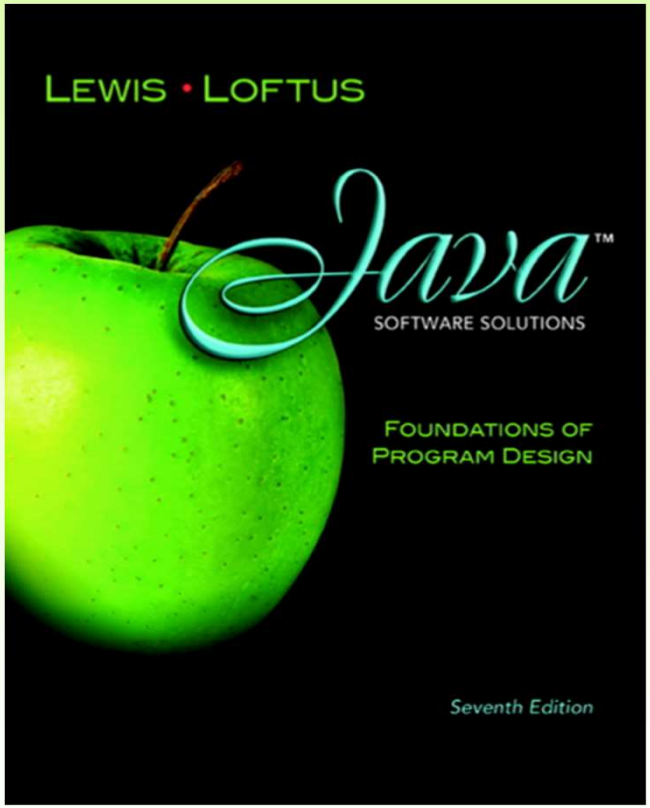
Chapter 8
Arrays



Java Software Solutions
Foundations of Program Design
Seventh Edition

John Lewis
William Loftus

Addison-Wesley
is an imprint of
PEARSON

Arrays

- Arrays are objects that help us organize large amounts of information
- Chapter 8 focuses on:
  - array declaration and use
  - bounds checking and capacity
  - arrays that store object references
  - variable length parameter lists
  - multidimensional arrays
  - polygons and polylines
  - mouse events and keyboard events

Outline

- **Declaring and Using Arrays**
- **Arrays of Objects**
- **Variable Length Parameter Lists**
- **Two-Dimensional Arrays**
- **Polygons and Polylines**
- **Mouse Events and Key Events**

Arrays

- The `ArrayList` class, introduced in Chapter 5, is used to organize a list of objects
- It is a class in the Java API
- An *array* is a programming language construct used to organize a list of objects
- It has special syntax to access elements
- As its name implies, the `ArrayList` class uses an array internally to manage the list of objects

Arrays

- An array is an ordered list of values:

The entire array has a single name

Each value has a numeric *index*

	0	1	2	3	4	5	6	7	8	9
`scores`	79	87	94	82	67	98	87	81	74	91

An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Arrays

- A particular value in an array is referenced using the array name followed by the index in brackets

- For example, the expression

```
scores[2]
```

  refers to the value `94` (the 3rd value in the array)

- That expression represents a place to store a single integer and can be used wherever an integer variable can be used

Arrays

- For example, an array element can be assigned a value, printed, or used in a calculation:

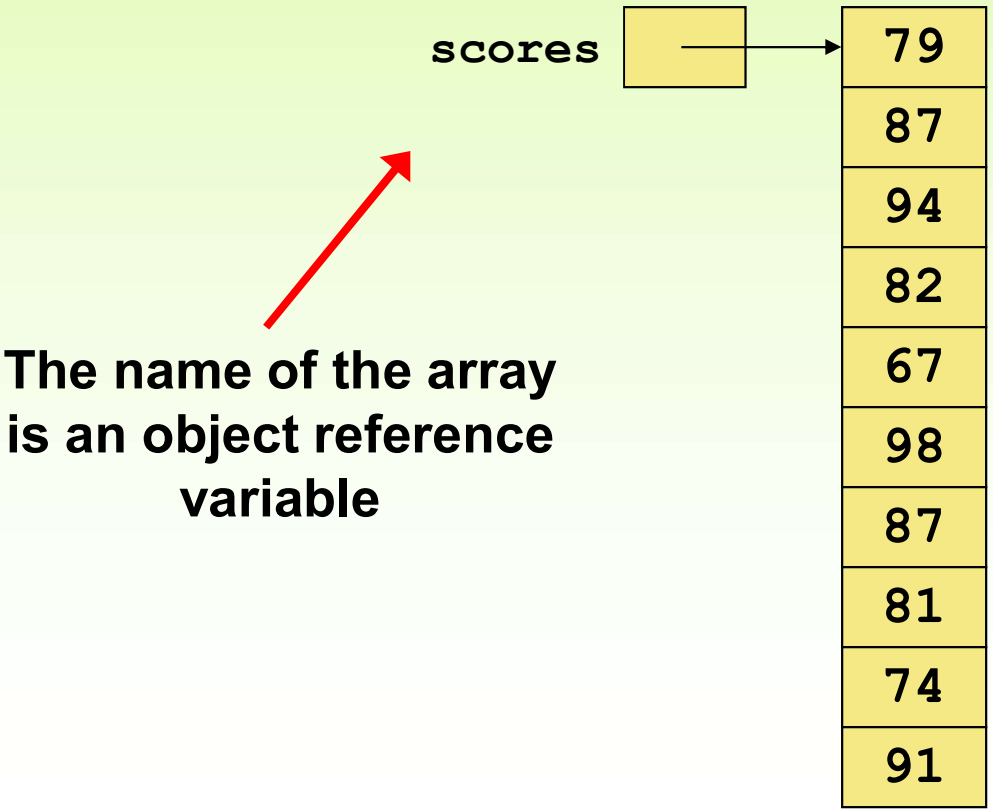
```
scores[2] = 89;

scores[first] = scores[first] + 2;

mean = (scores[0] + scores[1])/2;

System.out.println ("Top = " + scores[5]);

pick = scores[rand.nextInt(11)];
```

Arrays

- The values held in an array are called *array elements*
- An array stores multiple values of the same type – the *element type*
- The element type can be a primitive type or an object reference
- Therefore, we can create an array of integers, an array of characters, an array of `String` objects, an array of `Coin` objects, etc.

Arrays

- In Java, the array itself is an object that must be instantiated
- Another way to depict the `scores` array:



Declaring Arrays

- The `scores` array could be declared as follows:

```
int[] scores = new int[10];
```

- The type of the variable `scores` is `int[]` (an array of integers)
- Note that the array type does not specify its size, but each object of that type has a specific size
- The reference variable `scores` is set to a new array object that can hold 10 integers

Declaring Arrays

- Some other examples of array declarations:

```
int[] weights = new int[2000];

double[] prices = new double[500];

boolean[] flags;
flags = new boolean[20];

char[] codes = new char[1750];
```

Using Arrays

- The for-each version of the `for` loop can be used when processing array elements:

```
for (int score : scores)
   System.out.println (score);
```

- This is only appropriate when processing all array elements starting at index 0
- It can't be used to set the array values
- See `BasicArray.java`

