Lesson 5 – Part II
Arrays and ArrayLists

Assoc. Prof. Marenglen Biba

(C) 2010 Pearson Education, Inc. All rights reserved.

OBJECTIVES

In this Chapter you'll learn:

- What arrays are.
- To use arrays to store data in and retrieve data from lists and tables of values.
- To declare arrays, initialize arrays and refer to individual elements of arrays.
- To use the enhanced `for` statement to iterate through arrays.
- To pass arrays to methods.
- To declare and manipulate multidimensional arrays.
- To write methods that use variable-length argument lists.
- To read command-line arguments into a program.

(C) 2010 Pearson Education, Inc. All rights reserved.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.1 Introduction

- Data structures
  - Collections of related data items.
- Arrays
  - Data structures consisting of related data items of the same type.
  - Make it convenient to process related groups of values.
  - Remain the same length once they are created.

- Enhanced `for` statement for iterating over an array or collection of data items.

- Process command-line arguments in method `main`.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.1 Introduction (Cont.)

- Common array manipulations with `static` methods of class `Arrays` from the `java.util` package.

- `ArrayList` collection
  - Similar to arrays
  - Dynamic resizing
    - They automatically increase their size at execution time to accommodate additional elements

(C) 2010 Pearson Education, Inc. All rights reserved.

7.2 Arrays

- Array
  - Group of variables (called elements) containing values of the same type.
  - Arrays are objects so they are reference types.
  - Elements can be either primitive or reference types.
- Refer to a particular element in an array
  - Use the element's index.
  - Array-access expression—the name of the array followed by the index of the particular element in square brackets, `[]`.
- The first element in every array has index zero.
- The highest index in an array is one less than the number of elements in the array.

(C) 2010 Pearson Education, Inc. All rights reserved.

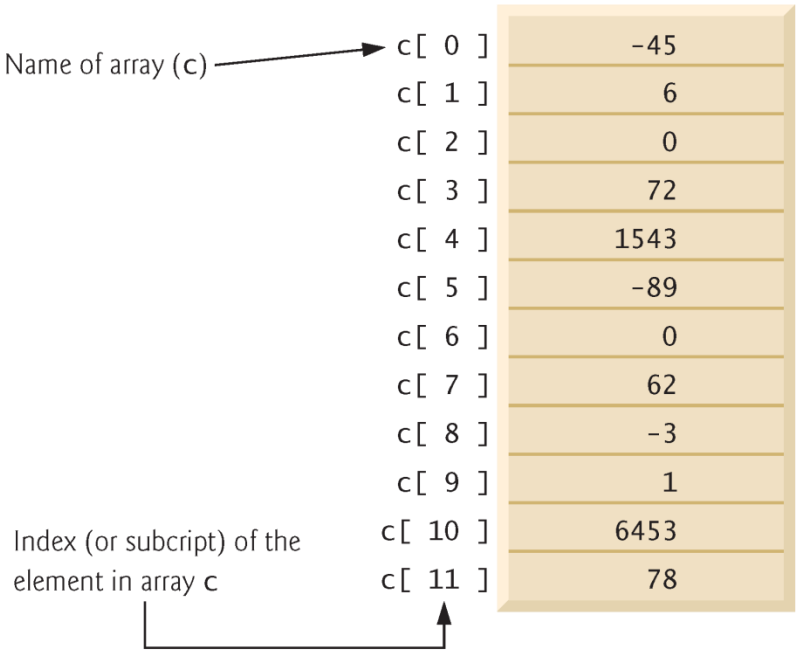

Fig. 7.1 | A 12-element array.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.2 Arrays (Cont.)

- An index must be a nonnegative integer.
  - Can use an expression as an index.
- An indexed array name is an array-access expression.
  - Can be used on the left side of an assignment to place a new value into an array element.
- Every array object knows its own length and stores it in a `length` instance variable.
  - `length` cannot be changed because it's a `final` variable.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.3 Declaring and Creating Arrays

- Array objects
  - Created with keyword `new`.
  - You specify the element type and the number of elements in an array-creation expression, which returns a reference that can be stored in an array variable.
- Declaration and array-creation expression for an array of 12 `int` elements

```
int[] c = new int[ 12 ];
```

- Can be performed in two steps as follows:

```
int[] c; // declare the array variable
c = new int[ 12 ]; // creates the array
```

(C) 2010 Pearson Education, Inc. All rights reserved.

7.3 Declaring and Creating Arrays (Cont.)

- When an array is created, each element of the array receives a default value
  - Zero for the numeric primitive-type elements, `false` for `boolean` elements and `null` for references.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.3 Declaring and Creating Arrays (Cont.)

- Every element of a primitive-type array contains a value of the array's declared element type.
  - Every element of an `int` array is an `int` value.

- Every element of a reference-type array is a reference to an object of the array's declared element type.
  - Every element of a `String` array is a reference to a `String` object.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.4 Examples Using Arrays

- Fig. 7.2 uses keyword `new` to create an array of 10 `int` elements, which are initially zero (the default for `int` variables).

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.2: InitArray.java
2  // Initializing the elements of an array to default values of zero.
3
4  public class InitArray
5  {
6     public static void main( String[] args )
7     {
8        int[] array; // declare array named array
9
10       array = new int[ 10 ]; // create the array object
11
12       System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings
13
14       // output each array element's value
15       for ( int counter = 0; counter < array.length; counter++ )
16          System.out.printf( "%5d%8d\n", counter, array[ counter ] );
17    } // end main
18 } // end class InitArray
```

Variable `array` will refer to an array of `int` values.

Creates an array of `10` `int` elements, each with the value `0` by default

`for` statement iterates while `counter` is less than the array's `length`

Array-access expression gets the value at the index represented by `counter`

Fig. 7.2 | Initializing the elements of an array to default values of zero. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
Index   Value
    0       0
    1       0
    2       0
    3       0
    4       0
    5       0
    6       0
    7       0
    8       0
    9       0
```

Fig. 7.2 | Initializing the elements of an array to default values of zero. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.4 Examples Using Arrays (Cont.)

- Array initializer
  - A comma-separated list of expressions (called an initializer list) enclosed in braces.
  - Used to create an array and initialize its elements.
  - Array length is determined by the number of elements in the initializer list.

```
int[] n = { 10, 20, 30, 40, 50 };
```

    - Creates a five-element array with index values 0–4.
- Compiler counts the number of initializers in the list to determine the size of the array
  - Sets up the appropriate `new` operation "behind the scenes."

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.3: InitArray.java
2  // Initializing the elements of an array with an array initializer.
3
4  public class InitArray
5  {
6     public static void main( String[] args )
7     {
8        // initializer list specifies the value for each element
9        int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10
11       System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings
12
13       // output each array element's value
14       for ( int counter = 0; counter < array.length; counter++ )
15          System.out.printf( "%5d%8d\n", counter, array[ counter ] );
16    } // end main
17 } // end class InitArray
```

Array initializer list for a 10-element `int` array

Fig. 7.3 | Initializing the elements of an array with an array initializer. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
Index   Value
    0      32
    1      27
    2      64
    3      18
    4      95
    5      14
    6      90
    7      70
    8      60
    9      37
```

Fig. 7.3 | Initializing the elements of an array with an array initializer. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.4 Examples Using Arrays (Cont.)

- The application in Fig. 7.4 creates a 10-element array and assigns to each element one of the even integers from 2 to 20 (2, 4, 6, …, 20).

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.4: InitArray.java
2  // Calculating values to be placed into elements of an array.
3
4  public class InitArray
5  {
6     public static void main( String[] args )
7     {
8        final int ARRAY_LENGTH = 10; // declare constant
9        int[] array = new int[ ARRAY_LENGTH ]; // create array
10
11       // calculate value for each array element
12       for ( int counter = 0; counter < array.length; counter++ )
13          array[ counter ] = 2 + 2 * counter;
14
15       System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings
16
17       // output each array element's value
18       for ( int counter = 0; counter < array.length; counter++ )
19          System.out.printf( "%5d%8d\n", counter, array[ counter ] );
20    } // end main
21 } // end class InitArray
```

Named constant representing the array length

Create an array with ARRAY_LENGTH elements

Calculates and stores a value for each element of the array

Fig. 7.4 | Calculating the values to be placed into the elements of an array. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
Index   Value
    0       2
    1       4
    2       6
    3       8
    4      10
    5      12
    6      14
    7      16
    8      18
    9      20
```

Fig. 7.4 | Calculating the values to be placed into the elements of an array. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.4 Examples Using Arrays (Cont.)

