Chapter 5 - Arrays

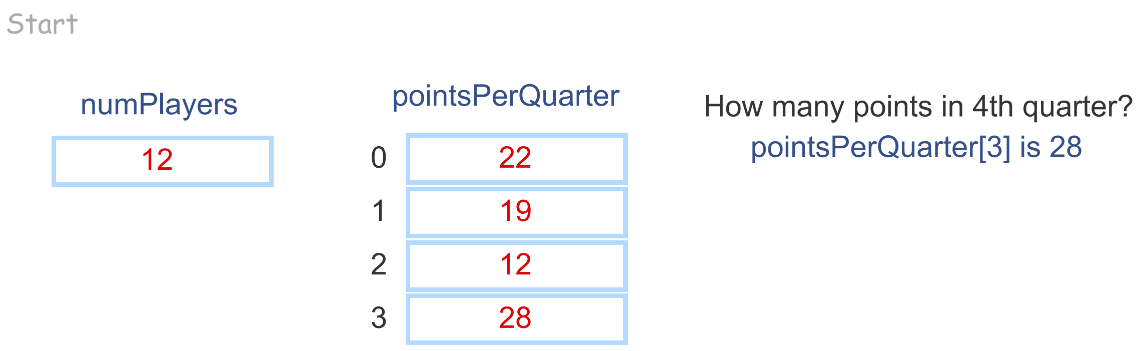
Section 5.1 - Array concept

Note_language_neutral

A typical variable stores one data item, like the number 59 or the character 'a'. Instead, sometimes a *list* of data items should be stored. Ex: A program recording points scored in each quarter of a basketball game needs a list of 4 numbers. Requiring a programmer to define 4 variables is annoying; 200 variables would be ridiculous. An ***array*** is a special variable having one name, but storing a list of data items, with each item directly accessible. Some languages use a construct similar to an array called a ***vector***. Each item in an array is known as an ***element***.

P Participation Activity

5.1.1: Sometimes a variable should store a list, or array, of data items.



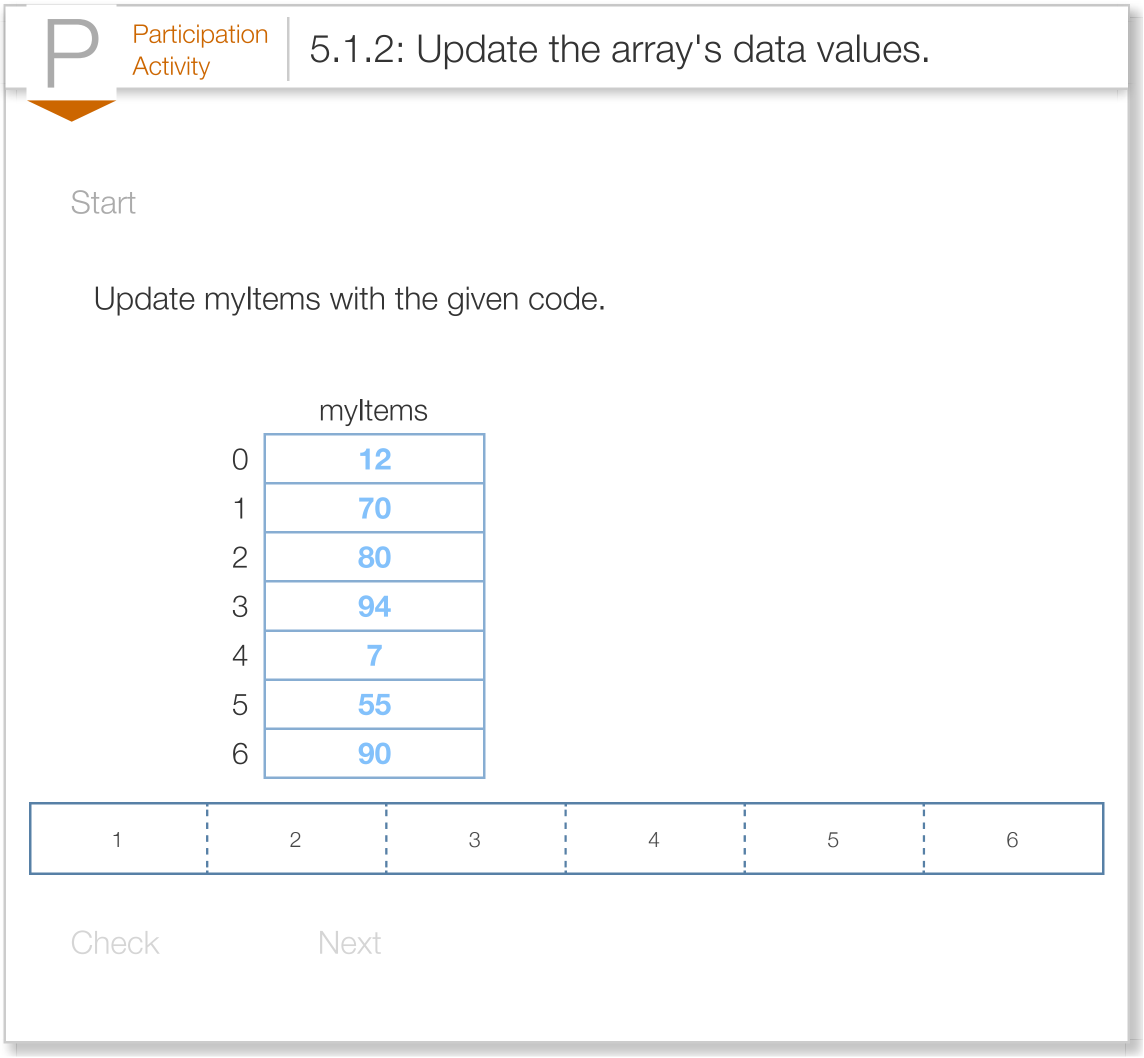
How many points in 4th quarter?
pointsPerQuarter[3] is 28

You might think of a normal variable as a truck, and an array variable as a train. A truck has just one car for carrying "data", but a train has many cars each of which can carry data.

Figure 5.1.1: A normal variable is like a truck, whereas an array variable is like a train.

(Source for above images: Truck, Train)

In an array, each element's location number is called the ***index***; myArray[2] has index 2. An array's key feature is that the index enables direct access to any element, as in myArray[2]; different languages may use different syntax, like myArray(3) or myVector.at(3). In many languages, indices start with 0 rather than 1, so an array with 4 elements has indices 0, 1, 2, and 3.



P Participation Activity

5.1.2: Update the array's data values.

Update myItems with the given code.

	myItems
0	12
1	70
2	80
3	94
4	7
5	55
6	90

Participation Activity

5.1.3: Array basics.

Array peoplePerDay has 365 elements, one for each day of the year. Valid accesses are peoplePerDay[0], [1], ..., [364].

#	Question	Your answer

1	Which assigns element 0 with the value 250?	peoplePerDay[250] = 0
		peoplePerDay[0] = 250
		peoplePerDay = 250

2	Which assigns element 1 with the value 99?	peoplePerDay[1] = 99
		peoplePerDay[99] = 1

3	Given the following statements:
	`peoplePerDay[9] = 5;`
	`peoplePerDay[8] = peoplePerDay[9] - 3;`
	What is the value of peoplePerDay[8]?
		8
		5
		2

4	Assume N is initially 1. Given the following:
	`peoplePerDay[N] = 15;`
	`N = N + 1;`
	`peoplePerDay[N] = peoplePerDay[N - 1] * 3;`
	What is the value of peoplePerDay[2]?
		15
		2
		45

Participation Activity

5.1.4: Arrays with element numbering starting with 0.

Array scoresList has 10 elements with indices 0 to 9, accessed as scoresList[0] to scoresList[9].

#	Question	Your answer
1	Assign the first element in scoresList with 77.	
2	Assign the second element in scoresList with 77.	
3	Assign the last element with 77.	
4	If that array instead has 100 elements, what is the last element's index?	
5	If the array's last index was 499, how many elements does the array have?	

(*Note_language_neutral) This section is mostly language neutral

Section 5.2 - Arrays

Previously-introduced variables could each only store a single item. Just as people often maintain lists of items like a grocery list or a course roster, a programmer commonly needs to maintain a list of items. A construct known as an array can be used for this purpose. An ***array*** is an ordered list of items of a given data type. Each item in an array is called an ***element***.

Construct 5.2.1: Array reference variable declaration.

```
dataType[] identifier;
```

The [ ] symbols, called ***brackets***, indicate that the variable is an ***array reference***. An array reference variable can refer to an array of various sizes. That array must be explicitly allocated by the program using the ***new operator***. The new operator is used by the program to allocate an array using the following form:

Construct 5.2.2: Array allocation.

```
identifier = new type[numElements];
```

This statement creates space in memory to store the array with the specific number of elements, and assigns the array reference variable to refer to that newly allocated array.

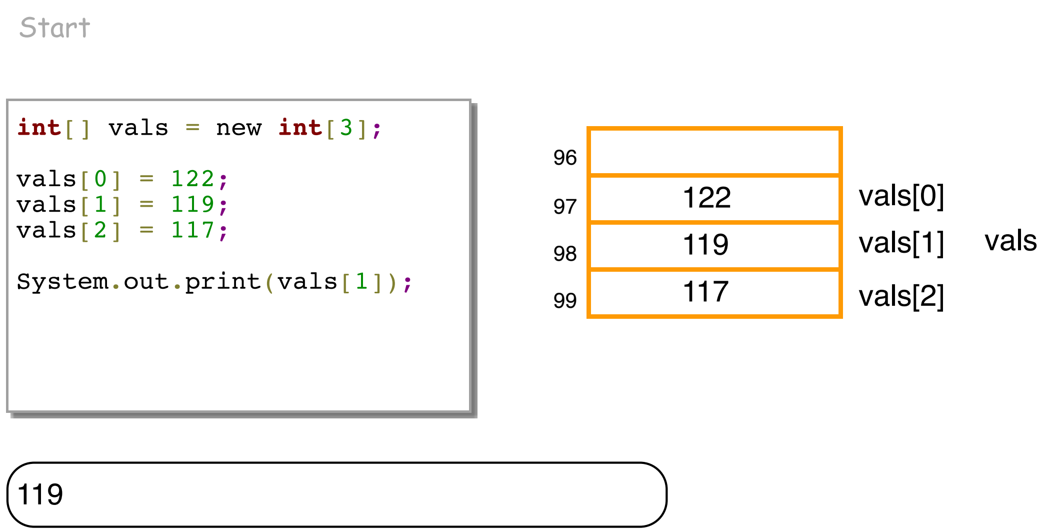
Similar to variable declaration, a good practice is to combine definition with initialization. For example, to define an array of 5 integers named myArray, a programmer can use the statement `int[] myArray = new int[5];`.

Terminology note: [ ] are ***brackets***, { } are ***braces***.

The following shows how to read and assign values within an array. The program creates a variable named vals with 3 elements, each of data type int. Those three elements are in fact each a separate variable that is accessed using the syntax vals[0], vals[1], and vals[2]. Note that the 3 elements are (some might say unfortunately) numbered 0 1 2 and not 1 2 3. In an array access, the number in brackets is called the ***index*** of the corresponding element.

P Participation Activity

5.2.1: An array definition creates multiple variables in memory, each accessible using [ ].