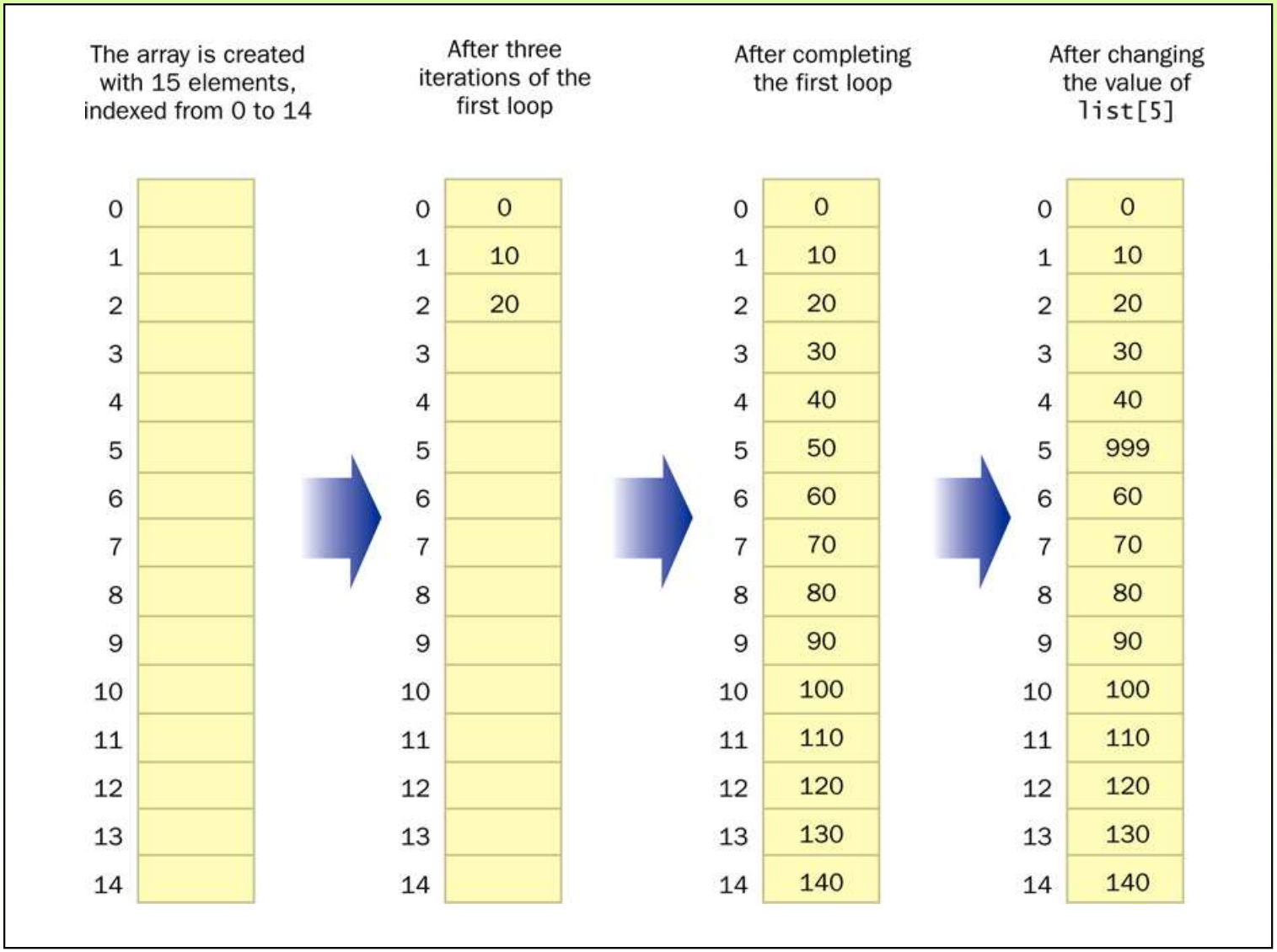
```java
//*******************************************************************
//  BasicArray.java       Author: Lewis/Loftus
//
//  Demonstrates basic array declaration and use.
//*******************************************************************

public class BasicArray
{
   //-----------------------------------------------------------------
   //  Creates an array, fills it with various integer values,
   //  modifies one value, then prints them out.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int LIMIT = 15, MULTIPLE = 10;

      int[] list = new int[LIMIT];

      //  Initialize the array values
      for (int index = 0; index < LIMIT; index++)
         list[index] = index * MULTIPLE;

      list[5] = 999;  // change one array value

      //  Print the array values
      for (int value : list)
         System.out.print (value + "  ");
   }
}
```

Output

```
0   10   20   30   40   999   60   70   80   90   100   110   120   130   140
```

```java
//*******************************************************************

public class BasicArray
{
   //-----------------------------------------------------------------
   //  Creates an array, fills it with various integer values,
   //  modifies one value, then prints them out.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int LIMIT = 15, MULTIPLE = 10;

      int[] list = new int[LIMIT];

      //  Initialize the array values
      for (int index = 0; index < LIMIT; index++)
         list[index] = index * MULTIPLE;

      list[5] = 999;  // change one array value

      //  Print the array values
      for (int value : list)
         System.out.print (value + "  ");
   }
}
```

Basic Array Example



Quick Check

Write an array declaration to represent the ages of 100 children.

Write code that prints each value in an array of integers named `values`.

Quick Check

Write an array declaration to represent the ages of 100 children.

```
int[] ages = new int[100];
```

Write code that prints each value in an array of integers named `values`.

```
for (int value : values)
    System.out.println(value);
```

Bounds Checking

- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element
- That is, the index value must be in range 0 to N-1
- The Java interpreter throws an `ArrayIndexOutOfBoundsException` if an array index is out of bounds
- This is called automatic *bounds checking*

Bounds Checking

- For example, if the array `codes` can hold 100 values, it can be indexed from 0 to 99
- If the value of `count` is 100, then the following reference will cause an exception to be thrown:

```
System.out.println(codes[count]);
```

- It's common to introduce *off-by-one errors* when using arrays:

problem

```
for (int index=0; index <= 100; index++)
   codes[index] = index*50 + epsilon;
```

Bounds Checking

- Each array object has a public constant called `length` that stores the size of the array
- It is referenced using the array name:

```
scores.length
```

- Note that `length` holds the number of elements, not the largest index
- See `ReverseOrder.java`
- See `LetterCount.java`

```java
//*****************************************************************
//  ReverseOrder.java       Author: Lewis/Loftus
//
//  Demonstrates array index processing.
//*****************************************************************

import java.util.Scanner;

public class ReverseOrder
{
   //-----------------------------------------------------------------
   //  Reads a list of numbers from the user, storing them in an
   //  array, then prints them in the opposite order.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Scanner scan = new Scanner (System.in);

      double[] numbers = new double[10];

      System.out.println ("The size of the array: " + numbers.length);
```

continue

continue

```
      for (int index = 0; index < numbers.length; index++)
      {
         System.out.print ("Enter number " + (index+1) + ": ");
         numbers[index] = scan.nextDouble();
      }

      System.out.println ("The numbers in reverse order:");

      for (int index = numbers.length-1; index >= 0; index--)
         System.out.print (numbers[index] + "  ");
   }
}
```

Sample Run

```
The size of the array: 10
Enter number 1: 18.36
Enter number 2: 48.9
Enter number 3: 53.5
Enter number 4: 29.06
Enter number 5: 72.404
Enter number 6: 34.8
Enter number 7: 63.41
Enter number 8: 45.55
Enter number 9: 69.0
Enter number 10: 99.18
The numbers in reverse order:
99.18  69.0  45.55  63.41  34.8  72.404  29.06  53.5  48.9  18.36
```

```java
//*****************************************************************
//  LetterCount.java       Author: Lewis/Loftus
//
//  Demonstrates the relationship between arrays and strings.
//*****************************************************************

import java.util.Scanner;

public class LetterCount
{
   //-----------------------------------------------------------------
   //  Reads a sentence from the user and counts the number of
   //  uppercase and lowercase letters contained in it.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int NUMCHARS = 26;

      Scanner scan = new Scanner (System.in);

      int[] upper = new int[NUMCHARS];
      int[] lower = new int[NUMCHARS];

      char current;   // the current character being processed
      int other = 0;  // counter for non-alphabetics
```

continue

continue

```
System.out.println ("Enter a sentence:");
String line = scan.nextLine();

//  Count the number of each letter occurence
for (int ch = 0; ch < line.length(); ch++)
{
   current = line.charAt(ch);
   if (current >= 'A' && current <= 'Z')
      upper[current-'A']++;
   else
      if (current >= 'a' && current <= 'z')
         lower[current-'a']++;
      else
         other++;
}
```

continue

continue

```
      //  Print the results
      System.out.println ();
      for (int letter=0; letter < upper.length; letter++)
      {
         System.out.print ( (char) (letter + 'A') );
         System.out.print (": " + upper[letter]);
         System.out.print ("\t\t" + (char) (letter + 'a') );
         System.out.println (": " + lower[letter]);
      }

      System.out.println ();
      System.out.println ("Non-alphabetic characters: " + other);
   }
}
```