- `final` variables must be initialized before they are used and cannot be modified thereafter.
- An attempt to modify a `final` variable after it's initialized causes a compilation error
  - `cannot assign a value to final variable` *`variableName`*
- An attempt to access the value of a `final` variable before it's initialized causes a compilation error
  - `variable` *`variableName`* `might not have been initialized`

(C) 2010 Pearson Education, Inc. All rights reserved.

7.4 Examples Using Arrays (Cont.)

- Figure 7.5 sums the values contained in a 10-element integer array.
- Often, the elements of an array represent a series of values to be used in a calculation.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.5: SumArray.java
2  // Computing the sum of the elements of an array.
3
4  public class SumArray
5  {
6     public static void main( String[] args )
7     {
8        int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for ( int counter = 0; counter < array.length; counter++ )
13          total += array[ counter ];
14
15       System.out.printf( "Total of array elements: %d\n", total );
16    } // end main
17 } // end class SumArray
```

Adds each value in `array` to `total`, which is displayed when the loop terminates

```
Total of array elements: 849
```

Fig. 7.5 | Computing the sum of the elements of an array.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.4 Examples Using Arrays (Cont.)

- Many programs present data to users in a graphical manner.
- Numeric values are often displayed as bars in a bar chart.
  - Longer bars represent proportionally larger numeric values.
- A simple way to display numeric data is with a bar chart that shows each numeric value as a bar of asterisks (*).
- Format specifier `%02d` indicates that an `int` value should be formatted as a field of two digits.
  - The `0` flag displays a leading `0` for values with fewer digits than the field width (`2`).

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.6: BarChart.java
2  // Bar chart printing program.
3
4  public class BarChart
5  {
6     public static void main( String[] args )
7     {
8        int[] array = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };
9
10       System.out.println( "Grade distribution:" );
11
12       // for each array element, output a bar of the chart
13       for ( int counter = 0; counter < array.length; counter++ )
14       {
15          // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
16          if ( counter == 10 )
17             System.out.printf( "%5d: ", 100 );
18          else
19             System.out.printf( "%02d-%02d: ",
20                counter * 10, counter * 10 + 9  );
21
```

Fig. 7.6 | Bar chart printing program. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
22            // print bar of asterisks
23            for ( int stars = 0; stars < array[ counter ]; stars++ )
24               System.out.print( "*" );
25
26            System.out.println(); // start a new line of output
27         } // end outer for
28      } // end main
29   } // end class BarChart
```

```
Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

Nested `for` loop uses the outer `for` loop's **`counter`** variable to determine which element of the array to access, then displays the appropriate number of asterisks

Fig. 7.6 | Bar chart printing program. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.4 Examples Using Arrays (Cont.)

- Sometimes, programs use counter variables to summarize data, such as the results of a survey.
- Fig. 6.8 used separate counters in a die-rolling program to track the number of occurrences of each side of a six-sided die as the program rolled the die 6000 times.
- Fig. 7.7 shows an array version of this application.
  - Line 14 of this program replaces lines 23–46 of Fig. 6.8.
- Array `frequency` must be large enough to store six counters.
  - We use a seven-element array in which we ignore `frequency[0]`
  - More logical to have the face value 1 increment `frequency[1]` than `frequency[0]`.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.7: RollDie.java
2  // Die-rolling program using arrays instead of switch.
3  import java.util.Random;
4
5  public class RollDie
6  {
7     public static void main( String[] args )
8     {
9        Random randomNumbers = new Random(); // random number generator
10       int[] frequency = new int[ 7 ]; // array of frequency counters
11
12       // roll die 6000 times; use die value as frequency index
13       for ( int roll = 1; roll <= 6000; roll++ )
14          ++frequency[ 1 + randomNumbers.nextInt( 6 ) ];
15
16       System.out.printf( "%s%10s\n", "Face", "Frequency" );
17
18       // output each array element's value
19       for ( int face = 1; face < frequency.length; face++ )
20          System.out.printf( "%4d%10d\n", face, frequency[ face ] );
21    } // end main
22 } // end class RollDie
```

Random number from 1 to 6 is used as index into `frequency` array to determine which element to increment

Fig. 7.7 | Die-rolling program using arrays instead of `switch`. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
Face Frequency
   1        988
   2        963
   3       1018
   4       1041
   5        978
   6       1012
```

Fig. 7.7 | Die-rolling program using arrays instead of `switch`. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.4 Examples Using Arrays (Cont.)

- Figure 7.8 uses arrays to summarize the results of data collected in a survey:
  - *Forty students were asked to rate the quality of the food in the student cafeteria on a scale of 1 to 10 (where 1 means awful and 10 means excellent). Place the 40 responses in an integer array, and summarize the results of the poll.*
- Array `responses` is a 40-element `int` array of the survey responses.
- 11-element array `frequency` counts the number of occurrences of each response (1 to 10).
  - Each element is initialized to zero by default.
  - We ignore `frequency[0]`.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.8: StudentPoll.java
2  // Poll analysis program.
3
4  public class StudentPoll
5  {
6     public static void main( String[] args )
7     {
8        // array of survey responses
9        int[] responses = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10, 1, 6, 3, 8, 6,
10          10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6, 5, 6, 7, 5, 6,
11          4, 8, 6, 8, 10 };
12       int[] frequency = new int[ 11 ]; // array of frequency counters
13
14       // for each answer, select responses element and use that value
15       // as frequency index to determine element to increment
16       for ( int answer = 0; answer < responses.length; answer++ )
17          ++frequency[ responses[ answer ] ];
18
19       System.out.printf( "%s%10s", "Rating", "Frequency" );
20
21       // output each array element's value
22       for ( int rating = 1; rating < frequency.length; rating++ )
23          System.out.printf( "%d%10d", rating, frequency[ rating ] );
24    } // end main
25 } // end class StudentPoll
```

Program needs only 10 counters; we ignore element 0

Increments appropriate element of `frequency` based on the value of `responses[answer]`

Fig. 7.8 | Poll analysis program. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
Rating Frequency
     1         2
     2         2
     3         2
     4         2
     5         5
     6        11
     7         5
     8         7
     9         1
    10         3
```

Fig. 7.8 | Poll analysis program. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.4 Examples Using Arrays (Cont.)

- If the data in the `responses` array contained invalid values, such as 13, the program would have attempted to add `1` to `frequency[13]`, which is outside the bounds of the array.
  - Java doesn't allow this.
  - JVM checks array indices to ensure that they are greater than or equal to 0 and less than the length of the array—this is called bounds checking.
  - If a program uses an invalid index, Java generates a so-called exception to indicate that an error occurred in the program at execution time.

(C) 2010 Pearson Education, Inc. All rights reserved.

Error-Prevention Tip 7.1

An exception indicates that an error has occurred in a program. You often can write code to recover from an exception and continue program execution, rather than abnormally terminating the program. When a program attempts to access an element outside the array bounds, an `ArrayIndexOutOfBoundsException` *occurs. Exception handling is discussed in Chapter 11.*

(C) 2010 Pearson Education, Inc. All rights reserved.