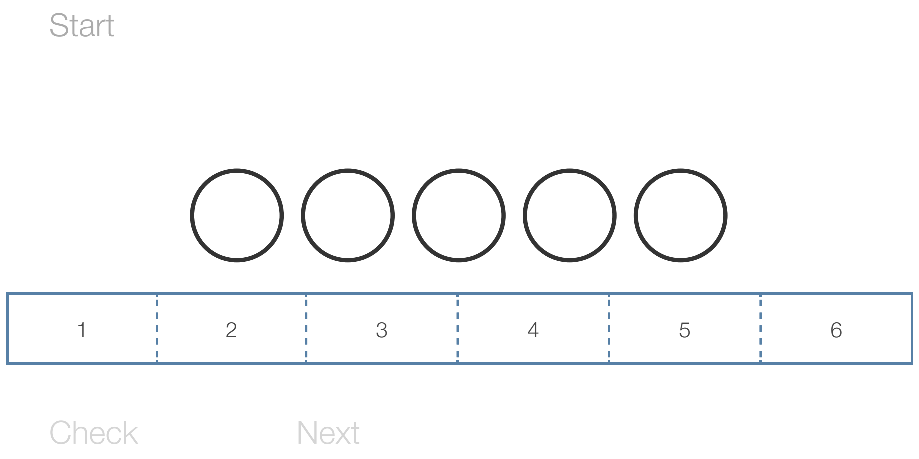
Start

```
int[] vals = new int[3];

vals[0] = 122;
vals[1] = 119;
vals[2] = 117;

System.out.print(vals[1]);
```

```
119
```

P Participation Activity

5.2.2: Select the index shown.



Participation Activity

5.2.3: Array basics.

Given:

```
int[] yearsArr = new int[4];

yearsArr[0] = 1999;
yearsArr[1] = 2012;
yearsArr[2] = 2025;
```

#	Question	Your answer
1	How many elements in memory does the array variable definition and initialization statement create?	0 1 3 4
2	What value is stored in yearsArr[1]?	1 1999 2012
3	What value does `curr = yearsArr[2]` assign to curr?	2 2025 Invalid index
4	What value does `curr = yearsArr[4]` assign to curr?	4 2025

		Invalid index
5	Recall that the array variable definition and initialization statement was `int[] yearsArr = new int[4]`. Is `curr = yearsArr[4]` a valid assignment?	Yes, it accesses the fourth element.
		No, yearsArr[4] does not exist.
6	What is the proper way to access the *first* element in array yearsArr?	yearsArr[1]
		yearsArr[0]
7	What are the contents of the array if the above code is followed by the statement: yearsArr[0] = yearsArr[2]?	1999, 2012, 1999, ?
		2012, 2012, 2025, ?
		2025, 2012, 2025, ?
8	What is the index of the *last* element for the following array: `int[] pricesArr = new int[100];`	99
		100
		101

Besides reducing the number of variables a programmer must define, a powerful aspect of arrays is that the index is an expression. Thus, an access could be written as userNums[i] where i is an int variable. As such, an array is useful to easily lookup the Nth item in a list. Consider the following program that allows a user to print the age of the Nth oldest known person to have ever lived.

Figure 5.2.1: Array's ith element can be directly accessed using [i]: Oldest people program.

```
import java.util.Scanner;

public class OldestPeople {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int[] oldestPeople = new int[5]; // Source: Wikipedia.org
      int nthPerson = 0;               // User input, Nth oldest p

      oldestPeople[0] = 122; // Died 1997 in France
      oldestPeople[1] = 119; // Died 1999 in U.S.
      oldestPeople[2] = 117; // Died 1993 in U.S.
      oldestPeople[3] = 117; // Died 1998 in Canada
      oldestPeople[4] = 116; // Died 2006 in Ecuador

      System.out.print("Enter N (1-5): ");
      nthPerson = scnr.nextInt();

      if ((nthPerson >= 1) && (nthPerson <= 5)) {
         System.out.print("The " + nthPerson + "th oldest person l
         System.out.println(oldestPeople[nthPerson - 1] + " years.
      }

      return;
   }
}
```

```
Enter N (1-5): 1
The 1th oldest p

...

Enter N (1-5): 4
The 4th oldest p

...

Enter N (1-5): 9

...

Enter N (1-5): 0

...

Enter N (1-5): 5
The 5th oldest p
```

The program can quickly access the Nth oldest person's age using `oldestPeople[nthPerson - 1]`. Note that the index is `nthPerson - 1` rather than just `nthPerson` because an array's indices start at 0, so the 1st age is at index 0, the 2nd at index 1, etc.

An array's index must be an integer type. The array index cannot be a floating-point type, even if the value is 0.0, 1.0, etc.

A key advantage of arrays becomes evident when used in conjunction with loops. To illustrate, the following program allows a user to enter 8 integer values, then prints those 8 values:

Figure 5.2.2: Arrays combined with loops are powerful together: User-entered numbers.

```
import java.util.Scanner;

public class ArrayPrinter {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;                 // Number of elements in array
      int[] userVals = new int[NUM_ELEMENTS]; // User numbers
      int i = 0;                                  // Loop index

      System.out.println("Enter " + NUM_ELEMENTS + " integer values..."
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print("Value: ");
         userVals[i] = scnr.nextInt();
      }

      System.out.print("You entered: ");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print(userVals[i] + " ");
      }
      System.out.println();

      return;
   }
}
```

```
Enter 8 int
Value: 5
Value: 99
Value: -1
Value: -44
Value: 8
Value: 5555
Value: 0
Value: 2
You entered
```

Consider how the program would have had to be written if using 8 separate variables. That program would have repeated variable definitions, output statements, and input statements. Now consider that program for NUM_ELEMENTS equal to 100, 1000, or more. With arrays and loops, the code would be the same as above. Only the constant literal 8 would be changed.

An array's elements are automatically initialized to default values when using the new operator to initialize the array reference. The default value for elements of integer and floating-point data types is zero, and the default value for boolean elements is false. For information on default values of other data types, see The Java Language Specification.

Initialization of the individual elements may be added to the array variable definition as shown below.

Construct 5.2.3: Additional array initialization.

```
type[] identifier = {val0, val1, ..., valN - 1};
```

Such initialization of the array elements does not require the use of the new operator, because the array's size is automatically set to the number of elements within the braces. For example, `int[] myArray = {1, 1, 1};` creates an array of three integer elements, each element initialized to 1. For larger arrays, initialization may be done by first defining the array, and then using a loop to fill the array.

P Participation Activity

5.2.4: Array definition and use.

#	Question	Your answer
1	Define and initialize an array named myVals that stores 10 elements of type int with default values.	
2	Assign the value stored at index 8 of array myVals to a variable x.	
3	Assign the value 555 to the element at index 2 of array myVals.	
4	Assign the value 777 to the second element of array myVals.	
5	Define an array of ints named myVals with 4 elements each initialized to 10.	

C Challenge Activity

5.2.1: Enter the output for the array.

Start

Enter the output of the following program.

```
public class arrayOutput {
   public static void main (String [] args) {
      final int NUM_ELEMENTS = 3;
      int [] userVals = new int[NUM_ELEMENTS];
      int i = 0;

      userVals[0] = 3;
      userVals[1] = 4;
      userVals[2] = 8;

      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.println(userVals[i]);
      }

      return;
   }
}
```

```
3
4
8
```

1	2	3	4	5

Check Next

C Challenge Activity

5.2.2: Printing array elements.

Write three statements to print the first three elements of array runTimes. Follow each statement with
775, 790, 805, 808}, print:

```
800
775
790
```

Note: These activities may test code with different test values. This activity will perform two tests, the
runTimes[5]), the second with a 4-element array (int runTimes[4]). See How to Use zyBooks.

Also note: If the submitted code tries to access an invalid array element, such as runTime[9] for a 5-e
strange results. Or the test may crash and report "Program end never reached", in which case the sy
caused the reported message.

```java
import java.util.Scanner;

public class PrintRunTimes {
   public static void main (String [] args) {
      int[] runTimes = new int[5];

      // Populate array
      runTimes[0] = 800;
      runTimes[1] = 775;
      runTimes[2] = 790;
      runTimes[3] = 805;
      runTimes[4] = 808;

      /* Your solution goes here  */

      return;
   }
}
```

Run

C Challenge Activity

5.2.3: Printing array elements with a for loop.

Write a for loop to print all elements in courseGrades, following each element with a space (including backwards. End each loop with a newline. Ex: If courseGrades = {7, 9, 11, 10}, print:

```
7 9 11 10
10 11 9 7
```

Hint: Use two for loops. Second loop starts with i = NUM_VALS - 1.

Note: These activities may test code with different test values. This activity will perform two tests, the courseGrades[4]), the second with a 2-element array (int courseGrades[2]). See How to Use zyBooks

Also note: If the submitted code tries to access an invalid array element, such as courseGrades[9] fo generate strange results. Or the test may crash and report "Program end never reached", in which ca case that caused the reported message.

```java
import java.util.Scanner;

public class CourseGradePrinter {
   public static void main (String [] args) {
      final int NUM_VALS = 4;
      int[] courseGrades = new int[NUM_VALS];
      int i = 0;

      courseGrades[0] = 7;
      courseGrades[1] = 9;
      courseGrades[2] = 11;
      courseGrades[3] = 10;

      /* Your solution goes here  */

      return;
   }
}
```

Run

Section 5.3 - Array iteration drill

The following activities can help one become comfortable with iterating through arrays or vectors, before learning to code such iteration.

P Participation Activity

5.3.1: Find the maximum value in the array.

Click "Store value" if a new maximum value is seen.

Start

Stored value

X	X	X	X	X	X	X

-1

Next value Store value

Time - Best time - Clear best

P Participation Activity

5.3.2: Negative value counting in array.

Click "Increment" if a negative value is seen.

Start

Counter

0

X	X	X	X	X	X	X

Next value Increment

Time - Best time - Clear best

P Participation Activity

5.3.3: Array sorting largest value.

Move the largest value to the right-most position. Click "Swap values" if the larger of the two current values is on the left.

Start

X	X	X	X	X	X	X

Next value Swap values

Time - Best time - Clear best

Section 5.4 - Iterating through arrays

Iterating through arrays using loops is commonplace and is an important programming skill to master.

Because array indices are numbered 0 to N - 1 rather than 1 to N, programmers commonly use this for loop structure:

Figure 5.4.1: Common for loop structure for iterating through an array.

```
// Iterating through myArray
for (i = 0; i < numElements; ++i) {
   // Loop body accessing myArray[i]
}
```

Note that index variable i is initialized to 0, and the loop expression is i < N rather than i <= N. If N were 5, the loop's iterations would set i to 0, 1, 2, 3, and 4, for a total of 5 iterations. The benefit of the loop structure is that each array element is accessed as myArray[i] rather than the more complex myArray[i - 1].

Programs commonly iterate through arrays to determine some quantity about the array's items. For example, the following program determines the maximum value in a user-entered list.