Sample Run

```
Enter a sentence:
In Casablanca, Humphrey Bogart never says "Play it again, Sam."

A: 0            a: 10
B: 1            b: 1
C: 1            c: 1
D: 0            d: 0
E: 0            e: 3
F: 0            f: 0
G: 0            g: 2
H: 1            h: 1
I: 1            i: 2
J: 0            j: 0
K: 0            k: 0
L: 0            l: 2
M: 0            m: 2
N: 0            n: 4
O: 0            o: 1
P: 1            p: 1
Q: 0            q: 0
```

continue

Sample Run (continued)

```
R: 0            r: 3
S: 1            s: 3
T: 0            t: 2
U: 0            u: 1
V: 0            v: 1
W: 0            w: 0
X: 0            x: 0
Y: 0            y: 3
Z: 0            z: 0

Non-alphabetic characters: 14
```

Alternate Array Syntax

- The brackets of the array type can be associated with the element type or with the name of the array
- Therefore the following two declarations are equivalent:

```
double[] prices;
double prices[];
```

- The first format generally is more readable and should be used

Initializer Lists

- An *initializer list* can be used to instantiate and fill an array in one step
- The values are delimited by braces and separated by commas
- Examples:

```
int[] units = {147, 323, 89, 933, 540,
               269, 97, 114, 298, 476};

char[] grades = {'A', 'B', 'C', 'D', ’F'};
```

Initializer Lists

- Note that when an initializer list is used:
  - the `new` operator is not used
  - no size value is specified
- The size of the array is determined by the number of items in the list
- An initializer list can be used only in the array declaration
- See `Primes.java`

```java
//*****************************************************************
//  Primes.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an initializer list for an array.
//*****************************************************************

public class Primes
{
   //-----------------------------------------------------------------
   //  Stores some prime numbers in an array and prints them.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int[] primeNums = {2, 3, 5, 7, 11, 13, 17, 19};

      System.out.println ("Array length: " + primeNums.length);

      System.out.println ("The first few prime numbers are:");

      for (int prime : primeNums)
         System.out.print (prime + "  ");
   }
}
```

```
//*************************************************************
//  Primes.java
//
//  Demonstrate array.
//*************************************************************

public class Primes
{
   //-----------------------------------------------------------
   //  Stores some prime numbers in an array and prints them.
   //-----------------------------------------------------------
   public static void main (String[] args)
   {
      int[] primeNums = {2, 3, 5, 7, 11, 13, 17, 19};

      System.out.println ("Array length: " + primeNums.length);

      System.out.println ("The first few prime numbers are:");

      for (int prime : primeNums)
         System.out.print (prime + "  ");
   }
}
```

Output

```
Array length: 8
The first few prime numbers are:
2  3  5  7  11  13  17  19
```

Arrays as Parameters

- An entire array can be passed as a parameter to a method
- Like any other object, the reference to the array is passed, making the formal and actual parameters aliases of each other
- Therefore, changing an array element within the method changes the original
- An individual array element can be passed to a method as well, in which case the type of the formal parameter is the same as the element type

Outline

Declaring and Using Arrays

Arrays of Objects

Variable Length Parameter Lists

Two-Dimensional Arrays

Polygons and Polylines

Mouse Events and Key Events

Arrays of Objects

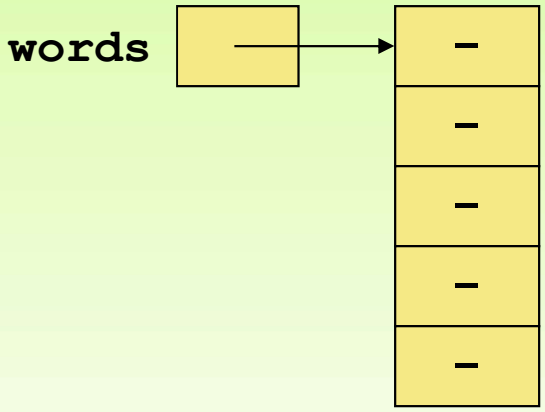
- The elements of an array can be object references
- The following declaration reserves space to store 5 references to `String` objects

```
String[] words = new String[5];
```

- It does NOT create the `String` objects themselves
- Initially an array of objects holds `null` references
- Each object stored in an array must be instantiated separately

Arrays of Objects

- The `words` array when initially declared:

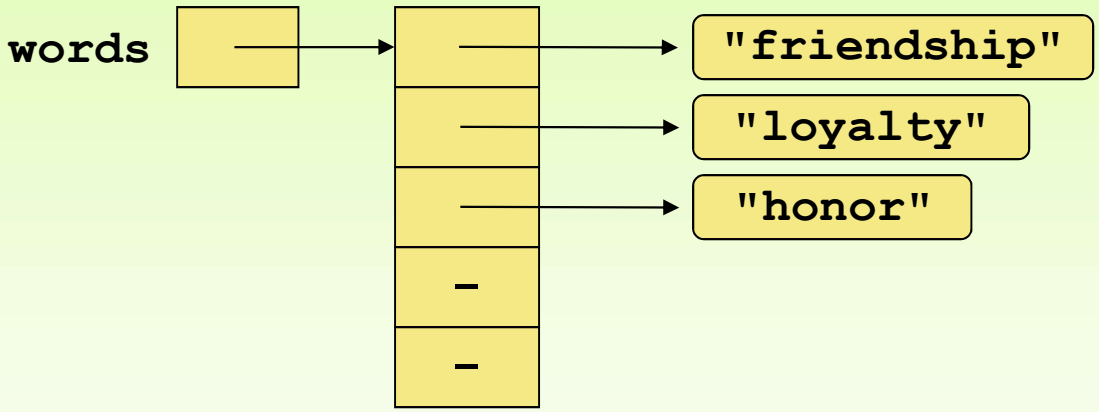


- At this point, the following line of code would throw a `NullPointerException`:

```
System.out.println(words[0]);
```

Arrays of Objects

- After some `String` objects are created and stored in the array:



Arrays of Objects

- Keep in mind that `String` objects can be created using literals
- The following declaration creates an array object called `verbs` and fills it with four `String` objects created using string literals

```
String[] verbs = {"play", "work", "eat",
                  "sleep", "run"};
```

Arrays of Objects

- The following example creates an array of `Grade` objects, each with a string representation and a numeric lower bound
- The letter grades include plus and minus designations, so must be stored as strings instead of `char`
- See `GradeRange.java`
- See `Grade.java`

```java
//*****************************************************************
//  GradeRange.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an array of objects.
//*****************************************************************

public class GradeRange
{
   //-----------------------------------------------------------------
   //  Creates an array of Grade objects and prints them.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Grade[] grades =
      {
         new Grade("A", 95), new Grade("A-", 90),
         new Grade("B+", 87), new Grade("B", 85), new Grade("B-", 80),
         new Grade("C+", 77), new Grade("C", 75), new Grade("C-", 70),
         new Grade("D+", 67), new Grade("D", 65), new Grade("D-", 60),
         new Grade("F", 0)
      };

      for (Grade letterGrade : grades)
         System.out.println (letterGrade);
   }
}
```

```
//*********************************************************************
//  GradeRange.java       Author: Lewis/Loftus
//
//  Demonstrates the use of enumerated objects.
//*********************************************************************

public class GradeRange
{
   //-----------------------------------------------------------------
   //  Creates an array of grades and prints them.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Grade[] grades =
      {
         new Grade("A", 95), new Grade("A-", 90),
         new Grade("B+", 87), new Grade("B", 85), new Grade("B-", 80),
         new Grade("C+", 77), new Grade("C", 75), new Grade("C-", 70),
         new Grade("D+", 67), new Grade("D", 65), new Grade("D-", 60),
         new Grade("F", 0)
      };

      for (Grade letterGrade : grades)
         System.out.println (letterGrade);
   }
}
```