7.5 Case Study: Card Shuffling and Dealing Simulation

- Examples thus far used arrays containing elements of primitive types.
- Elements of an array can be either primitive types or reference types.
- Next example uses an array of reference-type elements — objects representing playing cards—to develop a class that simulates card shuffling and dealing.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.5 Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Class `Card` (Fig. 7.9) contains two `String` instance variables—`face` and `suit`—that are used to store references to the face and suit names for a specific `Card`.
- Method `toString` creates a `String` consisting of the `face` of the `card`, `" of "` and the `suit` of the card.
  - Can invoke explicitly to obtain a string representation of a `Card`.
  - Called implicitly when the object is used where a `String` is expected.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.9: Card.java
2  // Card class represents a playing card.
3
4  public class Card
5  {
6     private String face; // face of card ("Ace", "Deuce", ...)
7     private String suit; // suit of card ("Hearts", "Diamonds", ...)
8
9     // two-argument constructor initializes card's face and suit
10    public Card( String cardFace, String cardSuit )
11    {
12       face = cardFace; // initialize face of card
13       suit = cardSuit; // initialize suit of card
14    } // end two-argument Card constructor
15
16    // return String representation of Card
17    public String toString()
18    {
19       return face + " of " + suit;
20    } // end method toString
21 } // end class Card
```

Must be declared with this first line it is to be called implicitly to convert `Card` objects to `String` representations

Fig. 7.9 | `Card` class represents a playing card.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.5 Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Class `DeckOfCards` (Fig. 7.10) declares as an instance variable a `Card` array named `deck`.
- `Deck`'s elements are `null` by default
  - Constructor fills the `deck` array with `Card` objects.
- Method `shuffle` shuffles the `Card`s in the deck.
  - Loops through all 52 `Card`s (array indices 0 to 51).
  - Each `Card` swapped with a randomly chosen other card in the deck.
- Method `dealCard` deals one `Card` in the array.
  - `currentCard` indicates the index of the next `Card` to be dealt
  - Returns `null` if there are no more cards to deal

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.10: DeckOfCards.java
2  // DeckOfCards class represents a deck of playing cards.
3  import java.util.Random;
4
5  public class DeckOfCards
6  {
7     private Card[] deck; // array of Card objects
8     private int currentCard; // index of next Card to be dealt
9     private static final int NUMBER_OF_CARDS = 52; // constant # of Cards
10    // random number generator
11    private static final Random randomNumbers = new Random();
12
13    // constructor fills deck of Cards
14    public DeckOfCards()
15    {
16       String[] faces = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
17          "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
18       String[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };
19
20       deck = new Card[ NUMBER_OF_CARDS ]; // create array of Card objects
21       currentCard = 0; // set currentCard so first Card dealt is deck[ 0 ]
22
```

Declares the array of `Card`s; `deck` is `null` until the array is created

Creates the array of `Card` variables

Fig. 7.10 | `DeckOfCards` class represents a deck of playing cards. (Part 1 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
23          // populate deck with Card objects
24          for ( int count = 0; count < deck.length; count++ )
25             deck[ count ] =
26                new Card( faces[ count % 13 ], suits[ count / 13 ] );
27       } // end DeckOfCards constructor
28
29       // shuffle deck of Cards with one-pass algorithm
30       public void shuffle()
31       {
32          // after shuffling, dealing should start at deck[ 0 ] again
33          currentCard = 0; // reinitialize currentCard
34
35          // for each Card, pick another random Card and swap them
36          for ( int first = 0; first < deck.length; first++ )
37          {
38             // select a random number between 0 and 51
39             int second =  randomNumbers.nextInt( NUMBER_OF_CARDS );
40
41             // swap current Card with randomly selected Card
42             Card temp = deck[ first ];
43             deck[ first ] = deck[ second ];
44             deck[ second ] = temp;
45          } // end for
46       } // end method shuffle
```

Creates a `Card` for the current array element

Swaps the current element with the randomly selected element

Fig. 7.10 | `DeckOfCards` class represents a deck of playing cards. (Part 2 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
47
48      // deal one Card
49      public Card dealCard()
50      {
51         // determine whether Cards remain to be dealt
52         if ( currentCard < deck.length )
53            return deck[ currentCard++ ]; // return current Card in array
54         else
55            return null; // return null to indicate that all Cards were dealt
56      } // end method dealCard
57   } // end class DeckOfCards
```

Ensures that `currentCard` is less than the `length` of the array; if so, a `Card` is returned; otherwise, `null` is returned

Fig. 7.10 | `DeckOfCards` class represents a deck of playing cards. (Part 3 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.5 Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Figure 7.11 demonstrates class `DeckOfCards` (Fig. 7.10).
- When a `Card` is output as a `String`, the `Card`'s `toString` method is implicitly invoked.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.11: DeckOfCardsTest.java
2  // Card shuffling and dealing.
3
4  public class DeckOfCardsTest
5  {
6     // execute application
7     public static void main( String[] args )
8     {
9        DeckOfCards myDeckOfCards = new DeckOfCards();
10       myDeckOfCards.shuffle(); // place Cards in random order
11
12       // print all 52 Cards in the order in which they are dealt
13       for ( int i = 1; i <= 52; i++ )
14       {
15          // deal and display a Card
16          System.out.printf( "%-19s", myDeckOfCards.dealCard() );
17
18          if ( i % 4 == 0 ) // output newline every 4 cards
19             System.out.println();
20       } // end for
21    } // end main
22 } // end class DeckOfCardsTest
```

Deals a `Card`; the `Card`'s `toString` method is called implicitly to obtain the `String` representation that is output

Fig. 7.11 | Card shuffling and dealing. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
Six of Spades      Eight of Spades    Six of Clubs       Nine of Hearts
Queen of Hearts    Seven of Clubs     Nine of Spades     King of Hearts
Three of Diamonds  Deuce of Clubs     Ace of Hearts      Ten of Spades
Four of Spades     Ace of Clubs       Seven of Diamonds  Four of Hearts
Three of Clubs     Deuce of Hearts    Five of Spades     Jack of Diamonds
King of Clubs      Ten of Hearts      Three of Hearts    Six of Diamonds
Queen of Clubs     Eight of Diamonds  Deuce of Diamonds  Ten of Diamonds
Three of Spades    King of Diamonds   Nine of Clubs      Six of Hearts
Ace of Spades      Four of Diamonds   Seven of Hearts    Eight of Clubs
Deuce of Spades    Eight of Hearts    Five of Hearts     Queen of Spades
Jack of Hearts     Seven of Spades    Four of Clubs      Nine of Diamonds
Ace of Diamonds    Queen of Diamonds  Five of Clubs      King of Spades
Five of Diamonds   Ten of Clubs       Jack of Spades     Jack of Clubs
```

Fig. 7.11 | Card shuffling and dealing. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.6 Enhanced `for` Statement

- Enhanced `for` statement
  - Iterates through the elements of an array without using a counter.
  - Avoids the possibility of "stepping outside" the array.
  - Also works with the Java API's prebuilt collections (see Section 7.14).
- Syntax:

```
for ( parameter : arrayName )
   statement
```

  where *parameter* has a type and an identifier and *arrayName* is the array through which to iterate.
- Parameter type must be consistent with the array's element type.
- The enhanced `for` statement simplifies the code for iterating through an array.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.12: EnhancedForTest.java
2  // Using enhanced for statement to total integers in an array.
3
4  public class EnhancedForTest
5  {
6     public static void main( String[] args )
7     {
8        int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for ( int number : array )
13          total += number;
14
15       System.out.printf( "Total of array elements: %d\n", total );
16    } // end main
17 } // end class EnhancedForTest
```

For each element in `array`, assign the element's value to `number`, then add `number` to `total`

```
Total of array elements: 849
```

Fig. 7.12 | Using the enhanced `for` statement to total integers in an array.

The enhanced `for` statement simplifies the code for iterating through an array.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.6 Enhanced `for` Statement (Cont.)

- The enhanced `for` statement can be used only to obtain array elements
  - It cannot be used to modify elements.
  - To modify elements, use the traditional counter-controlled `for` statement.
- Can be used in place of the counter-controlled `for` statement if you don't need to access the index of the element.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.7 Passing Arrays to Methods

- To pass an array argument to a method, specify the name of the array without any brackets.
  - Since every array object "knows" its own length, we need not pass the array length as an additional argument.
- To receive an array, the method's parameter list must specify an array parameter.
- When an argument to a method is an entire array or an individual array element of a reference type, the called method receives a copy of the reference.
- When an argument to a method is an individual array element of a primitive type, the called method receives a copy of the element's value.
  - Such primitive values are called scalars or scalar quantities.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.13: PassArray.java
2  // Passing arrays and individual array elements to methods.
3
4  public class PassArray
5  {
6     // main creates array and calls modifyArray and modifyElement
7     public static void main( String[] args )
8     {
9        int[] array = { 1, 2, 3, 4, 5 };
10
11       System.out.println(
12          "Effects of passing reference to entire array:\n" +
13          "The values of the original array are:" );
14
15       // output original array elements
16       for ( int value : array )
17          System.out.printf( "   %d", value );
18
19       modifyArray( array ); // pass array reference
20       System.out.println( "\n\nThe values of the modified array are:" );
21
22       // output modified array elements
23       for ( int value : array )
24          System.out.printf( "   %d", value );
```