Figure 5.4.2: Iterating through an array example: Program that finds the max item.

```
import java.util.Scanner;

public class ArrayMax {

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;              // Number of elements
      int[] userVals = new int[NUM_ELEMENTS]; // Array of user numbe
      int i = 0;                               // Loop index
      int maxVal = 0;                          // Computed max

      // Prompt user to populate array
      System.out.println("Enter " + NUM_ELEMENTS + " integer values.

      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print("Value: ");
         userVals[i] = scnr.nextInt();
      }

      // Determine largest (max) number
      maxVal = userVals[0];                    // Largest so far

      for (i = 0; i < NUM_ELEMENTS; ++i) {
         if (userVals[i] > maxVal) {
            maxVal = userVals[i];
         }
      }
      System.out.println("Max: " + maxVal);

      return;
   }
}
```

```
Enter 8 intege
Value: 3
Value: 5
Value: 23
Value: -1
Value: 456
Value: 1
Value: 6
Value: 83
Max: 456

...

Enter 8 intege
Value: -5
Value: -10
Value: -44
Value: -2
Value: -27
Value: -9
Value: -27
Value: -9
Max: -2
```

If the user enters numbers 7, -9, 55, 44, 20, -400, 0, 2, then the program will output "max: 55". The bottom part of the code iterates through the array to determine the maximum value. The main idea of that code is to use a variable maxVal to store the largest value seen "so far" as the program iterates through the array. During each iteration, if the array's current element value is larger than the max seen so far, the program writes that value to maxVal (akin to being able to carry only one item as you walk through a store, replacing the current item by a better item whenever you see one). Before entering the loop, maxVal must be initialized to some value because max will be compared with each array element's value. A logical error would be to initialize maxVal to 0, because 0 is not in fact the largest value seen so far, and would result in incorrect output (of 0) if the user entered all negative numbers. Instead, the program peeks at an array element (using the first element, though any element could have been used) and initializes maxVal to that element's value.

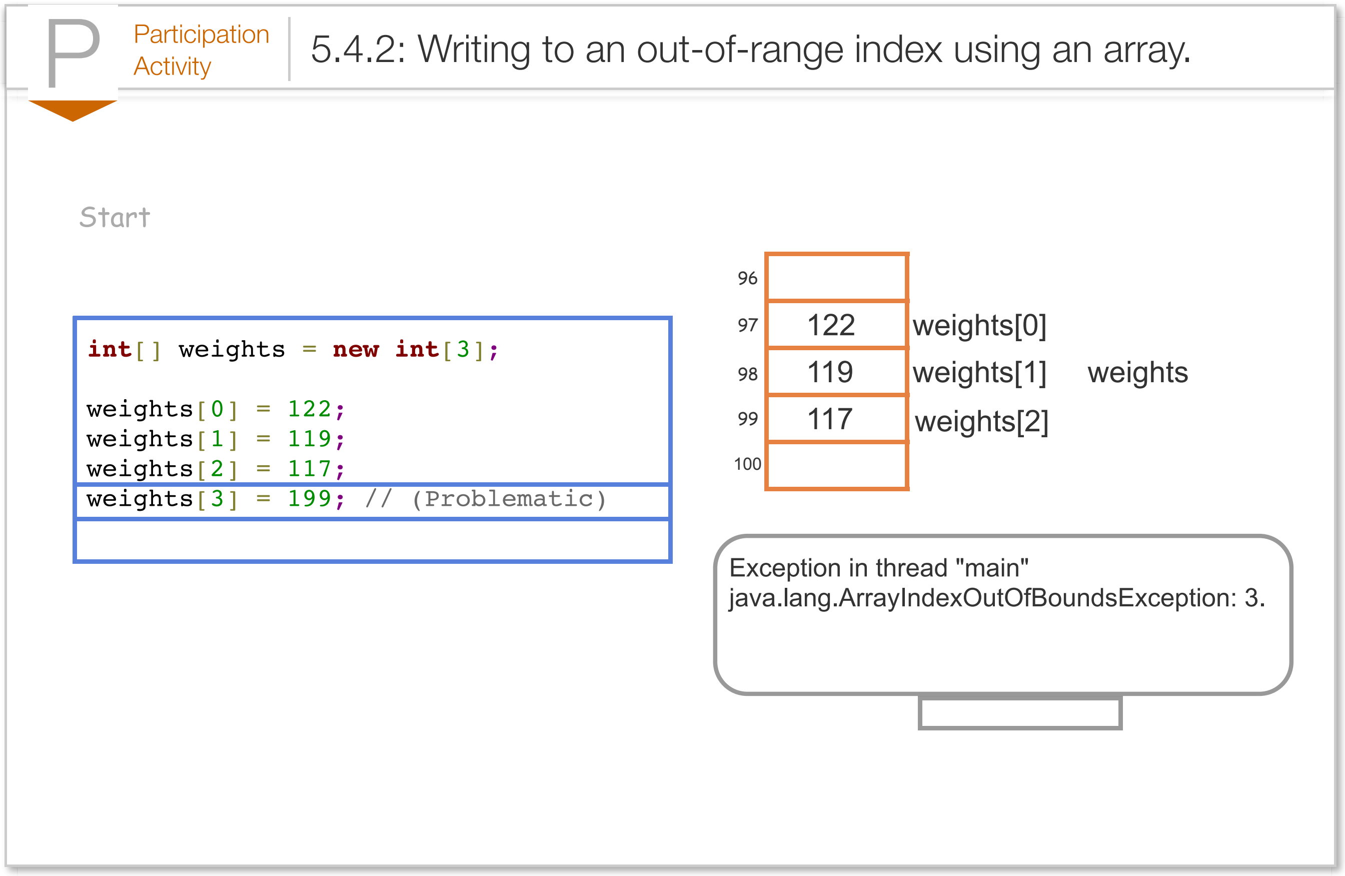
P Participation Activity

5.4.1: Array iteration.

Given an integer array myVals of size N_SIZE (i.e. int[] myVals = new int[N_SIZE]), complete the code to achieve the stated goal.

<table>
<tr><th>#</th><th>Question</th><th>Your answer</th></tr>
<tr><td>1</td><td>Determine the minimum number in the array, using the same initialization as the maximum number example above.</td><td><pre>
minVal = □;
for (i = 0; i < N_SIZE; ++i) {
   if (myVals[i] < minVal) {
      minVal = myVals[i];
   }
}
</pre></td></tr>
<tr><td>2</td><td>Count how many negative numbers exist in the array.</td><td><pre>
cntNeg = 0;
for (i = 0; i < N_SIZE; ++i) {
   if (□) {
     ++cntNeg;
   }
}
</pre></td></tr>
<tr><td>3</td><td>Count how many odd numbers exist in the array.</td><td><pre>
cntOdd = 0;
for (i = 0; i < N_SIZE; ++i) {
   if ((myVals[i] % 2) == 1) {□;
   }
}
</pre></td></tr>
</table>

A common error is to try to access an array with an index that is out of the array's index range, e.g., to try to access v[8] when v's valid indices are 0-7. Care should be taken whenever a user enters a number that is then used as an array index, and when using a loop index as an array index also, to ensure the index is within the array's valid index range. Checking whether an array index is in range is very important. Trying to access an array with an out-of-range index results in a runtime error that causes the program to terminate.



5.4.2: Writing to an out-of-range index using an array.

```
int[] weights = new int[3];

weights[0] = 122;
weights[1] = 119;
weights[2] = 117;
weights[3] = 199; // (Problematic)
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 3.
```

Iterating through an array for various purposes is an important programming skill to master. Here is another example, computing the sum of an array of int variables:

Figure 5.4.3: Iterating through an array example: Program that finds the sum of an array's elements.

```
import java.util.Scanner;

public class ArraySum {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;                // Number of elements
      int[] userVals = new int[NUM_ELEMENTS]; // User numbers
      int i = 0;                                 // Loop index
      int sumVal = 0;                            // For computing sum

      // Prompt user to populate array
      System.out.println("Enter " + NUM_ELEMENTS + " integer values.

      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print("Value: ");
         userVals[i] = scnr.nextInt();
      }

      // Determine sum
      sumVal = 0;

      for (i = 0; i < NUM_ELEMENTS; ++i) {
         sumVal = sumVal + userVals[i];
      }
      System.out.println("Sum: " + sumVal);

      return;
   }
}
```

```
Enter 8 intege
Value: 3
Value: 5
Value: 234
Value: 346
Value: 234
Value: 73
Value: 26
Value: -1
Sum: 920

...

Enter 8 intege
Value: 3
Value: 5
Value: 234
Value: 346
Value: 234
Value: 73
Value: 26
Value: 1
Sum: 922
```

Note that the code is somewhat different than the code computing the max. For computing the sum, the program initializes a variable sum to 0, then simply adds the current iteration's array element value to that sum.

P Participation Activity

5.4.3: Print the sum and average of an array's elements.

Modify the program to print the average as well as the sum. Hint: You don't actually have to change the loop, but rather change what you print.

```

import java.util.Scanner;

public class ArraySum {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;           // Number
      int[] userVals = new int[NUM_ELEMENTS]; // User nu
      int i = 0;                            // Loop in
      int sumVal = 0;                       // For com

      // Prompt user to populate array
      System.out.println("Enter " + NUM_ELEMENTS + " int

      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.println("Value: ");
         userVals[i] = scnr.nextInt();
      }
```

```
3 5 234 346 234 73 26 -1
```

Run

P Participation Activity

5.4.4: Print selected elements of an array.

Modify the program to instead just print each number that is greater than 21.

```

import java.util.Scanner;

public class ArraySum {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;           // Number
      int[] userVals = new int[NUM_ELEMENTS]; // User nu
      int i = 0;                            // Loop in
      int sumVal = 0;                       // For com

      // Prompt user to populate array
      System.out.println("Enter " + NUM_ELEMENTS + " int

      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.println("Value: ");
         userVals[i] = scnr.nextInt();
      }
```

```
3 5 234 346 234 73 26 -1
```

Run

C Challenge Activity

5.4.1: Enter the output for the array.

Start

Enter the output of the following program.

```
public class arrayOutput {
   public static void main (String [] args) {
      final int NUM_ELEMENTS = 3;
      int [] userVals = new int[NUM_ELEMENTS];
      int i = 0;

      userVals[0] = 3;
      userVals[1] = 5;
      userVals[2] = 9;

      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.println(userVals[i]);
      }

      return;
   }
}
```

```
3
5
9
```

1	2	3	4	5	6

Check Next

C Challenge Activity

5.4.2: Finding values in an array.

Set numMatches to the number of elements in userValues (having NUM_VALS elements) that equal r
userVals = {2, 2, 1, 2}, then numMatches = 3.

```
      int[] userValues = new int[NUM_VALS];
      int i = 0;
      int matchValue = 0;
      int numMatches = -99; // Assign numMatches with 0 before your for loop

      userValues[0] = 2;
      userValues[1] = 2;
      userValues[2] = 1;
      userValues[3] = 2;

      matchValue = 2;

      /* Your solution goes here  */

      System.out.println("matchValue: " + matchValue + ", numMatches: " +     numMat

      return;
   }
}
```

