studentArray[i] = new Student("Student " + i, 70+i, 80+i, 90+i);
```

9.10 Arrays of Objects

- When an array of objects is instantiated, each cell is null by default until reset to a new object.
- The next code segment prints the average of all students in the studentArray. Pay special attention to the technique used to send a message to objects in an array:

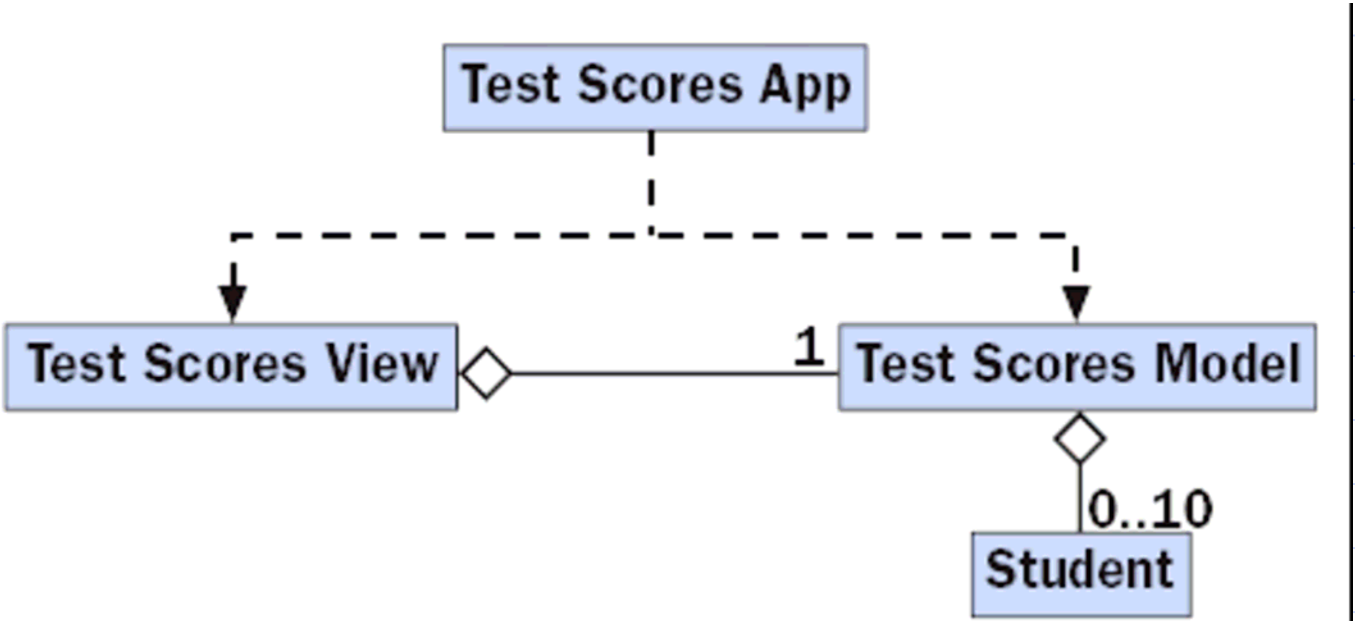
```
// Print the average of all students in the array.