Output

```
A       95
A-      90
B+      87
B       85
B-      80
C+      77
C       75
C-      70
D+      67
D       65
D-      60
F       0
```

```java
//*******************************************************************
//  Grade.java       Author: Lewis/Loftus
//
//  Represents a school grade.
//*******************************************************************

public class Grade
{
   private String name;
   private int lowerBound;

   //-----------------------------------------------------------------
   //  Constructor: Sets up this Grade object with the specified
   //  grade name and numeric lower bound.
   //-----------------------------------------------------------------
   public Grade (String grade, int cutoff)
   {
      name = grade;
      lowerBound = cutoff;
   }

   //-----------------------------------------------------------------
   //  Returns a string representation of this grade.
   //-----------------------------------------------------------------
   public String toString()
   {
      return name + "\t" + lowerBound;
   }
```

continue

continue

```
//-----------------------------------------------------------------
//  Name mutator.
//-----------------------------------------------------------------
public void setName (String grade)
{
   name = grade;
}

//-----------------------------------------------------------------
//  Lower bound mutator.
//-----------------------------------------------------------------
public void setLowerBound (int cutoff)
{
   lowerBound = cutoff;
}
```

continue

continue

```
    //-----------------------------------------------------------------
    //  Name accessor.
    //-----------------------------------------------------------------
    public String getName()
    {
       return name;
    }

    //-----------------------------------------------------------------
    //  Lower bound accessor.
    //-----------------------------------------------------------------
    public int getLowerBound()
    {
       return lowerBound;
    }
}
```

Arrays of Objects

- Now let's look at an example that manages a collection of `DVD` objects
- An initial capacity of 100 is created for the collection
- If more room is needed, a private method is used to create a larger array and transfer the current DVDs
- See `Movies.java`
- See `DVDCollection.java`
- See `DVD.java`

```java
//*****************************************************************
//  Movies.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an array of objects.
//*****************************************************************

public class Movies
{
   //-----------------------------------------------------------------
   //  Creates a DVDCollection object and adds some DVDs to it. Prints
   //  reports on the status of the collection.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      DVDCollection movies = new DVDCollection();

      movies.addDVD ("The Godfather", "Francis Ford Coppala", 1972, 24.95, true);
      movies.addDVD ("District 9", "Neill Blomkamp", 2009, 19.95, false);
      movies.addDVD ("Iron Man", "Jon Favreau", 2008, 15.95, false);
      movies.addDVD ("All About Eve", "Joseph Mankiewicz", 1950, 17.50, false);
      movies.addDVD ("The Matrix", "Andy & Lana Wachowski", 1999, 19.95, true);

      System.out.println (movies);

      movies.addDVD ("Iron Man 2", "Jon Favreau", 2010, 22.99, false);
      movies.addDVD ("Casablanca", "Michael Curtiz", 1942, 19.95, false);

      System.out.println (movies);
   }
}
```

Output

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
My DVD Collection

Number of DVDs: 5
Total cost: $98.30
Average cost: $19.66

DVD List:

$24.95  1972    The Godfather  Francis Ford Coppala   Blu-Ray
$19.95  2009    District 9     Neill Blomkamp
$15.95  2008    Iron Man       Jon Favreau
$17.50  1950    All About Eve  Joseph Mankiewicz
$19.95  1999    The Matrix     Andy & Lana Wachowski  Blu-Ray
```

continue

```
        System.out.println (movies);

        movies.addDVD ("Iron Man 2", "Jon Favreau", 2010, 22.99, false);
        movies.addDVD ("Casablanca", "Michael Curtiz", 1942, 19.95, false);

        System.out.println (movies);
    }
}
```

```
//**
//
//
//
//**

publ
{
    /
    /
    /
    /
    p
    {
```

Output

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
My D

Numb
Tota
Aver

DVD

$24.
$19.
$15.
$17.
$19.
```

cont

```
Sys

mov
mov

System.out.println (movies);
}
}
```

Output (continued)

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
My DVD Collection

Number of DVDs: 7
Total cost: $141.24
Average cost: $20.18

DVD List:

$24.95  1972   The Godfather  Francis Ford Coppala   Blu-Ray
$19.95  2009   District 9     Neill Blomkamp
$15.95  2008   Iron Man       Jon Favreau
$17.50  1950   All About Eve  Joseph Mankiewicz
$19.95  1999   The Matrix     Andy & Lana Wachowski  Blu-Ray
$22.99  2010   Iron Man 2     Jon Favreau
$19.95  1942   Casablanca     Michael Curtiz
```

```java
//*****************************************************************
//  DVDCollection.java       Author: Lewis/Loftus
//
//  Represents a collection of DVD movies.
//*****************************************************************

import java.text.NumberFormat;

public class DVDCollection
{
   private DVD[] collection;
   private int count;
   private double totalCost;

   //-----------------------------------------------------------------
   //  Constructor: Creates an initially empty collection.
   //-----------------------------------------------------------------
   public DVDCollection ()
   {
      collection = new DVD[100];
      count = 0;
      totalCost = 0.0;
   }
```

continue

continue

```
//-----------------------------------------------------------------
//  Adds a DVD to the collection, increasing the size of the
//  collection array if necessary.
//-----------------------------------------------------------------
public void addDVD (String title, String director, int year,
   double cost, boolean bluRay)
{
   if (count == collection.length)
      increaseSize();

   collection[count] = new DVD (title, director, year, cost, bluRay);
   totalCost += cost;
   count++;
}
```

continue

continue

```
//-----------------------------------------------------------------
//  Returns a report describing the DVD collection.
//-----------------------------------------------------------------
public String toString()
{
   NumberFormat fmt = NumberFormat.getCurrencyInstance();

   String report = "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
   report += "My DVD Collection\n\n";

   report += "Number of DVDs: " + count + "\n";
   report += "Total cost: " + fmt.format(totalCost) + "\n";
   report += "Average cost: " + fmt.format(totalCost/count);

   report += "\n\nDVD List:\n\n";

   for (int dvd = 0; dvd < count; dvd++)
      report += collection[dvd].toString() + "\n";

   return report;
}
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Increases the capacity of the collection by creating a
   //  larger array and copying the existing collection into it.
   //-----------------------------------------------------------------
   private void increaseSize ()
   {
      DVD[] temp = new DVD[collection.length * 2];

      for (int dvd = 0; dvd < collection.length; dvd++)
         temp[dvd] = collection[dvd];

      collection = temp;
   }
}
```

```java
//*****************************************************************
//   DVD.java        Author: Lewis/Loftus
//
//   Represents a DVD video disc.
//*****************************************************************

import java.text.NumberFormat;

public class DVD
{
   private String title, director;
   private int year;
   private double cost;
   private boolean bluRay;