Run

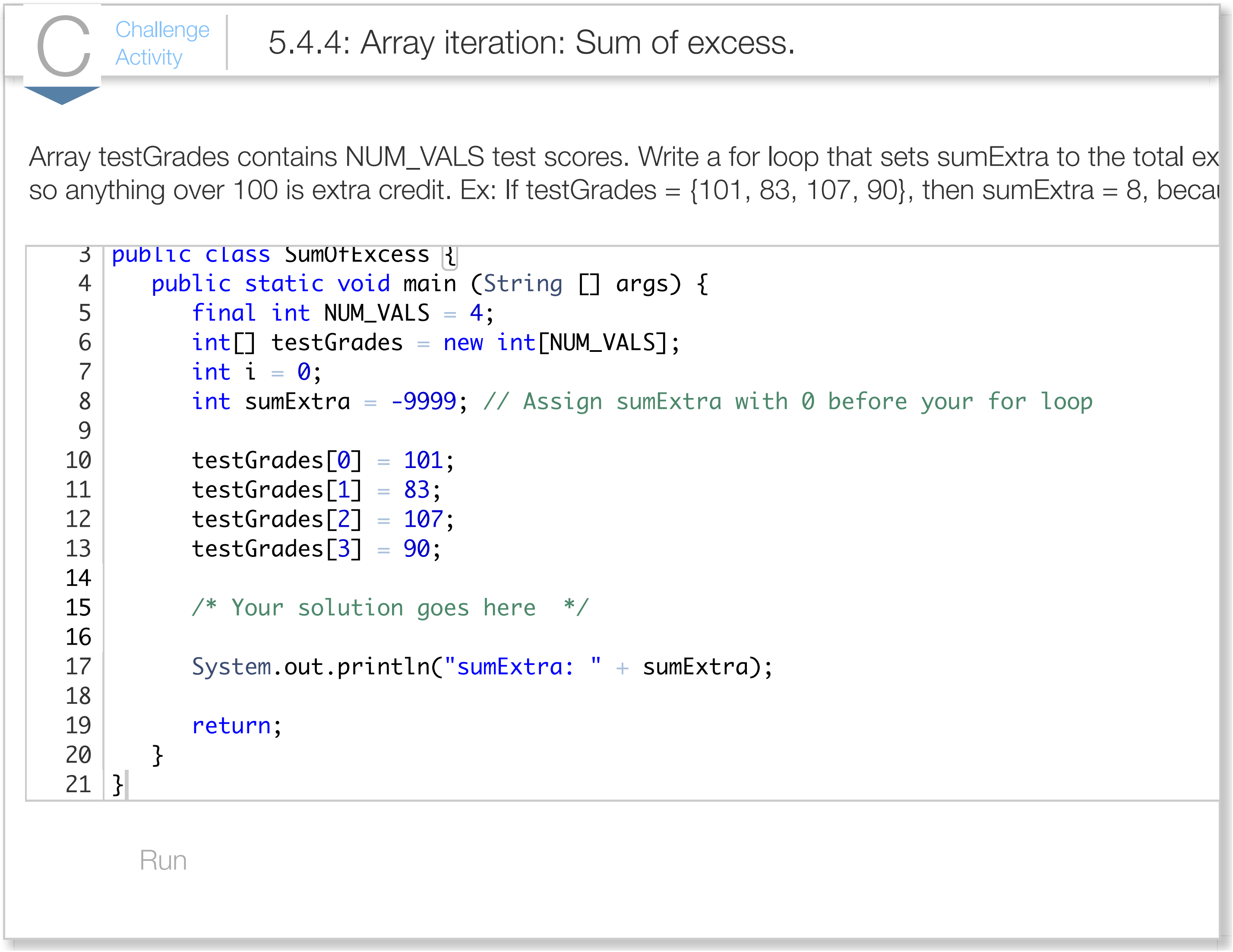
C Challenge Activity

5.4.3: Populating an array with a for loop.

Write a for loop to populate array userGuesses with NUM_GUESSES integers. Read integers using S
and user enters 9 5 2, then userGuesses is {9, 5, 2}.

```
import java.util.Scanner;

public class StoreGuesses {
   public static void main (String [] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_GUESSES = 3;
      int[] userGuesses = new int[NUM_GUESSES];
      int i = 0;

      /* Your solution goes here  */

      for (i = 0; i < NUM_GUESSES; ++i){
         System.out.print(userGuesses[i] + " ");
      }

      return;
   }
}
```

Run



5.4.4: Array iteration: Sum of excess.

Array testGrades contains NUM_VALS test scores. Write a for loop that sets sumExtra to the total ex
so anything over 100 is extra credit. Ex: If testGrades = {101, 83, 107, 90}, then sumExtra = 8, becau

```
public class SumOfExcess {
   public static void main (String [] args) {
      final int NUM_VALS = 4;
      int[] testGrades = new int[NUM_VALS];
      int i = 0;
      int sumExtra = -9999; // Assign sumExtra with 0 before your for loop

      testGrades[0] = 101;
      testGrades[1] = 83;
      testGrades[2] = 107;
      testGrades[3] = 90;

      /* Your solution goes here  */

      System.out.println("sumExtra: " + sumExtra);

      return;
   }
}
```

C Challenge Activity

5.4.5: Printing array elements separated by commas.

Write a for loop to print all NUM_VALS elements of array hourlyTemp. Separate elements with a comr {90, 92, 94, 95}, print:

```
90, 92, 94, 95
```

Note that the last element is not followed by a comma, space, or newline.

```

public class PrintWithComma {
   public static void main (String [] args) {
      final int NUM_VALS = 4;
      int[] hourlyTemp = new int[NUM_VALS];
      int i = 0;

      hourlyTemp[0] = 90;
      hourlyTemp[1] = 92;
      hourlyTemp[2] = 94;
      hourlyTemp[3] = 95;

      /* Your solution goes here  */

      System.out.println("");

      return;
   }
}
```

Run

Section 5.5 - Multiple arrays

Programmers commonly use multiple same-sized arrays to store related lists. For example, the following program maintains a list of letter weights in ounces, and another list indicating the corresponding postage cost for first class mail (usps.com).

Figure 5.5.1: Multiple array example: Letter postage cost program.

```
import java.util.Scanner;

public class PostageCalc {
   public static void main (String [] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 14;                       // Number of elements
      double[] letterWeights = new double[NUM_ELEMENTS]; // Weights in ounces
      int[] postageCosts = new int[NUM_ELEMENTS];        // Costs in cents (usps.com
      double userLetterWeight = 0.0;                     // Letter weight
      boolean foundWeight = false;                       // Found weight specified b
      int i = 0;                                         // Loop index

      // Populate letter weight/postage cost arrays
      letterWeights[i] = 1;    postageCosts[i] =  46; ++i;
      letterWeights[i] = 2;    postageCosts[i] =  66; ++i;
      letterWeights[i] = 3;    postageCosts[i] =  86; ++i;
      letterWeights[i] = 3.5;  postageCosts[i] = 106; ++i;
      letterWeights[i] = 4;    postageCosts[i] = 152; ++i;
      letterWeights[i] = 5;    postageCosts[i] = 172; ++i;
      letterWeights[i] = 6;    postageCosts[i] = 192; ++i;
      letterWeights[i] = 7;    postageCosts[i] = 212; ++i;
      letterWeights[i] = 8;    postageCosts[i] = 232; ++i;
      letterWeights[i] = 9;    postageCosts[i] = 252; ++i;
      letterWeights[i] = 10;   postageCosts[i] = 272; ++i;
      letterWeights[i] = 11;   postageCosts[i] = 292; ++i;
      letterWeights[i] = 12;   postageCosts[i] = 312; ++i;
      letterWeights[i] = 13;   postageCosts[i] = 332; ++i;

      // Prompt user to enter letter weight
      System.out.print("Enter letter weight (in ounces): ");
      userLetterWeight = scnr.nextDouble();

      // Postage costs is based on smallest letter weight greater than
      // or equal to mailing letter weight
      foundWeight = false;

      for (i = 0; (i < NUM_ELEMENTS) && (!foundWeight); ++i) {
         if( userLetterWeight <= letterWeights[i] ) {
            foundWeight = true;
            System.out.print("Postage for USPS first class mail is ");
            System.out.print(postageCosts[i]);
            System.out.println(" cents");
         }
      }

      if( !foundWeight ) {
         System.out.println("Letter is too heavy for USPS " +
                            "first class mail.");
      }

      return;
   }
}
```

```
Enter letter weight (in ounces): 3
Postage for USPS first class mail is 86 cents

...

Enter letter weight (in ounces): 9.5
Postage for USPS first class mail is 272 cents
```

```
...

Enter letter weight (in ounces): 15
Letter is too heavy for USPS first class mail.
```

Notice how the `if (userLetterWeight <= letterWeights[i])` statement compares the user-entered letter weight with the current element in the letterWeights array. If the entered weight is less than or equal to the current element in the letterWeights array, the program prints the element in postageCosts having that same index.

The loop's expression `(i < NUM_ELEMENTS) && (!foundWeight)` depends on the value of the variable foundWeight. This expression prevents the loop from iterating through the entire array once the correct letter weight has been found. Omitting the check for found from the loop expression would result in an incorrect output; the program would incorrectly print the postage cost for all letter weights greater than the user's letter weight.

Note that the array initialization uses [i] rather than [0], [1], etc. Such a technique is less prone to errors, and enables easy reordering or inserting of new letter weights and postage costs.

P Participation Activity

5.5.1: Multiple arrays in the above postage cost program.

#	Question	Your answer
1	letterWeights[0] is 1, meaning element 0 of letterWeights and postageCosts correspond to a weight of 1 ounce.	True
		False
2	postageCosts[2] represents the cost for a weight of 2 ounces.	True
		False
3	The program fails to provide a cost for a weight of 7.5.	True
		False

P Participation Activity

5.5.2: Postage calculation with negative weight error message.

Improve the program by also outputting "The next higher weight is ___ with a cost of ___ cents".

```

import java.util.Scanner;

public class PostageCalc {
   public static void main (String [] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 14;
      double[] letterWeights = new double[NUM_ELEMENTS];
      int[] postageCosts = new int[NUM_ELEMENTS];
      double userLetterWeight = 0.0;
      boolean foundWeight = false;
      int i = 0;

      // Populate letter weight/postage cost arrays
      letterWeights[i] = 1;    postageCosts[i] =  46; ++
      letterWeights[i] = 2;    postageCosts[i] =  66; ++
      letterWeights[i] = 3;    postageCosts[i] =  86; ++
      letterWeights[i] = 3.5;  postageCosts[i] = 106; ++
      letterWeights[i] = 4;    postageCosts[i] = 152; ++
```

```
3
```

Run

P Participation Activity

5.5.3: Multiple arrays.

#	Question	Your answer
1	Using two separate statements, define two related integer arrays named seatPosition and testScore (in that order) each with 130 elements.	
2	How many total elements are stored within array familyAges with 50 elements, and array familyHeights with 50 elements?	

C Challenge Activity

5.5.1: Printing the sum of two array elements.

Add each element in origList with the corresponding value in offsetAmount. Print each sum followed
Ex: If origList = {40, 50, 60, 70} and offsetAmount = {5, 7, 3, 0}, print:

```
45 57 63 70
```

```
      int i = 0;

      origList[0] = 40;
      origList[1] = 50;
      origList[2] = 60;
      origList[3] = 70;

      offsetAmount[0] = 5;
      offsetAmount[1] = 7;
      offsetAmount[2] = 3;
      offsetAmount[3] = 0;

      /* Your solution goes here  */

      System.out.println("");

      return;
   }
}
```

Run

C Challenge Activity

5.5.2: Multiple arrays: Key and value.