int sum = 0;
for (int i = 0; i < studentArray.length; i++)
   sum += studentArray[i].getAverage();     // Send message to object in array
System.out.println("The class average is " + sum / accountArray.length);
```

Case Study Design Techniques

- **UML diagrams**: Industry standard for designing and representing a set of interacting classes
- **Structure charts:** May be used to depict relationships between classes and the order of methods called in a program
- **Procedural decomposition:** Technique of dividing a problem into sub-problems and correlating each with a method

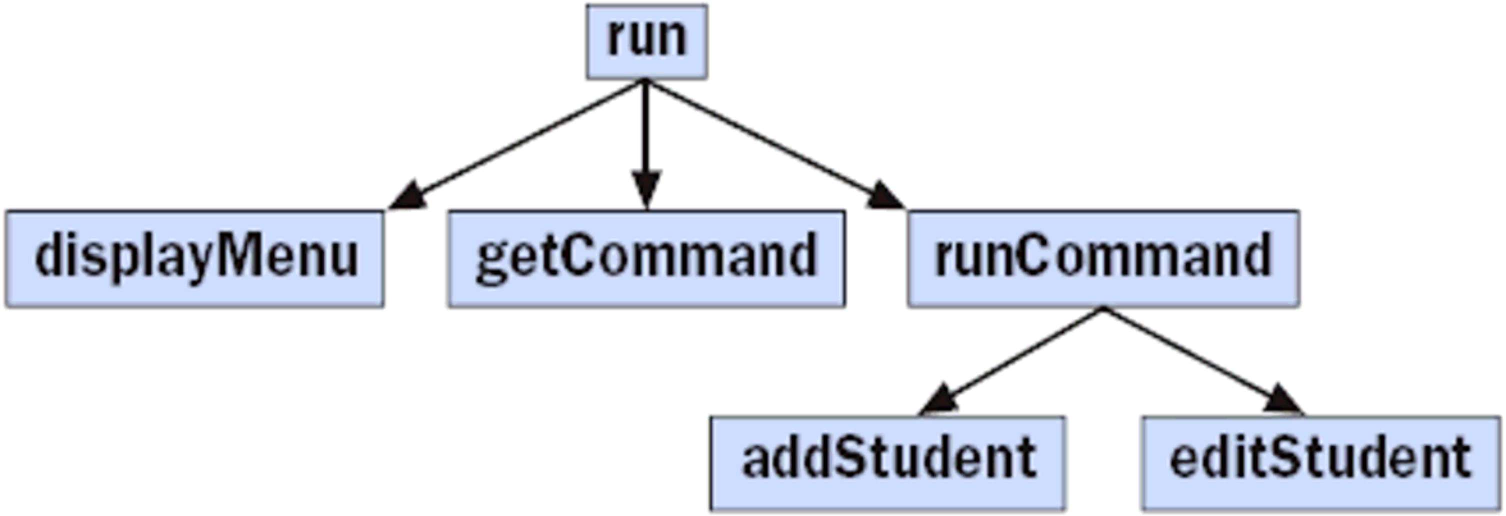
Case Study Design Techniques (cont.)

Figure 9-6: UML diagram of the classes in the student test scores program



Case Study Design Techniques (cont.)

- Figure 9-7: Structure chart for the methods of class `TestScoresView`



Case Study

TestModel.java

TestModel.txt

Student.java

Student.txt

TestScoresModel.java

TestScoresModel.txt

TestScoresView.java

TestScoresView.txt

Design, Testing, and Debugging Hints

- To create an array:
  - **1.** Declare an array variable.
  - **2.** Instantiate an array object and assign it to the array variable.
  - **3.** Initialize the cells in the array with data, as appropriate.
- When creating a new array object, try to determine an accurate estimate of the number of cells required.

Design, Testing, and Debugging Hints (cont.)

- Remember that array variables are `null` until they are assigned array objects.
- To avoid index out-of-bounds errors, remember that the index of an array cell ranges from 0 (the first position) to the length of the array minus 1.
- To access the last cell in an array, use the expression `<`*`array`*`>.length – 1`.

Design, Testing, and Debugging Hints (cont.)

- Avoid having multiple array variables refer to the same array.
- To copy the contents of one array to another, do not use `A = B;` instead, write a copy method and use `A = arrayCopy(B);`.
- When an array is not full:
  - Track the current number of elements
  - Avoid index out-of-bounds errors

Summary

- Arrays are collections of similar items or elements ordered by position.
- Arrays are useful when a program needs to manipulate many similar items, such as a group of students or a number of test scores.
- Arrays are objects.
  - Must be instantiated
  - Can be referred to by more than one variable

Summary (cont.)

- An array can be passed to a method as a parameter and returned as a value.
- Parallel arrays are useful for organizing information with corresponding elements.
- Two-dimensional arrays store values in a row-and-column arrangement.

Summary (cont.)

- An enhanced `for` loop is a simplified version of a loop for visiting each element of an array from the first position to the last position.