   //-----------------------------------------------------------------
   //  Creates a new DVD with the specified information.
   //-----------------------------------------------------------------
   public DVD (String title, String director, int year, double cost,
      boolean bluRay)
   {
      this.title = title;
      this.director = director;
      this.year = year;
      this.cost = cost;
      this.bluRay = bluRay;
   }
```

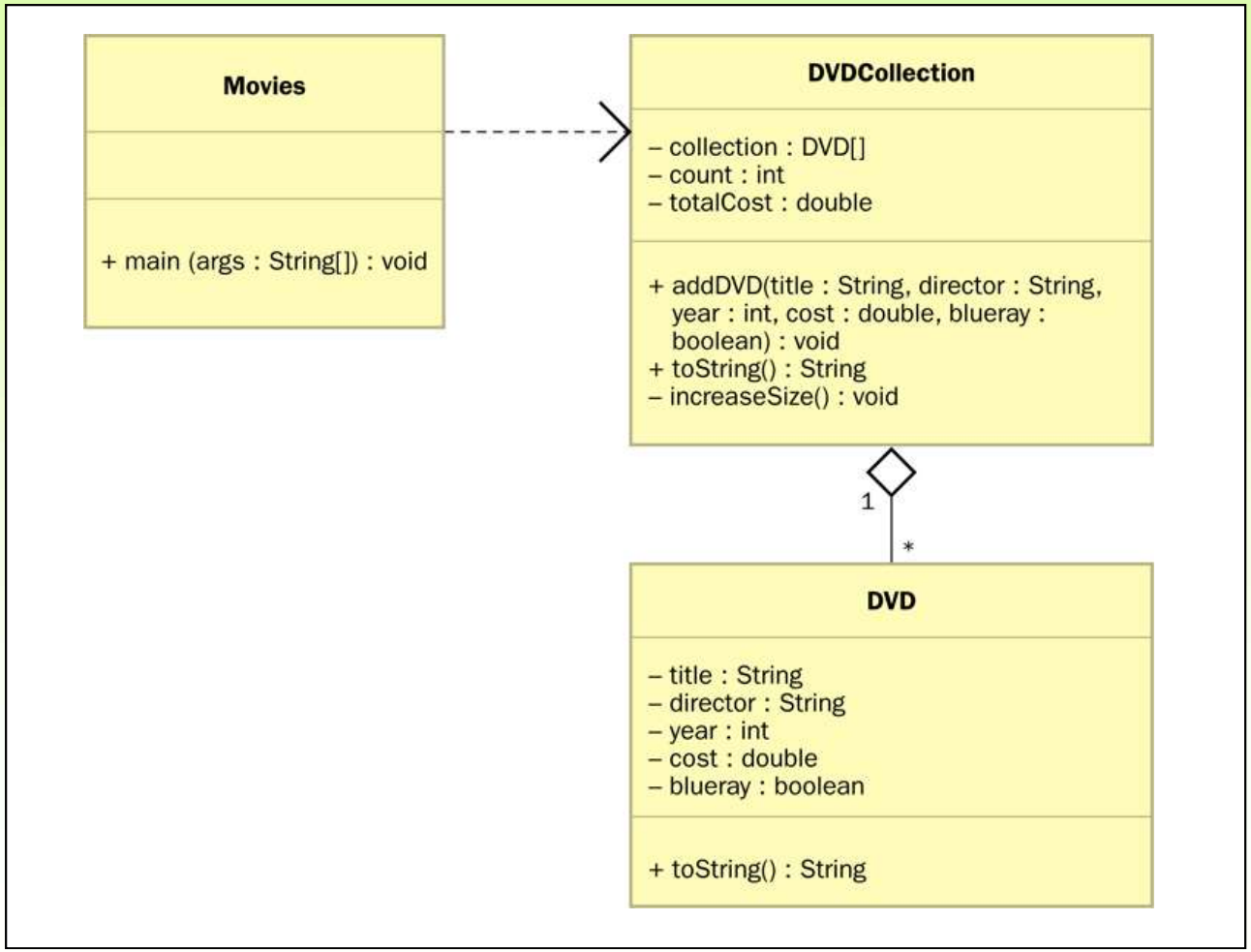
continue

continue

```
   //-----------------------------------------------------------------
   //  Returns a string description of this DVD.
   //-----------------------------------------------------------------
   public String toString()
   {
      NumberFormat fmt = NumberFormat.getCurrencyInstance();

      String description;

      description = fmt.format(cost) + "\t" + year + "\t";
      description += title + "\t" + director;

      if (bluRay)
         description += "\t" + "Blu-Ray";

      return description;
   }
}
```

Arrays of Objects

- A UML diagram for the `Movies` program:



Command-Line Arguments

- The signature of the `main` method indicates that it takes an array of `String` objects as a parameter
- These values come from *command-line arguments* that are provided when the interpreter is invoked
- For example, the following invocation of the interpreter passes three `String` objects into the `main` method of the `StateEval` program:

```
java StateEval pennsylvania texas arizona
```

- See `NameTag.java`

```java
//*****************************************************************
//  NameTag.java       Author: Lewis/Loftus
//
//  Demonstrates the use of command line arguments.
//*****************************************************************

public class NameTag
{
   //-----------------------------------------------------------------
   //  Prints a simple name tag using a greeting and a name that is
   //  specified by the user.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ();
      System.out.println ("     " + args[0]);
      System.out.println ("My name is " + args[1]);
   }
}
```

```
//************                                                    *****************
//  NameTag.ja
//
//  Demonstrat
//************                                                    *****************

public class N
{
   //---------                                                    ----------------
   //  Prints                                                     a name that is
   //  specifi
   //---------                                                    ----------------
   public stat
   {
      System.out.println ();
      System.out.println ("     " + args[0]);
      System.out.println ("My name is " + args[1]);
   }
}
```

Command-Line Execution

```
> java NameTag Howdy John

     Howdy
My name is John

> java NameTag Hello Bill

     Hello
My name is Bill
```

Outline

Declaring and Using Arrays

Arrays of Objects

Variable Length Parameter Lists

Two-Dimensional Arrays

Polygons and Polylines

Mouse Events and Key Events

Variable Length Parameter Lists

- Suppose we wanted to create a method that processed a different amount of data from one invocation to the next
- For example, let's define a method called `average` that returns the average of a set of integer parameters

```
// one call to average three values
mean1 = average (42, 69, 37);

// another call to average seven values
mean2 = average (35, 43, 93, 23, 40, 21, 75);
```

Variable Length Parameter Lists

- We could define overloaded versions of the `average` method
  - Downside: we'd need a separate version of the method for each additional parameter
- We could define the method to accept an array of integers
  - Downside: we'd have to create the array and store the integers prior to calling the method each time
- Instead, Java provides a convenient way to create *variable length parameter lists*

Variable Length Parameter Lists

- Using special syntax in the formal parameter list, we can define a method to accept any number of parameters of the same type
- For each call, the parameters are automatically put into an array for easy processing in the method

Indicates a variable length parameter list

```
public double average (int ... list)
{
  // whatever
}
```

element type

array name

Variable Length Parameter Lists

```
public double average (int ... list)
{
   double result = 0.0;

   if (list.length != 0)
   {
      int sum = 0;
      for (int num : list)
         sum += num;
      result = (double)num / list.length;
   }

   return result;
}
```

Variable Length Parameter Lists

- The type of the parameter can be any primitive or object type:

```
public void printGrades (Grade ... grades)
{
   for (Grade letterGrade : grades)
      System.out.println (letterGrade);
}
```

Quick Check

Write method called `distance` that accepts a variable number of integers (which each represent the distance of one leg of a trip) and returns the total distance of the trip.

Quick Check

Write method called `distance` that accepts a variable number of integers (which each represent the distance of one leg of a trip) and returns the total distance of the trip.