For any element in keysList with a value greater than 100, print the corresponding value in itemsList,
{42, 105, 101, 100} and itemsList = {10, 20, 30, 40}, print:

```
20 30
```

Since keysList[1] and keysList[2] have values greater than 100, the value of itemsList[1] and itemsList

```
      int i = 0;

      keysList[0] = 42;
      keysList[1] = 105;
      keysList[2] = 101;
      keysList[3] = 100;

      itemsList[0] = 10;
      itemsList[1] = 20;
      itemsList[2] = 30;
      itemsList[3] = 40;

      /* Your solution goes here  */

      System.out.println("");

      return;
   }
}
```

Run

Section 5.6 - Swapping two variables

Note_language_neutral2

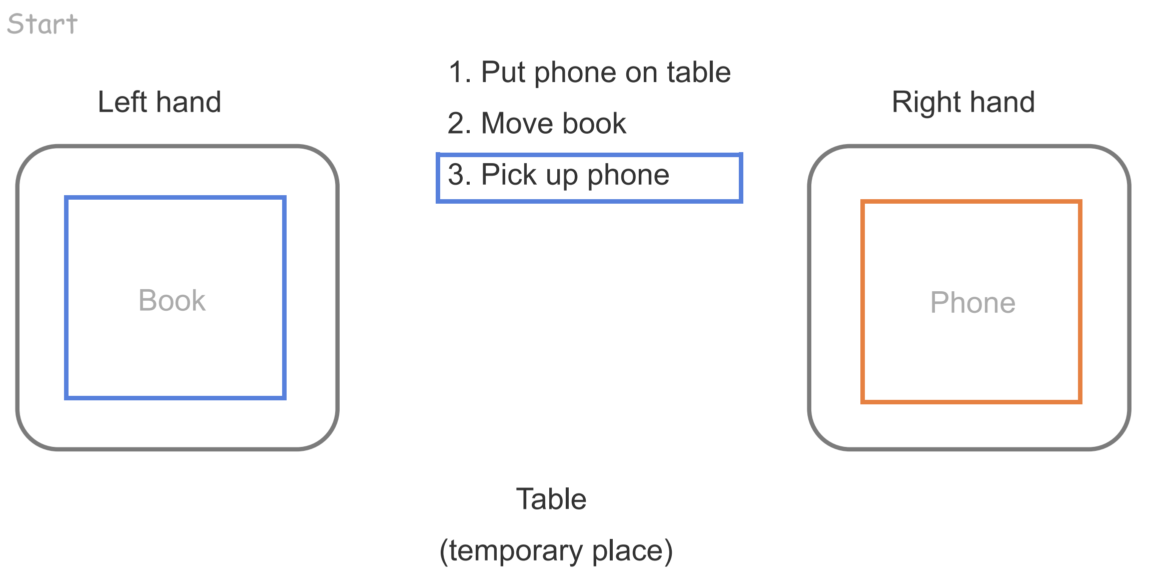
Sometimes a program must swap values among two variables. ***Swapping*** two variables x and y means to assign y's value to x, and x's value to y. If x is 33 and y is 55, then after swapping x is 55 and y is 33.

Swapping requires a temporary third variable. To understand the intuition of such temporary storage, consider a person holding a book in one hand and a phone in the other, wishing to swap the items.

The person can temporarily place the phone on a table, move the book to the other hand, then pick up the phone.

P Participation Activity

5.6.1: Swap idea: Use a temporary location.

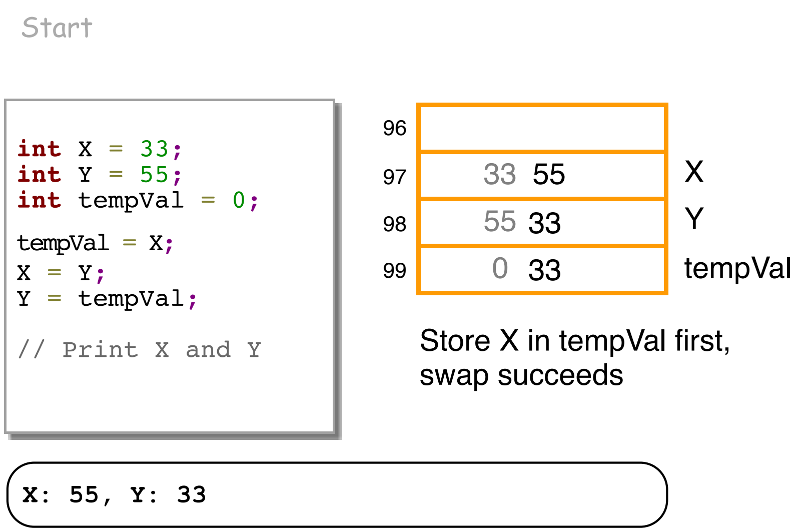


1. Put phone on table
2. Move book
3. Pick up phone

Similarly, swapping two variables uses a third variable to temporarily hold one value while the other value is copied over.

P Participation Activity

5.6.2: Swapping two variables requires a third temporary variable.



```
int X = 33;
int Y = 55;
int tempVal = 0;

tempVal = X;
X = Y;
Y = tempVal;

// Print X and Y
```

Store X in tempVal first, swap succeeds

```
X: 55, Y: 33
```

P Participation Activity

5.6.3: Swap.

Given x = 22 and y = 99. What are x and y after the given code?

#	Question	Your answer

1	`x = y;`
	`y = x;`
		x is 99 and y is 22.

		x is 22 and y is 99.
		x is 99 and y is 99.

2	`x = y;`
	`y = x;`
	`x = y;`
		x is 99 and y is 22.

		x is 99 and y is 99.
		x is 22 and y is 22.

3	`tempVal = x;`
	`x = y;`
	`y = x;`
		x is 99 and y is 22.

		x is 99 and y is 99.

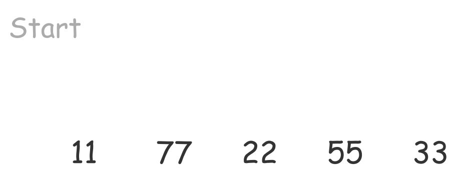
4	`tempVal = x;`
	`x = y;`
	`y = tempVal;`
		x is 99 and y is 22.

		x is 99 and y is 99.

If you have studied arrays or vectors (or other kinds of lists), know that most swaps are actually performed between two list elements. For example, reversing a list with N elements can be achieved by swapping element 1 and N, element 2 and N-1, element 3 and N-2, etc. (stopping at the middle of the list).

P Participation Activity

5.6.4: Reversing a list using swaps.



P Participation Activity

5.6.5: Reversing a list using swaps.

#	Question	Your answer
1	Using the above approach, how many swaps are needed to reverse this list: 999 888 777 666 555 444 333 222	

(*Note_language_neutral2) This section is mostly language neutral

Section 5.7 - Loop-modifying or copying/comparing arrays

Sometimes a program changes some elements' values or moves elements while iterating through a array. The following uses a loop to convert any negative array element values to 0.

Figure 5.7.1: Modifying an array during iteration example: Converting negatives to 0 program.

```
import java.util.Scanner;

public class NegativeToZero {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;                // Number of elements
      int[] userVals = new int[NUM_ELEMENTS]; // User numbers
      int i = 0;                                 // Loop index

      // Prompt user to input values
      System.out.println("Enter " + NUM_ELEMENTS + " integer values.");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print("Value: ");
         userVals[i] = scnr.nextInt();
      }

      // Convert negatives to 0
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         if (userVals[i] < 0) {
            userVals[i] = 0;
         }
      }

      // Print numbers
      System.out.print("New numbers: ");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print(userVals[i] + " ");
      }

      return;
   }
}
```

```
Enter 8 intege
Value: 5
Value: 67
Value: -5
Value: -4
Value: 5
Value: 6
Value: 6
Value: 4
New numbers: 5
```

P Participation Activity

5.7.1: Modifying an array in a loop.

What is the resulting array contents, assuming each question starts with an array of size 4 having contents -55, -1, 0, 9?

#	Question	Your answer

1	`for (i = 0; i < 4; ++i) {`
	`   itemsList[i] = i;`
	`}`
		-54, 0, 1, 10
		0, 1, 2, 3
		1, 2, 3, 4

		1, 2, 3, 4

2	`for (i = 0; i < 4; ++i) {`
	`   if (itemsList[i] < 0) {`
	`      itemsList[i] = itemsList[i] * -1;`
	`   }`
	`}`
		-55, -1, 0, -9
		55, 1, 0, -9
		55, 1, 0, 9

3	`for (i = 0; i < 4; ++i) {`
	`   itemsList[i] = itemsList[i+1];`
	`}`
		-1, 0, 9, 0
		0, -55, -1, 0
		Error

4	`for (i = 0; i < 3 ; ++i) {`
	`   itemsList[i] = itemsList[i+1];`
	`}`
		-1, 0, 9, 9
		Error
		-1, 0, 9, 0

5	`for (i = 0; i < 3 ; ++i) {`
	`   itemsList[i+1] = itemsList[i];`
	`}`
		-55, -55, -55, -55
		0, -55, -1, 0
		Error

P Participation Activity

5.7.2: Modifying an array during iteration example: Doubling element values.

Complete the following program to double each number in the array.

```

import java.util.Scanner;

public class NegativeToZero {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;           // Number
      int[] userVals = new int[NUM_ELEMENTS]; // User nu
      int i = 0;                            // Loop in

      // Prompt user to input values
      System.out.println("Enter " + NUM_ELEMENTS + " int
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.println("Value: ");
         userVals[i] = scnr.nextInt();
      }

      // Double each element. FIXME write this loop
```

```
5 67 -5 -4 5 6 6 4
```

Run

Copying an array is a common task. Given a second array of the same size, a loop can copy each element one-by-one. Modifications to either array do not affect the other.

Figure 5.7.2: Array copying: Converting negatives to 0 program.

```java
import java.util.Scanner;

public class NegativeToZeroCopy {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;               // Number of elements
      int[] userVals = new int[NUM_ELEMENTS];   // User numbers
      int[] copiedVals = new int[NUM_ELEMENTS]; // New numbers
      int i = 0;                                // Loop index

      // Prompt user for input values
      System.out.println("Enter " + NUM_ELEMENTS + " integer values.
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print("Value: ");
         userVals[i] = scnr.nextInt();
      }

      // Convert nums to newNums
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         copiedVals[i] = userVals[i];
      }

      // Convert negatives to 0
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         if (copiedVals[i] < 0) {
            copiedVals[i] = 0;
         }
      }

      // Print numbers
      System.out.println("\nOriginal and new values: ");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.println(userVals[i] + " became " + copiedVals[i]);
      }
      System.out.println();

      return;
   }
}
```

```
Enter 8 intege
Value: 12
Value: -5
Value: 34
Value: 75
Value: -14
Value: 33
Value: 12
Value: -104

Original and ne
12 became 12
-5 became 0
34 became 34
75 became 75
-14 became 0
33 became 33
12 became 12
-104 became 0
```

P Participation Activity

5.7.3: Array copying.

Given array firstList with size 4 and element values, 33, 44, 55, 66, and array secondList with size 4 and elements values 0, 0, 0, 0.

#	Question	Your answer
1	firstList = secondList copies 0s into each firstList element.	True False

2	This loop copies firstList to secondList, so that secondList becomes 33, 44, 55, 66:
	`for (i = 0; i < 4; ++i) {`
	`   secondList[i] = firstList[i];`
	`}`
		True
		False

3	After a for loop copies firstList to secondList, the assignment secondList[0] = 99 will modify both arrays.	True False

4	Given thirdList with size 5 and elements 22, 21, 20, 19, 18, the following causes firstList's values to be 22, 21, 20, 19, 18:
	`for (i = 0; i < 5; ++i) {`
	`   firstList[i] = thirdList[i];`
	`}`
		True
		False

C Challenge Activity

5.7.1: Decrement array elements.

Write a loop that subtracts 1 from each element in lowerScores. If the element was already 0 or nega
lowerScores = {5, 0, 2, -3} becomes {4, 0, 1, 0}.

```
      final int SCORES_SIZE = 4;
      int[] lowerScores = new int[SCORES_SIZE];
      int i = 0;

      lowerScores[0] = 5;
      lowerScores[1] = 0;
      lowerScores[2] = 2;
      lowerScores[3] = -3;

      /* Your solution goes here  */

      for (i = 0; i < SCORES_SIZE; ++i) {
         System.out.print(lowerScores[i] + " ");
      }
      System.out.println();

      return;
   }
}
```

Run

Challenge Activity

5.7.2: Copy and modify array elements.