Passes the reference to `array` into method `modifyArray`

Fig. 7.13 | Passing arrays and individual array elements to methods. (Part 1 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
25
26          System.out.printf(
27             "\n\nEffects of passing array element value:\n" +
28             "array[3] before modifyElement: %d\n", array[ 3 ] );
29
30          modifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
31          System.out.printf(
32             "array[3] after modifyElement: %d\n", array[ 3 ] );
33       } // end main
34
35       // multiply each element of an array by 2
36       public static void modifyArray( int[] array2 )
37       {
38          for ( int counter = 0; counter < array2.length; counter++ )
39             array2[ counter ] *= 2;
40       } // end method modifyArray
41
42       // multiply argument by 2
43       public static void modifyElement( int element )
44       {
45          element *= 2;
46          System.out.printf(
47             "Value of element in modifyElement: %d\n", element );
48       } // end method modifyElement
49    } // end class PassArray
```

Passes a copy of `array[3]`'s `int` value into `modifyElement`

Method receives copy of an array's reference, which gives the method direct access to the original array in memory

Method receives copy of an `int` value; the method cannot modify the original `int` value in `main`

Fig. 7.13 | Passing arrays and individual array elements to methods. (Part 2 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8
```

Fig. 7.13 | Passing arrays and individual array elements to methods. (Part 3 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.7 Passing Arrays to Methods (Cont.)

- Pass-by-value (also called call-by-value)
  - A copy of the argument's *value is passed to the called method.*
  - The called method works exclusively with the copy.
  - Changes to the called method's copy do not affect the original variable's value in the caller.
- Pass-by-reference (also called call-by-reference)
  - The called method can access the argument's value in the caller directly and modify that data, if necessary.
  - Improves performance by eliminating the need to copy possibly large amounts of data.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.7 Passing Arrays to Methods (Cont.)

- A method call can pass two types of values to a method
  - Copies of primitive values
  - Copies of references to objects

- Objects cannot be passed to methods.
  - Reference to objects are instead passed

(C) 2010 Pearson Education, Inc. All rights reserved.

Performance Tip 7.1

Passing arrays by reference makes sense for performance reasons. If arrays were passed by value, a copy of each element would be passed. For large, frequently passed arrays, this would waste time and consume considerable storage for the copies of the arrays.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.8 Case Study: Class `GradeBook` Using an Array to Store Grades

- Previous versions of class `GradeBook` process a set of grades entered by the user, but do not maintain the individual grade values in instance variables of the class.
  - Repeat calculations require the user to reenter the same grades.
  - We solve this problem by storing grades in an array.
- The `grades` array's size is determined by the length of the array that is passed to the constructor.
  - So a `GradeBook` object can process a variable number of grades.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.14: GradeBook.java
2  // GradeBook class using an array to store test grades.
3
4  public class GradeBook
5  {
6     private String courseName; // name of course this GradeBook represents
7     private int[] grades; // array of student grades
8
9     // two-argument constructor initializes courseName and grades array
10    public GradeBook( String name, int[] gradesArray )
11    {
12       courseName = name; // initialize courseName
13       grades = gradesArray; // store grades
14    } // end two-argument GradeBook constructor
15
16    // method to set the course name
17    public void setCourseName( String name )
18    {
19       courseName = name; // store the course name
20    } // end method setCourseName
21
```

Will refer to an array passed by the creator of the GradeBook

Receives the array from the GradeBook creator

Initializes the grades instance variable

Fig. 7.14 | GradeBook class using an array to store test grades. (Part 1 of 7.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
22      // method to retrieve the course name
23      public String getCourseName()
24      {
25         return courseName;
26      } // end method getCourseName
27
28      // display a welcome message to the GradeBook user
29      public void displayMessage()
30      {
31         // getCourseName gets the name of the course
32         System.out.printf( "Welcome to the grade book for\n%s!\n\n",
33            getCourseName() );
34      } // end method displayMessage
35
36      // perform various operations on the data
37      public void processGrades()
38      {
39         // output grades array
40         outputGrades();
41
42         // call method getAverage to calculate the average grade
43         System.out.printf( "\nClass average is %.2f\n", getAverage() );
44
```

Fig. 7.14 | GradeBook class using an array to store test grades. (Part 2 of 7.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
45          // call methods getMinimum and getMaximum
46          System.out.printf( "Lowest grade is %d\nHighest grade is %d\n\n",
47             getMinimum(), getMaximum() );
48
49          // call outputBarChart to print grade distribution chart
50          outputBarChart();
51       } // end method processGrades
52
53       // find minimum grade
54       public int getMinimum()
55       {
56          int lowGrade = grades[ 0 ]; // assume grades[ 0 ] is smallest
57
58          // loop through grades array
59          for ( int grade : grades )
60          {
61             // if grade lower than lowGrade, assign it to lowGrade
62             if ( grade < lowGrade )
63                lowGrade = grade; // new lowest grade
64          } // end for
65
66          return lowGrade; // return lowest grade
67       } // end method getMinimum
68
```

Fig. 7.14 | GradeBook class using an array to store test grades. (Part 3 of 7.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
69      // find maximum grade
70      public int getMaximum()
71      {
72         int highGrade = grades[ 0 ]; // assume grades[ 0 ] is largest
73
74         // loop through grades array
75         for ( int grade : grades )
76         {
77            // if grade greater than highGrade, assign it to highGrade
78            if ( grade > highGrade )
79               highGrade = grade; // new highest grade
80         } // end for
81
82         return highGrade; // return highest grade
83      } // end method getMaximum
84
```

Fig. 7.14 | GradeBook class using an array to store test grades. (Part 4 of 7.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
85      // determine average grade for test
86      public double getAverage()
87      {
88         int total = 0; // initialize total
89
90         // sum grades for one student
91         for ( int grade : grades )
92            total += grade;
93
94         // return average of grades
95         return (double) total / grades.length;
96      } // end method getAverage
97
98      // output bar chart displaying grade distribution
99      public void outputBarChart()
100     {
101        System.out.println( "Grade distribution:" );
102
103        // stores frequency of grades in each range of 10 grades
104        int[] frequency = new int[ 11 ];
105
106        // for each grade, increment the appropriate frequency
107        for ( int grade : grades )
108           ++frequency[ grade / 10 ];
```

Calculation is based on the length of the array used to initialize the GradeBook

Fig. 7.14 | GradeBook class using an array to store test grades. (Part 5 of 7.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
109
110        // for each grade frequency, print bar in chart
111        for ( int count = 0; count < frequency.length; count++ )
112        {
113           // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
114           if ( count == 10 )
115              System.out.printf( "%5d: ", 100 );
116           else
117              System.out.printf( "%02d-%02d: ",
118                 count * 10, count * 10 + 9  );
119
120           // print bar of asterisks
121           for ( int stars = 0; stars < frequency[ count ]; stars++ )
122              System.out.print( "*" );
123
124           System.out.println(); // start a new line of output
125        } // end outer for
126     } // end method outputBarChart
127
```