```
public int distance (int ... list)
{
    int sum = 0;
    for (int num : list)
        sum = sum + num;
    return sum;
}
```

Variable Length Parameter Lists

- A method that accepts a variable number of parameters can also accept other parameters
- The following method accepts an `int`, a `String` object, and a variable number of `double` values into an array called `nums`

```
public void test (int count, String name,
                  double ... nums)
{
   // whatever
}
```

Variable Length Parameter Lists

- The varying number of parameters must come last in the formal arguments
- A method cannot accept two sets of varying parameters
- Constructors can also be set up to accept a variable number of parameters
- See `VariableParameters.java`
- See `Family.java`

```java
//*****************************************************************
//  VariableParameters.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a variable length parameter list.
//*****************************************************************

public class VariableParameters
{
   //-----------------------------------------------------------------
   //  Creates two Family objects using a constructor that accepts
   //  a variable number of String objects as parameters.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Family lewis = new Family ("John", "Sharon", "Justin", "Kayla",
         "Nathan", "Samantha");

      Family camden = new Family ("Stephen", "Annie", "Matt", "Mary",
         "Simon", "Lucy", "Ruthie", "Sam", "David");

      System.out.println(lewis);
      System.out.println();
      System.out.println(camden);
   }
}
```

```
//*************************************************************************
//  VariableParameters.java                                  : Lewis/Loftus
//
//  Demonstrates the use of                                  ength parameter list.
//*************************************************************************

public class VariableParame
{
   //-----------------------                                  ----------------------------
   //  Creates two Family c                                   a constructor that accepts
   //  a variable number of                                   ts as parameters.
   //-----------------------                                  ----------------------------
   public static void main                                   s)
   {
      Family lewis = new Fa                                   "Sharon", "Justin", "Kayla",
         "Nathan", "Samanth

      Family camden = new F                                   en", "Annie", "Matt", "Mary",
         "Simon", "Lucy", "                                   ", "David");

      System.out.println(le
      System.out.println();
      System.out.println(camden);
   }
}
```

Output

```
John
Sharon
Justin
Kayla
Nathan
Samantha

Stephen
Annie
Matt
Mary
Simon
Lucy
Ruthie
Sam
David
```

```java
//*******************************************************************
//  Family.java       Author: Lewis/Loftus
//
//  Demonstrates the use of variable length parameter lists.
//*******************************************************************

public class Family
{
   private String[] members;

   //-----------------------------------------------------------------
   //  Constructor: Sets up this family by storing the (possibly
   //  multiple) names that are passed in as parameters.
   //-----------------------------------------------------------------
   public Family (String ... names)
   {
      members = names;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns a string representation of this family.
   //-----------------------------------------------------------------
   public String toString()
   {
      String result = "";

      for (String name : members)
         result += name + "\n";

      return result;
   }
}
```

Outline

Declaring and Using Arrays

Arrays of Objects

Variable Length Parameter Lists

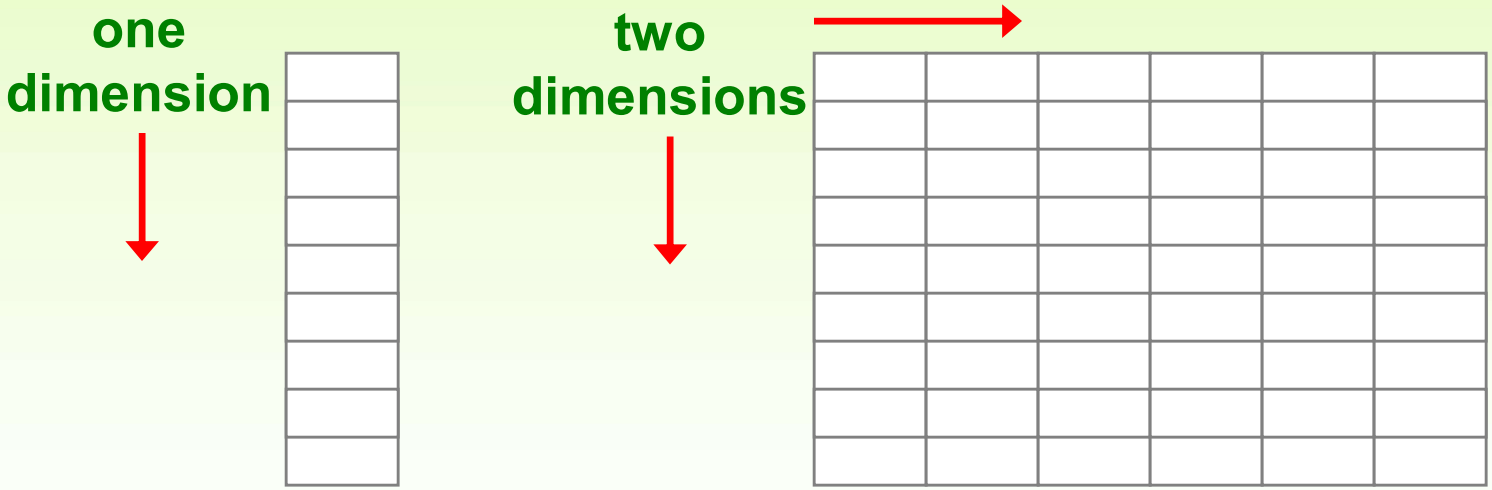
Two-Dimensional Arrays

Polygons and Polylines

Mouse Events and Key Events

Two-Dimensional Arrays

- A *one-dimensional array* stores a list of elements
- A *two-dimensional array* can be thought of as a table of elements, with rows and columns



Two-Dimensional Arrays

- To be precise, in Java a two-dimensional array is an array of arrays
- A two-dimensional array is declared by specifying the size of each dimension separately:

```
int[][] table = new int[12][50];
```

- A array element is referenced using two index values:

```
value = table[3][6]
```

- The array stored in one row can be specified using one index

Two-Dimensional Arrays

Expression	Type	Description
`table`	**`int[][]`**	2D array of integers, or array of integer arrays
`table[5]`	**`int[]`**	array of integers
`table[5][12]`	**`int`**	integer

- See `TwoDArray.java`
- See `SodaSurvey.java`

```java
//*****************************************************************
//  TwoDArray.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a two-dimensional array.
//*****************************************************************

public class TwoDArray
{
   //-----------------------------------------------------------------
   //  Creates a 2D array of integers, fills it with increasing
   //  integer values, then prints them out.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int[][] table = new int[5][10];

      // Load the table with values
      for (int row=0; row < table.length; row++)
         for (int col=0; col < table[row].length; col++)
            table[row][col] = row * 10 + col;

      // Print the table
      for (int row=0; row < table.length; row++)
      {
         for (int col=0; col < table[row].length; col++)
            System.out.print (table[row][col] + "\t");
         System.out.println();
      }
   }
}
```

```
//*******************************************************************
//  TwoDArray.java        Author: Lewis/Loftus
//
```

Output

```
0       1       2       3       4       5       6       7       8       9
10      11      12      13      14      15      16      17      18
        19
20      21      22      23      24      25      26      27      28
        29
30      31      32      33      34      35      36      37      38
        39
40      41      42      43      44      45      46      47      48
        49
```

```
   public static void main (String[] args)
   {
      int[][] table = new int[5][10];

      // Load the table with values
      for (int row=0; row < table.length; row++)
         for (int col=0; col < table[row].length; col++)
            table[row][col] = row * 10 + col;

      // Print the table
      for (int row=0; row < table.length; row++)
      {
         for (int col=0; col < table[row].length; col++)
            System.out.print (table[row][col] + "\t");
         System.out.println();
      }
   }
}
```

```java
//*****************************************************************
//  SodaSurvey.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a two-dimensional array.
//*****************************************************************

import java.text.DecimalFormat;

public class SodaSurvey
{
   //-----------------------------------------------------------------
   //  Determines and prints the average of each row (soda) and each
   //  column (respondent) of the survey scores.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int[][] scores = { {3, 4, 5, 2, 1, 4, 3, 2, 4, 4},
                         {2, 4, 3, 4, 3, 3, 2, 1, 2, 2},
                         {3, 5, 4, 5, 5, 3, 2, 5, 5, 5},
                         {1, 1, 1, 3, 1, 2, 1, 3, 2, 4} };

      final int SODAS = scores.length;
      final int PEOPLE = scores[0].length;

      int[] sodaSum = new int[SODAS];
      int[] personSum = new int[PEOPLE];
```

continue

continue