Write a loop that sets newScores to oldScores shifted once left, with element 0 copied to the end. Ex newScores = {20, 30, 40, 10}.

Note: These activities may test code with different test values. This activity will perform two tests, the (newScores = {10, 20, 30, 40}), the second with a 1-element array (newScores = {199}). See How to

Also note: If the submitted code tries to access an invalid array element, such as newScores[9] for a generate strange results. Or the test may crash and report "Program end never reached", in which ca case that caused the reported message.

```
      int[] oldScores = new int[SCORES_SIZE];
      int[] newScores = new int[SCORES_SIZE];
      int i = 0;

      oldScores[0] = 10;
      oldScores[1] = 20;
      oldScores[2] = 30;
      oldScores[3] = 40;

      /* Your solution goes here  */

      for (i = 0; i < SCORES_SIZE; ++i) {
         System.out.print(newScores[i] + " ");
      }
      System.out.println();

      return;
   }
}
```

Run

C Challenge Activity

5.7.3: Modify array elements using other elements.

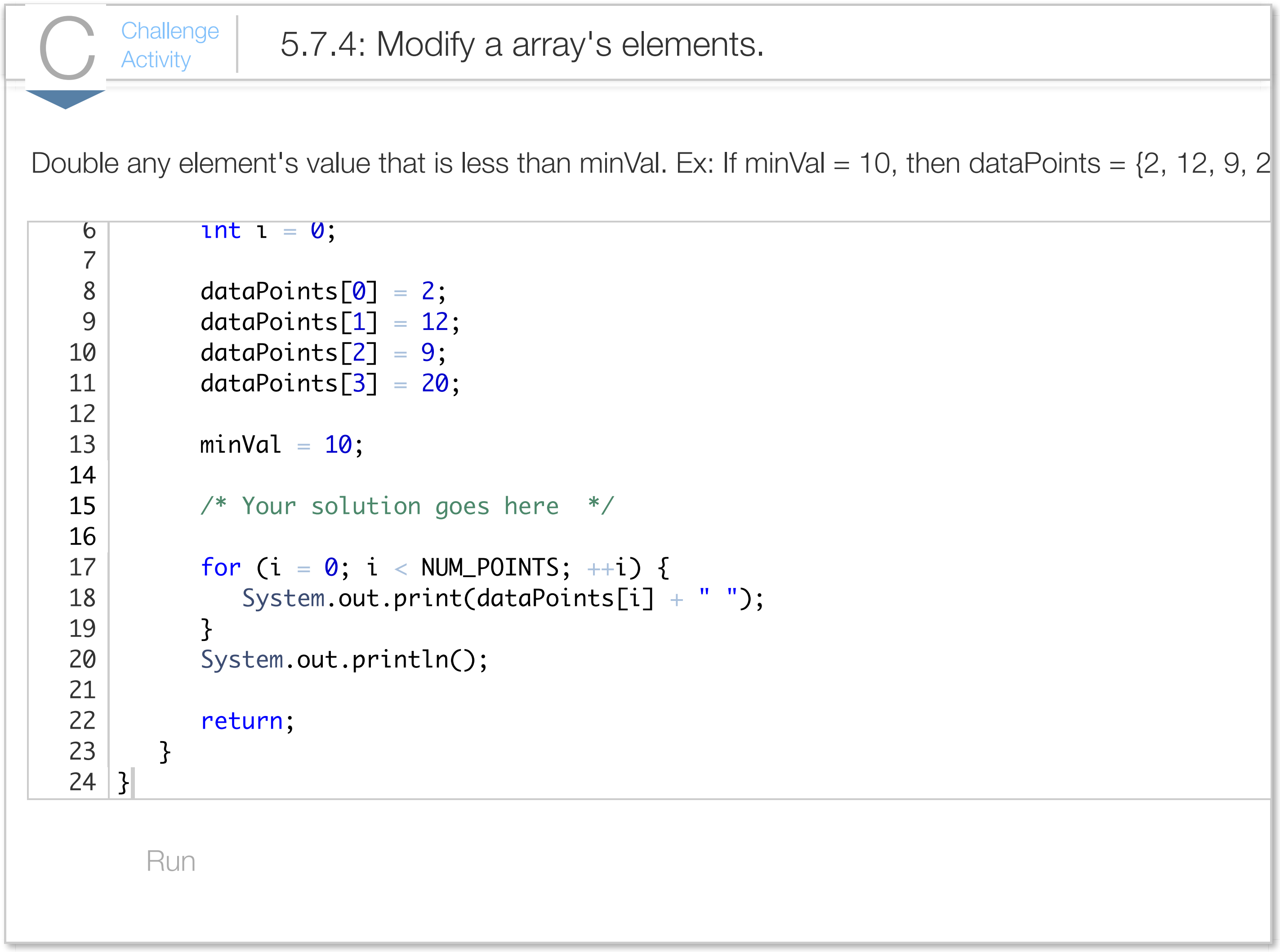
Write a loop that sets each array element to the sum of itself and the next element, except for the las careful not to index beyond the last element. Ex:

```
Initial scores:         10, 20, 30, 40
Scores after the loop: 30, 50, 70, 40
```

The first element is 30 or 10 + 20, the second element is 50 or 20 + 30, and the third element is 70 c the same.

```
      final int SCORES_SIZE = 4;
      int[] bonusScores = new int[SCORES_SIZE];
      int i = 0;

      bonusScores[0] = 10;
      bonusScores[1] = 20;
      bonusScores[2] = 30;
      bonusScores[3] = 40;

      /* Your solution goes here  */

      for (i = 0; i < SCORES_SIZE; ++i) {
         System.out.print(bonusScores[i] + " ");
      }
      System.out.println();

      return;
   }
}
```

Run



5.7.4: Modify a array's elements.

Double any element's value that is less than minVal. Ex: If minVal = 10, then dataPoints = {2, 12, 9, 2

```
      int i = 0;

      dataPoints[0] = 2;
      dataPoints[1] = 12;
      dataPoints[2] = 9;
      dataPoints[3] = 20;

      minVal = 10;

      /* Your solution goes here  */

      for (i = 0; i < NUM_POINTS; ++i) {
         System.out.print(dataPoints[i] + " ");
      }
      System.out.println();

      return;
   }
}
```

Section 5.8 - Debugging example: Reversing an array

A common array modification is to reverse an array's elements. One way to accomplish this goal is to perform a series of swaps. For example, starting with an array of numbers 10 20 30 40 50 60 70 80, we could first swap the first item with the last item, yielding 80 20 30 40 50 60 70 10. We could next swap the second item with the second-to-last item, yielding 80 70 30 40 50 60 20 10. The next swap would yield 80 70 60 40 50 30 20 10, and the last would yield 80 70 60 50 40 30 20 10.

With this basic idea of how to reverse an array, we can attempt to write a program to carry out such reversal. Below we develop such a program but we make common mistakes along the way, to aid learning from examples of what not to do.

A first attempt to write a program that reverses an array appears below:

Figure 5.8.1: First program attempt to reverse array: Invalid access out of array bounds.

```
import java.util.Scanner;

public class ArrayReverseElem {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;              // Number of elements
      int[] userVals = new int[NUM_ELEMENTS]; // User numbers
      int i = 0;                               // Loop index

      // Prompt user to input values
      System.out.println("Enter " + NUM_ELEMENTS
              + " integer values...");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print("Value: ");
         userVals[i] = scnr.nextInt();
      }

      // Reverse array's elements
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         userVals[i] = userVals[NUM_ELEMENTS - i]; // Swap
      }

      // Print numbers
      System.out.print("\nNew values: ");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print(userVals[i] + " ");
      }

      return;
   }
}
```

```
Enter 8 integer valu
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80
Exception in thread
        at javaappli
```

Something went wrong: The program did not reverse the array, and an array-index-out-of-bounds exception occurred. Let's try to find the code that caused the problem.

The first and third for loops are fairly standard, so let's initially focus attention on the middle for loop that does the reversing. The swap statement inside that loop is `userNums[i] = userNums[NUM_ELEMENTS - i]`. When i is 0, the statement will execute `userNums[0] = userNums[8];`. However, userNums has size 8 and thus valid indices are 0..7. userNums[8] does not exist. The program should actually swap elements 0 and 7, then 1 and 6, etc. Thus, let's change the right-side index to `NUM_VALUES - 1 - i`. The revised program is shown below.

Figure 5.8.2: Next program attempt to reverse an array: Doesn't reverse properly; we forgot to swap.

```
import java.util.Scanner;

public class ArrayReverseElem {

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;                // Number of elements
      int[] userVals = new int[NUM_ELEMENTS]; // User numbers
      int i = 0;                                 // Loop index

      // Prompt user to input values
      System.out.println("Enter " + NUM_ELEMENTS
              + " integer values...");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print("Value: ");
         userVals[i] = scnr.nextInt();
      }

      // Reverse array's elements
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         userVals[i] = userVals[NUM_ELEMENTS - 1 - i]; // Swap
      }

      // Print numbers
      System.out.print("\nNew values: ");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print(userVals[i] + " ");
      }

      return;
   }
}
```

```
Enter 8 integer valu
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values: 80 70 6
```

The last four elements are still wrong. To determine what went wrong, we can manually (i.e., on paper) trace the loop's execution.

- i is 0: userNums[0] = userNums[7]. Array now: 80 20 30 40 50 60 70 80.
- i is 1: userNums[1] = userNums[6]. Array now: 80 70 30 40 50 60 70 80.
- i is 2: userNums[2] = userNums[5]. Array now: 80 70 60 40 50 60 70 80.
- i is 3: userNums[3] = userNums[4]. Array now: 80 70 60 50 50 60 70 80.
- i is 4: userNums[4] = userNums[3]. Array now: 80 70 60 50 50 60 70 80. *Uh-oh, where did 40 go?*

We failed to actually swap the array elements, instead the code just copies values in one direction. We need to add code to properly swap. We add a variable tempVal to temporarily hold `userNums[NUM_VALUES - 1 - i]` so we don't lose that element's value.

Figure 5.8.3: Program with proper swap: However, the program's output shows the array doesn't change.

```
import java.util.Scanner;

public class ArrayReverseElem {

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;               // Number of elements
      int[] userVals = new int[NUM_ELEMENTS]; // User numbers
      int i = 0;                                // Loop index
      int tempVal = 0;             // Temp variable for swapping

      // Prompt user to input values
      System.out.println("Enter " + NUM_ELEMENTS
              + " integer values...");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print("Value: ");
         userVals[i] = scnr.nextInt();
      }

      // Reverse array's elements
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         tempVal = userVals[i];                        // Temp for swap
         userVals[i] = userVals[NUM_ELEMENTS - 1 - i]; // First part of swap
         userVals[NUM_ELEMENTS - 1 - i] = tempVal;     // Swap complete
      }

      // Print numbers
      System.out.print("\nNew values: ");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print(userVals[i] + " ");
      }

      return;
   }
}
```

```
Enter
Value
Value
Value
Value
Value
Value
Value
Value

New va
```

The new values are not reversed. Again, let's manually trace the loop iterations.

- i is 0: userNums[0] = userNums[7]. Array now: 80 20 30 40 50 60 70 10.
- i is 1: userNums[1] = userNums[6]. Array now: 80 70 30 40 50 60 20 10.
- i is 2: userNums[2] = userNums[5]. Array now: 80 70 60 40 50 30 20 10.
- i is 3: userNums[3] = userNums[4]. Array now: 80 70 60 50 40 30 20 10. *Looks reversed.*
- i is 4: userNums[4] = userNums[3]. Array now: 80 70 60 40 50 30 20 10. *Why are we*

still swapping?