Fig. 7.14 | GradeBook class using an array to store test grades. (Part 6 of 7.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
128    // output the contents of the grades array
129    public void outputGrades()
130    {
131       System.out.println( "The grades are:\n" );
132
133       // output each student's grade
134       for ( int student = 0; student < grades.length; student++ )
135          System.out.printf( "Student %2d: %3d\n",
136             student + 1, grades[ student ] );
137    } // end method outputGrades
138 } // end class GradeBook
```

Fig. 7.14 | `GradeBook` class using an array to store test grades. (Part 7 of 7.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.8 Case Study: Class GradeBook Using an Array to Store Grades (Cont.)

- The application of Fig. 7.15 creates an object of class `GradeBook` (Fig. 7.14) using the `int` array `grades-Array`.
- Lines 12–13 pass a course name and `gradesArray` to the `GradeBook` constructor.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.15: GradeBookTest.java
2  // GradeBookTest creates a GradeBook object using an array of grades,
3  // then invokes method processGrades to analyze them.
4  public class GradeBookTest
5  {
6     // main method begins program execution
7     public static void main( String[] args )
8     {
9        // array of student grades
10       int[] gradesArray = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
11
12       GradeBook myGradeBook = new GradeBook(
13          "CS101 Introduction to Java Programming", gradesArray );
14       myGradeBook.displayMessage();
15       myGradeBook.processGrades();
16    } // end main
17 } // end class GradeBookTest
```

Initializes the GradeBook with array gradesArray

Fig. 7.15 | GradeBookTest creates a GradeBook object using an array of grades, then invokes method processGrades to analyze them. (Part 1 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

The grades are:

Student  1:  87
Student  2:  68
Student  3:  94
Student  4: 100
Student  5:  83
Student  6:  78
Student  7:  85
Student  8:  91
Student  9:  76
Student 10:  87

Class average is 84.90
Lowest grade is 68
Highest grade is 100
```

Fig. 7.15 | `GradeBookTest` creates a `GradeBook` object using an array of grades, then invokes method `processGrades` to analyze them. (Part 2 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

Fig. 7.15 | `GradeBookTest` creates a `GradeBook` object using an array of grades, then invokes method `processGrades` to analyze them. (Part 3 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.9 Multidimensional Arrays

- Two-dimensional arrays are often used to represent tables of values consisting of information arranged in rows and columns.
- Identify a particular table element with two indices.
  - By convention, the first identifies the element's row and the second its column.
- Multidimensional arrays can have more than two dimensions.
- Java does not support multidimensional arrays directly
  - Allows you to specify one-dimensional arrays whose elements are also one-dimensional arrays, thus achieving the same effect.
- In general, an array with *m* rows and *n* columns is called an *m*-by-*n* array.

(C) 2010 Pearson Education, Inc. All rights reserved.

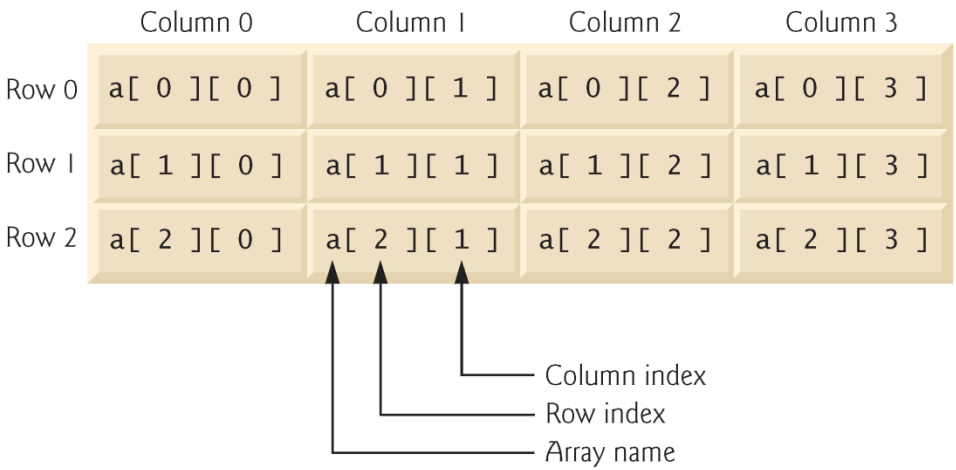

Fig. 7.16 | Two-dimensional array with three rows and four columns.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.9 Multidimensional Arrays (Cont.)

- Multidimensional arrays can be initialized with array initializers in declarations.
- A two-dimensional array `b` with two rows and two columns could be declared and initialized with nested array initializers as follows:

```
int[][] b = { { 1, 2 }, { 3, 4 } };
```

  - The initial values are grouped by row in braces.
  - The number of nested array initializers (represented by sets of braces within the outer braces) determines the number of rows.
  - The number of initializer values in the nested array initializer for a row determines the number of columns in that row.
  - Rows can have different lengths.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.9 Multidimensional Arrays (Cont.)

- The lengths of the rows in a two-dimensional array are not required to be the same:

```
int[][] b = { { 1, 2 }, { 3, 4, 5 } };
```

  - Each element of `b` is a reference to a one-dimensional array of `int` variables.
  - The `int` array for row `0` is a one-dimensional array with two elements (`1` and `2`).
  - The `int` array for row `1` is a one-dimensional array with three elements (`3`, `4` and `5`).

(C) 2010 Pearson Education, Inc. All rights reserved.

7.9 Multidimensional Arrays (Cont.)

- A multidimensional array with the same number of columns in every row can be created with an array-creation expression.

```
int[][] b = new int[ 3 ][ 4 ];
```

  - 3 rows and 4 columns.
- The elements of a multidimensional array are initialized when the array object is created.
- A multidimensional array in which each row has a different number of columns can be created as follows:

```
int[][] b = new int[ 2 ][ ];   // create 2 rows
b[ 0 ] = new int[ 5 ]; // create 5 columns for row 0
b[ 1 ] = new int[ 3 ]; // create 3 columns for row 1
```

  - Creates a two-dimensional array with two rows.
  - Row 0 has five columns, and row 1 has three columns.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.9 Multidimensional Arrays (Cont.)

- Figure 7.17 demonstrates initializing two-dimensional arrays with array initializers and using nested `for` loops to traverse the arrays.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.17: InitArray.java
2  // Initializing two-dimensional arrays.
3
4  public class InitArray
5  {
6     // create and output two-dimensional arrays
7     public static void main( String[] args )
8     {
9        int[][] array1 = { { 1, 2, 3 }, { 4, 5, 6 } };
10       int[][] array2 = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
11
12       System.out.println( "Values in array1 by row are" );
13       outputArray( array1 ); // displays array1 by row
14
15       System.out.println( "\nValues in array2 by row are" );
16       outputArray( array2 ); // displays array2 by row
17    } // end main
18
```

array1 has two rows, each with three columns

array2 has three rows with two, one and three columns, respectively

Fig. 7.17 | Initializing two-dimensional arrays. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
19      // output rows and columns of a two-dimensional array
20      public static void outputArray( int[][] array )
21      {
22         // loop through array's rows
23         for ( int row = 0; row < array.length; row++ )
24         {
25            // loop through columns of current row
26            for ( int column = 0; column < array[ row ].length; column++ )
27               System.out.printf( "%d  ", array[ row ][ column ] );
28
29            System.out.println(); // start new line of output
30         } // end outer for
31      } // end method outputArray
32   } // end class InitArray
```

Method can received any two-dimensional array of `int` values

Outer loop determines number of rows in the array

Inner loop determines number of columns in the *current row*

```
Values in array1 by row are
1  2  3
4  5  6

Values in array2 by row are
1  2
3
4  5  6
```

Fig. 7.17 | Initializing two-dimensional arrays. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.10 Case Study: Class GradeBook Using a Two-Dimensional Array

- In most semesters, students take several exams.
- Figure 7.18 contains a version of class `GradeBook` that uses a two-dimensional array `grades` to store the grades of a number of students on multiple exams.
  - Each row represents a student's grades for the entire course.
  - Each column represents the grades of all the students who took a particular exam.
- In this example, we use a ten-by-three array containing ten students' grades on three exams.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.18: GradeBook.java
2  // GradeBook class using a two-dimensional array to store grades.
3
4  public class GradeBook
5  {
6     private String courseName; // name of course this grade book represents
7     private int[][] grades; // two-dimensional array of student grades
8
9     // two-argument constructor initializes courseName and grades array
10    public GradeBook( String name, int[][] gradesArray )
11    {
12       courseName = name; // initialize courseName
13       grades = gradesArray; // store grades
14    } // end two-argument GradeBook constructor
15
16    // method to set the course name
17    public void setCourseName( String name )
18    {
19       courseName = name; // store the course name
20    } // end method setCourseName
21
```