```
      for (int soda=0; soda < SODAS; soda++)
         for (int person=0; person < PEOPLE; person++)
         {
            sodaSum[soda] += scores[soda][person];
            personSum[person] += scores[soda][person];
         }

      DecimalFormat fmt = new DecimalFormat ("0.#");
      System.out.println ("Averages:\n");

      for (int soda=0; soda < SODAS; soda++)
         System.out.println ("Soda #" + (soda+1) + ": " +
                     fmt.format ((float)sodaSum[soda]/PEOPLE));

      System.out.println ();
      for (int person=0; person < PEOPLE; person++)
         System.out.println ("Person #" + (person+1) + ": " +
                     fmt.format ((float)personSum[person]/SODAS));
   }
}
```

continue

```
        for (int soda=0;
            for (int perso                          person++)
            {
                sodaSum[soc                         son];
                personSum[p                         [person];
            }

        DecimalFormat fmt                           "0.#");
        System.out.printl

        for (int soda=0;
            System.out.pri                          +1) + ": " +
                        fmt                         m[soda]/PEOPLE));

        System.out.printl
        for (int person=0                           son++)
            System.out.pri                          rson+1) + ": " +
                        fmt                         Sum[person]/SODAS));
    }
}
```

Output

```
Averages:

Soda #1: 3.2
Soda #2: 2.6
Soda #3: 4.2
Soda #4: 1.9

Person #1: 2.2
Person #2: 3.5
Person #3: 3.2
Person #4: 3.5
Person #5: 2.5
Person #6: 3
Person #7: 2
Person #8: 2.8
Person #9: 3.2
Person #10: 3.8
```

Multidimensional Arrays

- An array can have many dimensions – if it has more than one dimension, it is called a *multidimensional array*
- Each dimension subdivides the previous one into the specified number of elements
- Each dimension has its own `length` constant
- Because each dimension is an array of array references, the arrays within one dimension can be of different lengths
  - these are sometimes called *ragged arrays*

Outline

Declaring and Using Arrays

Arrays of Objects

Variable Length Parameter Lists

Two-Dimensional Arrays

Polygons and Polylines

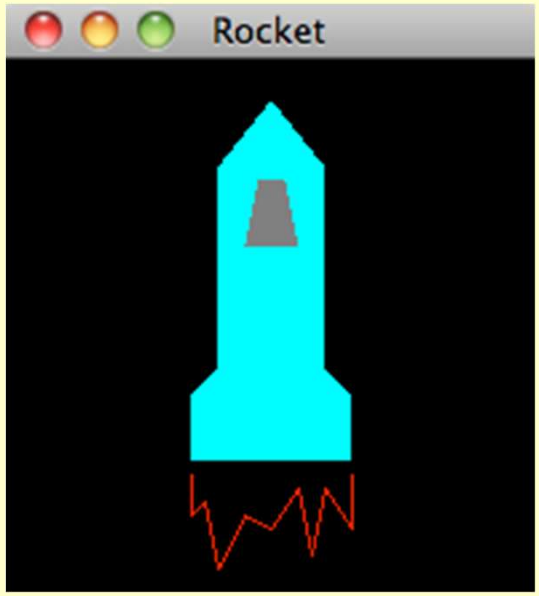
Mouse Events and Key Events

Polygons and Polylines

- Arrays can be helpful in graphics processing
- For example, they can be used to store a list of coordinates
- A *polygon* is a multisided, closed shape
- A *polyline* is similar to a polygon except that its endpoints do not meet, and it cannot be filled
- See `Rocket.java`
- See `RocketPanel.java`

```java
//*****************************************************************
//  Rocket.java       Author: Lewis/Loftus
//
//  Demonstrates the use of polygons and polylines.
//*****************************************************************

import javax.swing.JFrame;

public class Rocket
{
   //-----------------------------------------------------------------
   //  Creates the main frame of the program.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Rocket");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      RocketPanel panel = new RocketPanel();

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

```
//*******************                                    *******************
//  Rocket.java
//
//  Demonstrates the u                                    .
//*******************                                    *******************

import javax.swing.JFr

public class Rocket
{
   //-----------------                                    -------------------
   //  Creates the mai
   //-----------------                                    -------------------
   public static void
   {
      JFrame frame = new JFrame ("Rocket");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      RocketPanel panel = new RocketPanel();

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```



```java
//*****************************************************************
//  RocketPanel.java       Author: Lewis/Loftus
//
//  Demonstrates the use of polygons and polylines.
//*****************************************************************

import javax.swing.JPanel;
import java.awt.*;

public class RocketPanel extends JPanel
{
   private int[] xRocket = {100, 120, 120, 130, 130, 70, 70, 80, 80};
   private int[] yRocket = {15, 40, 115, 125, 150, 150, 125, 115, 40};

   private int[] xWindow = {95, 105, 110, 90};
   private int[] yWindow = {45, 45, 70, 70};

   private int[] xFlame = {70, 70, 75, 80, 90, 100, 110, 115, 120,
                           130, 130};
   private int[] yFlame = {155, 170, 165, 190, 170, 175, 160, 185,
                           160, 175, 155};
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Constructor: Sets up the basic characteristics of this panel.
   //-----------------------------------------------------------------
   public RocketPanel()
   {
      setBackground (Color.black);
      setPreferredSize (new Dimension(200, 200));
   }

   //-----------------------------------------------------------------
   //  Draws a rocket using polygons and polylines.
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent (page);

      page.setColor (Color.cyan);
      page.fillPolygon (xRocket, yRocket, xRocket.length);

      page.setColor (Color.gray);
      page.fillPolygon (xWindow, yWindow, xWindow.length);

      page.setColor (Color.red);
      page.drawPolyline (xFlame, yFlame, xFlame.length);
   }
}
```

The Polygon Class

- The `Polygon` class can also be used to define and draw a polygon
- It is part of the `java.awt` package
- Versions of the overloaded `drawPolygon` and `fillPolygon` methods take a single `Polygon` object as a parameter instead of arrays of coordinates

Outline

Declaring and Using Arrays

Arrays of Objects

Variable Length Parameter Lists

Two-Dimensional Arrays

Polygons and Polylines

Mouse Events and Key Events

Mouse Events

- Events related to the mouse are separated into *mouse events* and *mouse motion events*
- Mouse Events:

mouse pressed	the mouse button is pressed down
mouse released	the mouse button is released
mouse clicked	the mouse button is pressed down and released without moving the mouse in between
mouse entered	the mouse pointer is moved onto (over) a component
mouse exited	the mouse pointer is moved off of a component

Mouse Events

- Mouse motion events:

mouse moved	the mouse is moved
mouse dragged	the mouse is moved while the mouse button is pressed down

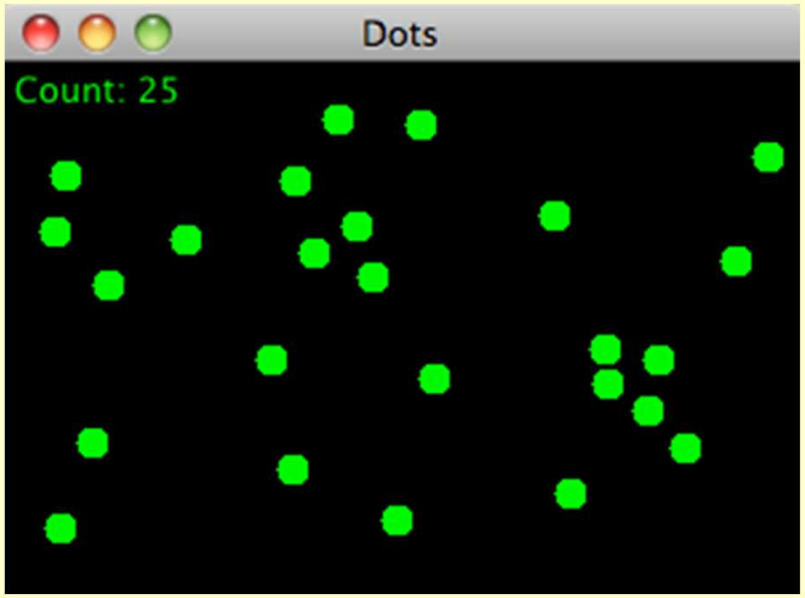
- Listeners for mouse events are created using the `MouseListener` and `MouseMotionListener` interfaces
- A `MouseEvent` object is passed to the appropriate method when a mouse event occurs

Mouse Events

- For a given program, we may only care about one or two mouse events
- To satisfy the implementation of a listener interface, empty methods must be provided for unused events
- See `Dots.java`
- See `DotsPanel.java`