Tracing makes clear that the for loop should not iterate over the entire array. The reversal is completed halfway through the iterations. The solution is to set the loop expression to `i < (NUM_VALUES / 2)`.

Figure 5.8.4: Program with correct loop bound: Running the program yields the correct output.

```
import java.util.Scanner;

public class ArrayReverseElem {

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int NUM_ELEMENTS = 8;             // Number of elements
      int[] userVals = new int[NUM_ELEMENTS]; // User numbers
      int i = 0;                              // Loop index
      int tempVal = 0;            // Temp variable for swapping

      // Prompt user to input values
      System.out.println("Enter " + NUM_ELEMENTS
              + " integer values...");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print("Value: ");
         userVals[i] = scnr.nextInt();
      }

      // Reverse array's elements
      for (i = 0; i < (NUM_ELEMENTS / 2); ++i) {
         tempVal = userVals[i];                        // Temp for swap
         userVals[i] = userVals[NUM_ELEMENTS - 1 - i]; // First part of swap
         userVals[NUM_ELEMENTS - 1 - i] = tempVal;     // Swap complete
      }

      // Print numbers
      System.out.print("\nNew values: ");
      for (i = 0; i < NUM_ELEMENTS; ++i) {
         System.out.print(userVals[i] + " ");
      }

      return;
   }
}
```

```
Enter
Value
Value
Value
Value
Value
Value
Value
Value

New va
```

We should ensure the program works if the number of elements is odd rather than even. Suppose the array has 5 elements (0-4) with values 10 20 30 40 50. NUM_VALUES / 2 would be 5 / 2 = 2, meaning the loop expression would be i < 2. The iteration when i is 0 would swap elements 0 and 4 (5-1-0), yielding 50 20 30 40 10. The iteration for i=1 would swap elements 1 and 3, yielding 50 40 30 20 10. The loop would then not execute again because i is 2. So the results are correct for an odd number of elements, because the middle element will just not move.

The mistakes made above are each very common when dealing with loops and arrays, especially for beginning programmers. An incorrect (in this case out-of-range) index, an incorrect swap, and an incorrect loop expression. The lesson is that loops and arrays require attention to detail, greatly aided by manually executing the loop to determine what is happening on each iteration. Ideally, a programmer will take more care when writing the original program, but the above mistakes are quite common.

P Participation Activity

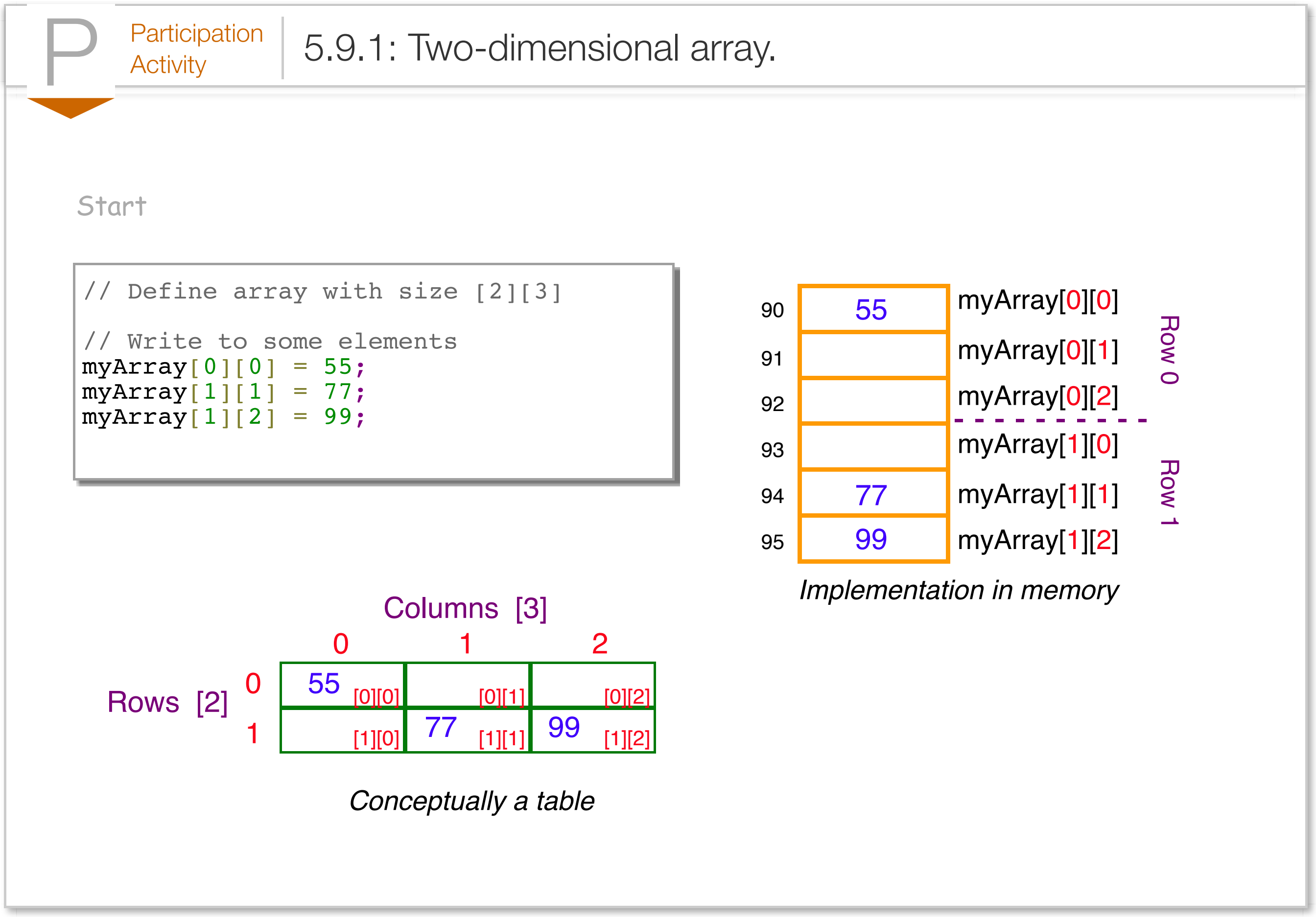
5.8.1: Array reversal example.

Questions refer to the problematic example in this section.

#	Question	Your answer
1	The first problem was trying to access a non-existent element.	True False
2	The second problem was failing to properly swap, using just this statement: userNums[i] = userNums[NUM_ELEMENTS - 1 - i]; // Swap	True False
3	The third problem was that the loop did not iterate over all the elements, but rather stopped one short.	True False
4	The programmer probably should have been more careful in creating the first version of the program.	True False

Section 5.9 - Two-dimensional arrays

An array can be defined with two dimensions. `int[][] myArray = new int[R][C]` represents a table of int variables with R rows and C columns, so R*C elements total. For example, `int[][] myArray = new int[2][3]` creates a table with 2 rows and 3 columns, for 6 int variables total. Example accesses are `myArray[0][0] = 33;` or `num = myArray[1][2]`.



P Participation Activity

5.9.1: Two-dimensional array.

Start

```
// Define array with size [2][3]

// Write to some elements
myArray[0][0] = 55;
myArray[1][1] = 77;
myArray[1][2] = 99;
```

Conceptually, a two-dimensional array is a table with rows and columns. The compiler maps two-dimensional array elements to one-dimensional memory, each row following the previous row, known as ***row-major order***.

Figure 5.9.1: Using a two-dimensional array: A driving distance between cities example.

```
import java.util.Scanner;

/* Direct driving distances between cities, in miles */
/* 0: Boston  1: Chicago  2: Los Angeles */
public class CityDist {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int cityA = 0;                                // Starting city
      int cityB = 0;                                // Destination city
      int [][] DrivingDistances = new int[3][3]; // Driving dista

      // Initialize distances array
      DrivingDistances[0][0] = 0;
      DrivingDistances[0][1] = 960;  // Boston-Chicago
      DrivingDistances[0][2] = 2960; // Boston-Los Angeles
      DrivingDistances[1][0] = 960;  // Chicago-Boston
      DrivingDistances[1][1] = 0;
      DrivingDistances[1][2] = 2011; // Chicago-Los Angeles
      DrivingDistances[2][0] = 2960; // Los Angeles-Boston
      DrivingDistances[2][1] = 2011; // Los Angeles-Chicago
      DrivingDistances[2][2] = 0;

      System.out.println("0: Boston 1: Chicago  2: Los Angeles");

      System.out.print("Enter city pair (Ex: 1 2) -- ");
      cityA = scnr.nextInt();
      cityB = scnr.nextInt();

      System.out.print("Distance: " + DrivingDistances[cityA][cityB]);
      System.out.println(" miles.");

      return;
   }
}
```

```
0: Boston 1: Chica
Enter city pair (I
Distance: 2011 mi

...

0: Boston 1: Chica
Enter city pair (I
Distance: 2960 mi

...

0: Boston 1: Chica
Enter city pair (I
Distance: 0 miles
```

A programmer can initialize a two-dimensional array's elements during definition using nested braces, as below. Multiple lines make the rows and columns more visible.

Construct 5.9.1: Initializing a two-dimensional array during definition.

```
// Initializing a 2D array
int[][] numVals = { {22, 44, 66}, {97, 98, 99} };

// Use multiple lines to make rows more visible
int[][] numVals = {
      {22, 44, 66}, // Row 0
      {97, 98, 99}  // Row 1
};
```

Arrays of three or more dimensions can also be defined, as in: `int[][][] myArray = new int[2][3][5]`, which defines a total of 2*3*5 or 30 elements. Note the rapid growth in size -- an array defined as `int[][][][] myArray = new int[100][100][5][3]` would have 100*100*5*3 or 150,000 elements. A programmer should make sure not to unnecessarily occupy available memory with a large array.

P Participation Activity

5.9.2: Two-dimensional arrays.

#	Question	Your answer
1	Define and initialize a two dimensional array of integers named dataVals with 4 rows and 7 columns using default element values.	
2	How many total integers elements are in an array with 4 rows and 7 columns?	
3	How many elements are in the array defined as: char[][] streetName = new char[20][50];	
4	Write a statement that assigns 99 into the fifth row, third column of array dataVals. Note: the first row/column is at index 0, not 1.	

C Challenge Activity

5.9.1: Find 2D array max and min.

Find the maximum value and minimum value in milesTracker. Assign the maximum value to maxMiles
Sample output for the given program:

```
Min miles: -10
Max miles: 40
```

```
      final int NUM_ROWS = 2;
      final int NUM_COLS = 2;
      int [][] milesTracker = new int[NUM_ROWS][NUM_COLS];
      int i = 0;
      int j = 0;
      int maxMiles = 0; // Assign with first element in milesTracker before loop
      int minMiles = 0; // Assign with first element in milesTracker before loop

      milesTracker[0][0] = -10;
      milesTracker[0][1] = 20;
      milesTracker[1][0] = 30;
      milesTracker[1][1] = 40;

      /* Your solution goes here  */

      System.out.println("Min miles: " + minMiles);
      System.out.println("Max miles: " + maxMiles);
   }
}
```

Run

Section 5.10 - Java example: Salary calculation with arrays

P Participation Activity

5.10.1: Various tax rates.