Fig. 7.18 | GradeBook class using a two-dimensional array to store grades. (Part 1 of 9.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
22      // method to retrieve the course name
23      public String getCourseName()
24      {
25         return courseName;
26      } // end method getCourseName
27
28      // display a welcome message to the GradeBook user
29      public void displayMessage()
30      {
31         // getCourseName gets the name of the course
32         System.out.printf( "Welcome to the grade book for\n%s!\n\n",
33            getCourseName() );
34      } // end method displayMessage
35
```

Fig. 7.18 | `GradeBook` class using a two-dimensional array to store grades. (Part 2 of 9.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
36      // perform various operations on the data
37      public void processGrades()
38      {
39         // output grades array
40         outputGrades();
41
42         // call methods getMinimum and getMaximum
43         System.out.printf( "\n%s %d\n%s %d\n\n",
44            "Lowest grade in the grade book is", getMinimum(),
45            "Highest grade in the grade book is", getMaximum() );
46
47         // output grade distribution chart of all grades on all tests
48         outputBarChart();
49      } // end method processGrades
50
```

Fig. 7.18 | `GradeBook` class using a two-dimensional array to store grades. (Part 3 of 9.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
51      // find minimum grade
52      public int getMinimum()
53      {
54         // assume first element of grades array is smallest
55         int lowGrade = grades[ 0 ][ 0 ];
56
57         // loop through rows of grades array
58         for ( int[] studentGrades : grades )
59         {
60            // loop through columns of current row
61            for ( int grade : studentGrades )
62            {
63               // if grade less than lowGrade, assign it to lowGrade
64               if ( grade < lowGrade )
65                  lowGrade = grade;
66            } // end inner for
67         } // end outer for
68
69         return lowGrade; // return lowest grade
70      } // end method getMinimum
71
```

Nested loop searches for lowest grade across all exams. Notice that the outer loop's control variable is a one-dimensional array; Java represents two-dimensional arrays as an array of one-dimensional arrays.

Fig. 7.18 | `GradeBook` class using a two-dimensional array to store grades. (Part 4 of 9.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
72      // find maximum grade
73      public int getMaximum()
74      {
75         // assume first element of grades array is largest
76         int highGrade = grades[ 0 ][ 0 ];
77
78         // loop through rows of grades array
79         for ( int[] studentGrades : grades )
80         {
81            // loop through columns of current row
82            for ( int grade : studentGrades )
83            {
84               // if grade greater than highGrade, assign it to highGrade
85               if ( grade > highGrade )
86                  highGrade = grade;
87            } // end inner for
88         } // end outer for
89
90         return highGrade; // return highest grade
91      } // end method getMaximum
92
```

Nested loop searches for highest grade across all exams

Fig. 7.18 | `GradeBook` class using a two-dimensional array to store grades. (Part 5 of 9.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
93      // determine average grade for particular set of grades
94      public double getAverage( int[] setOfGrades )
95      {
96         int total = 0; // initialize total
97
98         // sum grades for one student
99         for ( int grade : setOfGrades )
100           total += grade;
101
102        // return average of grades
103        return (double) total / setOfGrades.length;
104     } // end method getAverage
105
106     // output bar chart displaying overall grade distribution
107     public void outputBarChart()
108     {
109        System.out.println( "Overall grade distribution:" );
110
111        // stores frequency of grades in each range of 10 grades
112        int[] frequency = new int[ 11 ];
113
```

Method receives one-dimensional array of `int` values and returns the average of one student's grades

Fig. 7.18 | `GradeBook` class using a two-dimensional array to store grades. (Part 6 of 9.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
114         // for each grade in GradeBook, increment the appropriate frequency
115         for ( int[] studentGrades : grades )
116         {
117            for ( int grade : studentGrades )
118               ++frequency[ grade / 10 ];
119         } // end outer for
120
121         // for each grade frequency, print bar in chart
122         for ( int count = 0; count < frequency.length; count++ )
123         {
124            // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
125            if ( count == 10 )
126               System.out.printf( "%5d: ", 100 );
127            else
128               System.out.printf( "%02d-%02d: ",
129                  count * 10, count * 10 + 9  );
130
131            // print bar of asterisks
132            for ( int stars = 0; stars < frequency[ count ]; stars++ )
133               System.out.print( "*" );
134
135            System.out.println(); // start a new line of output
136         } // end outer for
137      } // end method outputBarChart
```

Fig. 7.18 | GradeBook class using a two-dimensional array to store grades. (Part 7 of 9.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
138
139    // output the contents of the grades array
140    public void outputGrades()
141    {
142       System.out.println( "The grades are:\n" );
143       System.out.print( "            " ); // align column heads
144
145       // create a column heading for each of the tests
146       for ( int test = 0; test < grades[ 0 ].length; test++ )
147          System.out.printf( "Test %d  ", test + 1 );
148
149       System.out.println( "Average" ); // student average column heading
150
151       // create rows/columns of text representing array grades
152       for ( int student = 0; student < grades.length; student++ )
153       {
154          System.out.printf( "Student %2d", student + 1 );
155
156          for ( int test : grades[ student ] ) // output student's grades
157             System.out.printf( "%8d", test );
158
```

Fig. 7.18 | `GradeBook` class using a two-dimensional array to store grades. (Part 8 of 9.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
159            // call method getAverage to calculate student's average grade;
160            // pass row of grades as the argument to getAverage
161            double average = getAverage( grades[ student ] );
162            System.out.printf( "%9.2f\n", average );
163         } // end outer for
164      } // end method outputGrades
165   } // end class GradeBook
```

Passes one row of array grades to method `getAverage`

Fig. 7.18 | `GradeBook` class using a two-dimensional array to store grades. (Part 9 of 9.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.19: GradeBookTest.java
2  // GradeBookTest creates GradeBook object using a two-dimensional array
3  // of grades, then invokes method proctessGrades to analyze them.
4  public class GradeBookTest
5  {
6     // main method begins program execution
7     public static void main( String[] args )
8     {
9        // two-dimensional array of student grades
10       int[][] gradesArray = { { 87, 96, 70 },
11                               { 68, 87, 90 },
12                               { 94, 100, 90 },
13                               { 100, 81, 82 },
14                               { 83, 65, 85 },
15                               { 78, 87, 65 },
16                               { 85, 75, 83 },
17                               { 91, 94, 100 },
18                               { 76, 72, 84 },
19                               { 87, 93, 73 } };
20
21       GradeBook myGradeBook = new GradeBook(
22          "CS101 Introduction to Java Programming", gradesArray );
```

Fig. 7.19 | `GradeBookTest` creates `GradeBook` object using a two-dimensional array of grades, then invokes method `processGrades` to analyze them. (Part 1 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
23         myGradeBook.displayMessage();
24         myGradeBook.processGrades();
25      } // end main
26   } // end class GradeBookTest
```

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

The grades are:

            Test 1  Test 2  Test 3  Average
Student  1      87      96      70    84.33
Student  2      68      87      90    81.67
Student  3      94     100      90    94.67
Student  4     100      81      82    87.67
Student  5      83      65      85    77.67
Student  6      78      87      65    76.67
Student  7      85      75      83    81.00
Student  8      91      94     100    95.00
Student  9      76      72      84    77.33
Student 10      87      93      73    84.33

Lowest grade in the grade book is 65
Highest grade in the grade book is 100
```

Fig. 7.19 | `GradeBookTest` creates `GradeBook` object using a two-dimensional array of grades, then invokes method `processGrades` to analyze them. (Part 2 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
Overall grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: ***
70-79: ******
80-89: ***********
90-99: *******
  100: ***
```

Fig. 7.19 | `GradeBookTest` creates `GradeBook` object using a two-dimensional array of grades, then invokes method `processGrades` to analyze them. (Part 3 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.11 Variable-Length Argument Lists

- Variable-length argument lists
  - Can be used to create methods that receive an unspecified number of arguments.
  - Parameter type followed by an ellipsis (. . .) indicates that the method receives a variable number of arguments of that particular type.
  - The ellipsis can occur only once at the end of a parameter list.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.20: VarargsTest.java
2  // Using variable-length argument lists.
3
4  public class VarargsTest
5  {
6     // calculate average
7     public static double average( double... numbers )
8     {
9        double total = 0.0; // initialize total
10
11       // calculate total using the enhanced for statement
12       for ( double d : numbers )
13          total += d;
14
15       return total / numbers.length;
16    } // end method average
17
18    public static void main( String[] args )
19    {
20       double d1 = 10.0;
21       double d2 = 20.0;
22       double d3 = 30.0;
23       double d4 = 40.0;
24
```

Variable number of `double` values can be passed to this method

Variable arguments are automatically placed in an array referenced by the parameter

Fig. 7.20 | Using variable-length argument lists. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
25         System.out.printf( "d1 = %.1f\nd2 = %.1f\nd3 = %.1f\nd4 = %.1f\n\n",
26            d1, d2, d3, d4 );
27
28         System.out.printf( "Average of d1 and d2 is %.1f\n",
29            average( d1, d2 ) );
30         System.out.printf( "Average of d1, d2 and d3 is %.1f\n",
31            average( d1, d2, d3 ) );
32         System.out.printf( "Average of d1, d2, d3 and d4 is %.1f\n",
33            average( d1, d2, d3, d4 ) );
34      } // end main
35   } // end class VarargsTest
```

```
d1 = 10.0
d2 = 20.0
d3 = 30.0
d4 = 40.0

Average of d1 and d2 is 15.0
Average of d1, d2 and d3 is 20.0
Average of d1, d2, d3 and d4 is 25.0
```

Fig. 7.20 | Using variable-length argument lists. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.12 Using Command-Line Arguments

- Command-line arguments
  - Can pass arguments from the command line to an application.
  - Arguments that appear after the class name in the `java` command are received by `main` in the `String` array `args`.
  - The number of command-line arguments is obtained by accessing the array's `length` attribute.
  - Command-line arguments are separated by white space, not commas.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.21: InitArray.java
2  // Initializing an array using command-line arguments.
3
4  public class InitArray
5  {
6     public static void main( String[] args )
7     {
8        // check number of command-line arguments
9        if ( args.length != 3 )
10          System.out.println(
11             "Error: Please re-enter the entire command, including\n" +
12             "an array size, initial value and increment." );
13       else
14       {
15          // get array size from first command-line argument
16          int arrayLength = Integer.parseInt( args[ 0 ] );
17          int[] array = new int[ arrayLength ]; // create array
18
19          // get initial value and increment from command-line arguments
20          int initialValue = Integer.parseInt( args[ 1 ] );
21          int increment = Integer.parseInt( args[ 2 ] );
22
```

Check whether there are 3 command-line arguments

Use first command-line argument as length of array to create

Use second and third command-line arguments as a starting value and increment for the values that will be generated in lines 24–25

Fig. 7.21 | Initializing an array using command-line arguments. (Part 1 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
23            // calculate value for each array element
24            for ( int counter = 0; counter < array.length; counter++ )
25               array[ counter ] = initialValue + increment * counter;
26
27            System.out.printf( "%s%8s\n", "Index", "Value" );
28
29            // display array index and value
30            for ( int counter = 0; counter < array.length; counter++ )
31               System.out.printf( "%5d%8d\n", counter, array[ counter ] );
32         } // end else
33      } // end main
34   } // end class InitArray
```

```
java InitArray
Error: Please re-enter the entire command, including
an array size, initial value and increment.
```

Fig. 7.21 | Initializing an array using command-line arguments. (Part 2 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

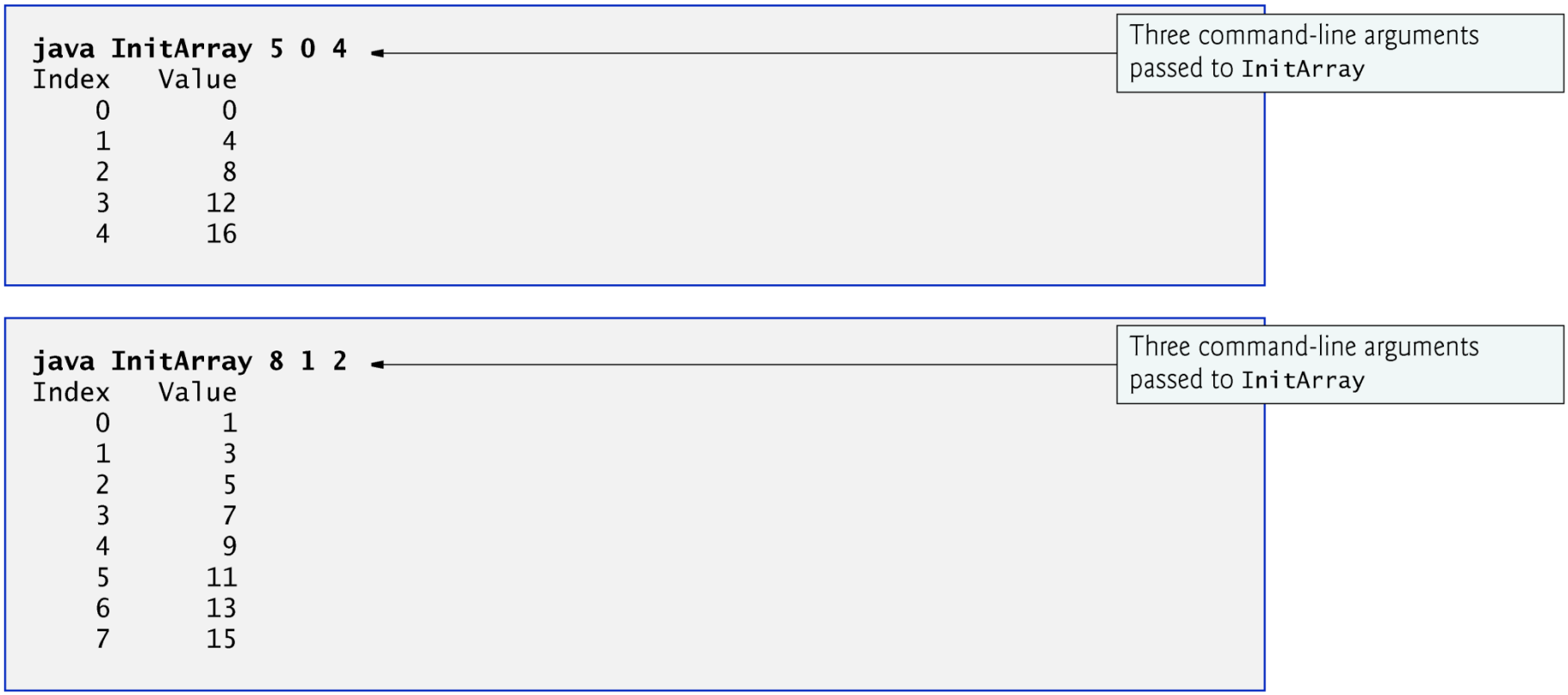

Fig. 7.21 | Initializing an array using command-line arguments. (Part 3 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.13 Class `Arrays`