Arrays are useful to process tabular information. Income taxes are based on annual salary, usually with a tiered approach. Below is an example of a simple tax table:

Annual Salary	Tax Rate
0 to 20000	10%
Above 20000 to 50000	20%
Above 50000 to 100000	30%
Above 100000	40%

The below program uses an array salaryBase to hold the cutoffs for each salary level and a parallel array taxBase that has the corresponding tax rate.

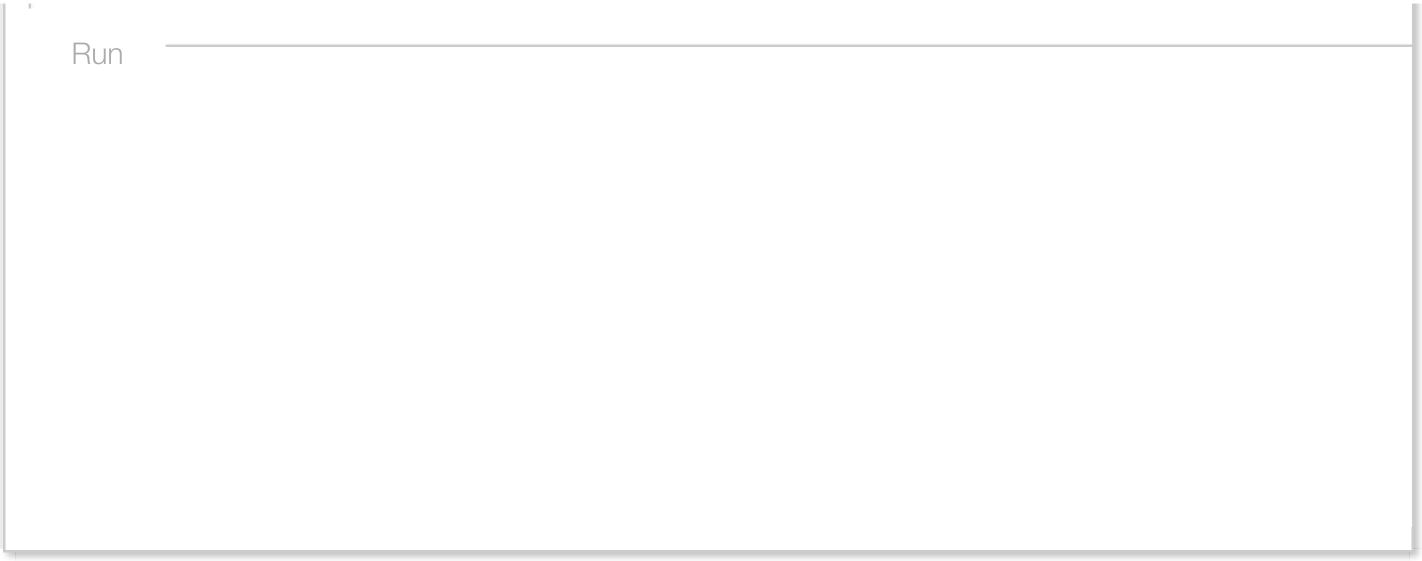
1. Run the program and enter annual salaries of 40000 and 50000, then enter 0.
2. Modify the program to use two parallel arrays named annualSalaries and taxesToPay, each with 10 elements. Array annualSalaries holds up to 10 annual salaries entered; array taxesToPay holds up to 10 corresponding amounts of taxes to pay for those annual salaries. Print the total annual salaries and taxes to pay after all input has been processed.
3. Run the program again with the same annual salary numbers as above.
4. Challenge: Modify the program from the previous step to use a 2-dimensional array of 10 elements named salariesAndTaxes instead of two one-dimensional parallel arrays. The 2D array's first column will hold the salaries, the second the taxes to pay for each salary.

The following program calculates the tax rate and tax to pay based on annual income.

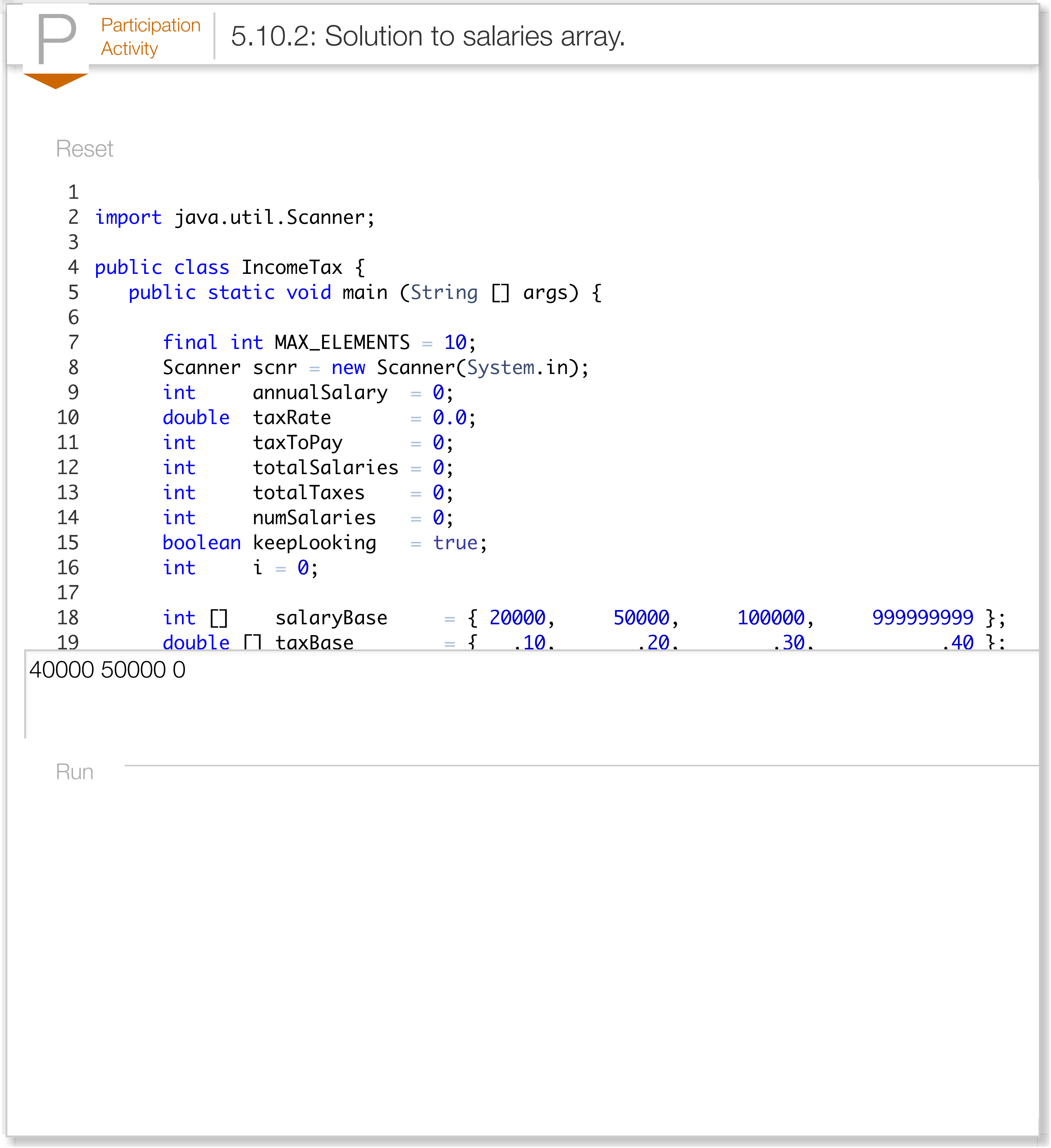
Reset

```

import java.util.Scanner;

public class IncomeTax {
   public static void main (String [] args) {

      final int MAX_ELEMENTS = 10;
      Scanner scnr = new Scanner(System.in);
      int     annualSalary = 0;
      double  taxRate      = 0.0;
      int     taxToPay     = 0;
      int     numSalaries  = 0;
      boolean keepLooking  = true;
      int     i = 0;

      int []    salaryBase = { 20000,     50000,     100000,     999999999 };
      double [] taxBase    = {   .10,       .20,        .30,            .40 };
      // FIXME: Define annualSalaries and taxesToPay arrays to hold 10 elements each
      // FIXME: Use the final constant MAX_ELEMENTS to declare the arrays
```

```
40000 50000 0
```



A solution to above problem follows.



5.10.2: Solution to salaries array.

```java

import java.util.Scanner;

public class IncomeTax {
   public static void main (String [] args) {

      final int MAX_ELEMENTS = 10;
      Scanner scnr = new Scanner(System.in);
      int     annualSalary  = 0;
      double  taxRate       = 0.0;
      int     taxToPay      = 0;
      int     totalSalaries = 0;
      int     totalTaxes    = 0;
      int     numSalaries   = 0;
      boolean keepLooking   = true;
      int     i = 0;

      int []    salaryBase     = { 20000,     50000,     100000,     999999999 };
      double [] taxBase        = {   .10,       .20,        .30,           .40 };
```

```
40000 50000 0
```

Section 5.11 - Java example: Domain name validation with

P Participation Activity

5.11.1: Validate domain names with arrays.

Arrays are useful to process lists.

A ***top-level domain*** (TLD) name is the last part of an Internet domain name like .com in example.com. A ***core generic top-level domain*** (core gTLD) is a TLD that is either .com, .net, .org, or .info. A ***restricted top-level domain*** is a TLD that is either .biz, .name, or .pro. A ***second-level domain*** is a single name that precedes a TLD as in apple in apple.com.

The following program repeatedly prompts for a domain name, and indicates whether that domain name consists of a second-level domain followed by a core gTLD. Valid core gTLD's are stored in an array. For this program, a valid domain name must contain only one period, such as apple.com, but not support.apple.com. The program ends when the user presses just the Enter key in response to a prompt.

1. Run the program and enter domain names to validate.
2. Extend the program to also recognize restricted TLDs using an array, and statements to validate against that array. The program should also report whether the TLD is a core gTLD or a restricted gTLD. Run the program again.

Reset

```java
import java.util.Scanner;

public class GtldValidation {

   public static void main (String [ ] args) {
      Scanner scnr = new Scanner(System.in);

      // Define the list of valid core gTLDs
      String [ ] validCoreGtld = { ".com", ".net", ".org", ".info" };
      // FIXME: Define an array named validRestrictedGtld that has the names
      //        of the restricted domains, .biz, .name, and .pro
      String  inputName         = "";
      String  searchName        = "";
      String  theGtld           = "";
      boolean isValidDomainName = false;
      boolean isCoreGtld        = false;
      boolean isRestrictedGtld  = false;
      int     periodCounter     = 0;
      int     periodPosition    = 0;
```

```
apple.com
APPLE.com
apple.comm
```

Run

P Participation Activity

5.11.2: Validate domain names with arrays (solution).

A solution to the problem posed above follows.

Reset

```
import java.util.Scanner;

public class GtldValidation_Solution {

   public static void main (String [ ] args) {
      Scanner scnr = new Scanner(System.in);

      // Define the list of valid core gTLDs
      String [ ] validCoreGtld       = { ".com", ".net", ".org", ".info" };
      String [ ] validRestrictedGtld = { ".biz", ".name", ".pro" };
      String  inputName          = "";
      String  searchName         = "";
      String  theGtld            = "";
      boolean isValidDomainName  = false;
      boolean isCoreGtld         = false;
      boolean isRestrictedGtld   = false;
      int     periodCounter      = 0;
      int     periodPosition     = 0;
      int     i = 0;
```

```
apple.com
APPLE.com
apple.comm
```

Run