- `Arrays` class
  - Provides `static` methods for common array manipulations.
- Methods include
  - `sort` for sorting an array (ascending order by default)
  - `binarySearch` for searching a sorted array
  - `equals` for comparing arrays
  - `fill` for placing values into an array.
- Methods are overloaded for primitive-type arrays and for arrays of objects.
- `System` class `static` `arraycopy` method
  - Copies contents of one array into another.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.22: ArrayManipulations.java
2  // Arrays class methods and System.arraycopy.
3  import java.util.Arrays;
4
5  public class ArrayManipulations
6  {
7     public static void main( String[] args )
8     {
9        // sort doubleArray into ascending order
10       double[] doubleArray = { 8.4, 9.3, 0.2, 7.9, 3.4 };
11       Arrays.sort( doubleArray );
12       System.out.printf( "\ndoubleArray: " );
13
14       for ( double value : doubleArray )
15          System.out.printf( "%.1f ", value );
16
17       // fill 10-element array with 7s
18       int[] filledIntArray = new int[ 10 ];
19       Arrays.fill( filledIntArray, 7 );
20       displayArray( filledIntArray, "filledIntArray" );
21
```

Sorts an array's contents into their default sort order

Fills an array's elements with the value specified as the second argument

Fig. 7.22 | `Arrays` class methods. (Part 1 of 4.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
22         // copy array intArray into array intArrayCopy
23         int[] intArray = { 1, 2, 3, 4, 5, 6 };
24         int[] intArrayCopy = new int[ intArray.length ];
25         System.arraycopy( intArray, 0, intArrayCopy, 0, intArray.length );
26         displayArray( intArray, "intArray" );
27         displayArray( intArrayCopy, "intArrayCopy" );
28
29         // compare intArray and intArrayCopy for equality
30         boolean b = Arrays.equals( intArray, intArrayCopy );
31         System.out.printf( "\n\nintArray %s intArrayCopy\n",
32            ( b ? "==" : "!=" ) );
33
34         // compare intArray and filledIntArray for equality
35         b = Arrays.equals( intArray, filledIntArray );
36         System.out.printf( "intArray %s filledIntArray\n",
37            ( b ? "==" : "!=" ) );
38
39         // search intArray for the value 5
40         int location = Arrays.binarySearch( intArray, 5 );
41
42         if ( location >= 0 )
43            System.out.printf(
44               "Found 5 at element %d in intArray\n", location );
```

Copies elements of the array in the first argument, into the array specified as the third argument

Compares contents of two arrays for equality

Compares contents of two arrays for equality

Fig. 7.22 | `Arrays` class methods. (Part 2 of 4.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
45          else
46             System.out.println( "5 not found in intArray" );
47
48          // search intArray for the value 8763
49          location = Arrays.binarySearch( intArray, 8763 );
50
51          if ( location >= 0 )
52             System.out.printf(
53                "Found 8763 at element %d in intArray\n", location );
54          else
55             System.out.println( "8763 not found in intArray" );
56       } // end main
57
58       // output values in each array
59       public static void displayArray( int[] array, String description )
60       {
61          System.out.printf( "\n%s: ", description );
62
63          for ( int value : array )
64             System.out.printf( "%d ", value );
65       } // end method displayArray
66    } // end class ArrayManipulations
```

Searches for second argument in the array specified as the first argument

Fig. 7.22 | `Arrays` class methods. (Part 3 of 4.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```

Fig. 7.22 | `Arrays` class methods. (Part 4 of 4.)

(C) 2010 Pearson Education, Inc. All rights reserved.

7.14 Introduction to Collections and Class `ArrayList`

- Java API provides several predefined data structures, called collections, used to store groups of related objects.
  - Each provides efficient methods that organize, store and retrieve your data without requiring knowledge of how the data is being stored.
  - Reduce application-development time.
- Arrays do not automatically change their size at execution time to accommodate additional elements.
- `ArrayList<T>` (package `java.util`) can dynamically change its size to accommodate more elements.
  - `T` is a placeholder for the type of element stored in the collection.
  - This is similar to specifying the type when declaring an array, except that only nonprimitive types can be used with these collection classes.
- Classes with this kind of placeholder that can be used with any type are called generic classes.

(C) 2010 Pearson Education, Inc. All rights reserved.

Method	Description
`add`	Adds an element to the end of the `ArrayList`.
`clear`	Removes all the elements from the `ArrayList`.
`contains`	Returns `true` if the `ArrayList` contains the specified element; otherwise, returns `false`.
`get`	Returns the element at the specified index.
`indexOf`	Returns the index of the first occurrence of the specified element in the `ArrayList`.
`remove`	Removes the first occurrence of the specified value.
`remove`	Removes the element at the specified index.
`size`	Returns the number of elements stored in the `ArrayList`.
`trimToSize`	Trims the capacity of the `ArrayList` to current number of elements.

Fig. 7.23 | Some methods and properties of class `ArrayList<T>`.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.14 Introduction to Collections and Class `ArrayList` (Cont.)

- Figure 7.24 demonstrates some common `ArrayList` capabilities.
- An `ArrayList`'s capacity indicates how many items it can hold without growing.
- When the `ArrayList` grows, it must create a larger internal array and copy each element to the new array.
  - This is a time-consuming operation. It would be inefficient for the `ArrayList` to grow each time an element is added.
  - An `ArrayList` grows only when an element is added and the number of elements is equal to the capacity—i.e., there is no space for the new element.

(C) 2010 Pearson Education, Inc. All rights reserved.

7.14 Introduction to Collections and Class `ArrayList` (Cont.)

- Method `add` adds elements to the `ArrayList`.
  - One-argument version appends its argument to the end of the `ArrayList`.
  - Two-argument version inserts a new element at the specified position.
  - Collection indices start at zero.
- Method `size` returns the number of elements in the `ArrayList`.
- Method `get` obtains the element at a specified index.
- Method `remove` deletes an element with a specific value.
  - An overloaded version of the method removes the element at the specified index.
- Method `contains` determines if an item is in the `ArrayList`.

(C) 2010 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 7.24: ArrayListCollection.java
2  // Generic ArrayList collection demonstration.
3  import java.util.ArrayList;
4
5  public class ArrayListCollection
6  {
7     public static void main( String[] args )
8     {
9        // create a new ArrayList of Strings
10       ArrayList< String > items = new ArrayList< String >();
11
12       items.add( "red" ); // append an item to the list
13       items.add( 0, "yellow" ); // insert the value at index 0
14
15       // header
16       System.out.print(
17          "Display list contents with counter-controlled loop:" );
18
19       // display the colors in the list
20       for ( int i = 0; i < items.size(); i++ )
21          System.out.printf( " %s", items.get( i ) );
22
```

Creates an `ArrayList` that stores `String` elements

Add elements the the `ArrayList`

Method `size` returns the number of elements in the collection; method `get` returns the element at the specified index

Fig. 7.24 | Generic `ArrayList<T>` collection demonstration. (Part 1 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
23          // display colors using foreach in the display method
24          display( items,
25             "\nDisplay list contents with enhanced for statement:" );
26
27          items.add( "green" ); // add "green" to the end of the list
28          items.add( "yellow" ); // add "yellow" to the end of the list
29          display( items, "List with two new elements:" );
30
31          items.remove( "yellow" ); // remove the first "yellow"
32          display( items, "Remove first instance of yellow:" );
33
34          items.remove( 1 ); // remove item at index 1
35          display( items, "Remove second list element (green):" );
36
37          // check if a value is in the List
38          System.out.printf( "\"red\" is %sin the list\n",
39             items.contains( "red" ) ? "": "not " );
40
41          // display number of elements in the List
42          System.out.printf( "Size: %s\n", items.size() );
43       } // end main
44
```

Method `remove` deletes the first occurrence of the specified value

This version of `remove` deletes the element at the specified index

Method `contains` determines whether the specified value is in the collection

Fig. 7.24 | Generic `ArrayList<T>` collection demonstration. (Part 2 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

```
45      // display the ArrayList's elements on the console
46      public static void display( ArrayList< String > items, String header )
47      {
48         System.out.print( header ); // display header
49
50         // display each element in items
51         for ( String item : items )
52            System.out.printf( " %s", item );
53
54         System.out.println(); // display end of line
55      } // end method display
56   } // end class ArrayListCollection
```

Can use the enhanced `for` statement with collections

```
Display list contents with counter-controlled loop: yellow red
Display list contents with enhanced for statement: yellow red
List with two new elements: yellow red green yellow
Remove first instance of yellow: red green yellow
Remove second list element (green): red yellow
"red" is in the list
Size: 2
```

Fig. 7.24 | Generic `ArrayList<T>` collection demonstration. (Part 3 of 3.)

(C) 2010 Pearson Education, Inc. All rights reserved.

Exercises: Random number generator

- Program that that simulates tossing a coin.
- Application to play a game of guess the number.

(C) 2010 Pearson Education, Inc. All rights reserved.

End of Part 2

(C) 2010 Pearson Education, Inc. All rights reserved.