```java
//*****************************************************************
//  Dots.java       Author: Lewis/Loftus
//
//  Demonstrates mouse events.
//*****************************************************************

import javax.swing.JFrame;

public class Dots
{
   //-----------------------------------------------------------------
   //  Creates and displays the application frame.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Dots");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add (new DotsPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

```
//**************************************************************
//  Dots.java
//
//  Demonstrates
//**************************************************************

import javax.swi

public class Dot
{
   //-----------------------------------------------------------
   //  Creates a
   //-----------------------------------------------------------
   public static
   {
      JFrame frame = new JFrame ("Dots");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add (new DotsPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```



```
//*******************************************************************
//  DotsPanel.java       Author: Lewis/Loftus
//
//  Represents the primary panel for the Dots program.
//*******************************************************************

import java.util.ArrayList;
import javax.swing.JPanel;
import java.awt.*;
import java.awt.event.*;

public class DotsPanel extends JPanel
{
   private final int SIZE = 6;  // radius of each dot

   private ArrayList<Point> pointList;
```

continue

continue

```
//-----------------------------------------------------------------
//  Constructor: Sets up this panel to listen for mouse events.
//-----------------------------------------------------------------
public DotsPanel()
{
   pointList = new ArrayList<Point>();

   addMouseListener (new DotsListener());
   setBackground (Color.black);
   setPreferredSize (new Dimension(300, 200));
}

//-----------------------------------------------------------------
//  Draws all of the dots stored in the list.
//-----------------------------------------------------------------
public void paintComponent (Graphics page)
{
   super.paintComponent(page);
   page.setColor (Color.green);

   for (Point spot : pointList)
      page.fillOval (spot.x-SIZE, spot.y-SIZE, SIZE*2, SIZE*2);

   page.drawString ("Count: " + pointList.size(), 5, 15);
}
```

continue

continue

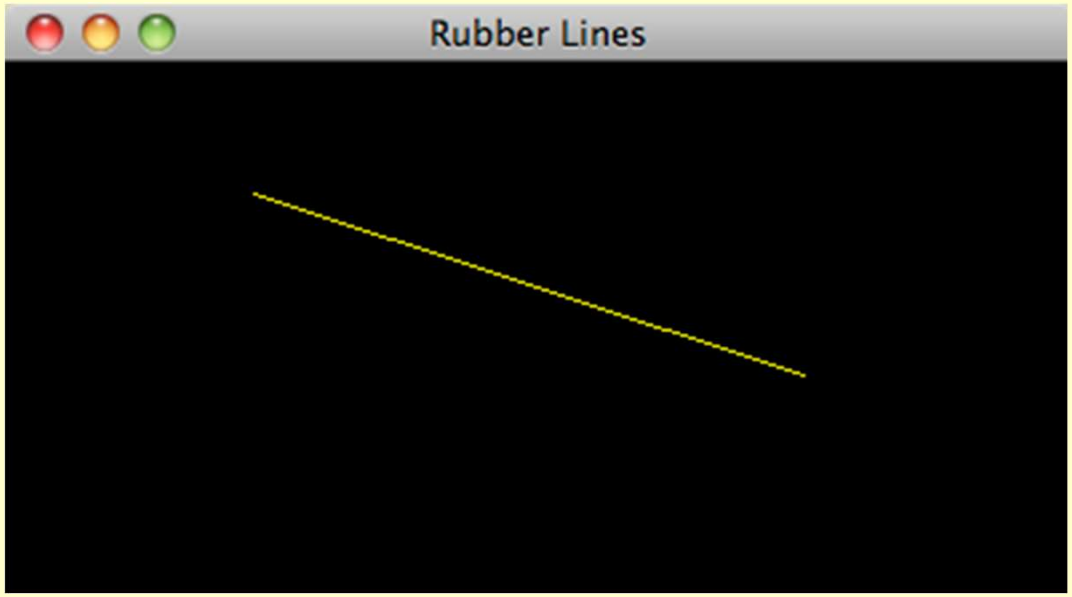
```
   //*****************************************************************
   //  Represents the listener for mouse events.
   //*****************************************************************
   private class DotsListener implements MouseListener
   {
      //-----------------------------------------------------------------
      //  Adds the current point to the list of points and redraws
      //  the panel whenever the mouse button is pressed.
      //-----------------------------------------------------------------
      public void mousePressed (MouseEvent event)
      {
         pointList.add(event.getPoint());
         repaint();
      }

      //-----------------------------------------------------------------
      //  Provide empty definitions for unused event methods.
      //-----------------------------------------------------------------
      public void mouseClicked (MouseEvent event) {}
      public void mouseReleased (MouseEvent event) {}
      public void mouseEntered (MouseEvent event) {}
      public void mouseExited (MouseEvent event) {}
   }
}
```

Mouse Events

- *Rubberbanding* is the visual effect in which a shape is "stretched" as it is drawn using the mouse
- The following example continually redraws a line as the mouse is dragged
- See `RubberLines.java`
- See `RubberLinesPanel.java`

```java
//*****************************************************************
//  RubberLines.java       Author: Lewis/Loftus
//
//  Demonstrates mouse events and rubberbanding.
//*****************************************************************

import javax.swing.JFrame;

public class RubberLines
{
   //-----------------------------------------------------------------
   //  Creates and displays the application frame.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Rubber Lines");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add (new RubberLinesPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

```
//********                                                    ********
//   Rubber
//
//   Demons
//********                                                    ********

import jav

public cla
{
   //-----                                                    --------
   //   Cre
   //-----                                                    --------
   public
   {
      JFrame frame = new JFrame ("Rubber Lines");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add (new RubberLinesPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```



```java
//*****************************************************************
//  RubberLinesPanel.java       Author: Lewis/Loftus
//
//  Represents the primary drawing panel for the RubberLines program.
//*****************************************************************

import javax.swing.JPanel;
import java.awt.*;
import java.awt.event.*;

public class RubberLinesPanel extends JPanel
{
   private Point point1 = null, point2 = null;

   //-----------------------------------------------------------------
   //  Constructor: Sets up this panel to listen for mouse events.
   //-----------------------------------------------------------------
   public RubberLinesPanel()
   {
      LineListener listener = new LineListener();
      addMouseListener (listener);
      addMouseMotionListener (listener);

      setBackground (Color.black);
      setPreferredSize (new Dimension(400, 200));
   }
```

continue

continue

```
//-----------------------------------------------------------------
//  Draws the current line from the initial mouse-pressed point to
//  the current position of the mouse.
//-----------------------------------------------------------------
public void paintComponent (Graphics page)
{
   super.paintComponent (page);
   page.setColor (Color.yellow);
   if (point1 != null && point2 != null)
      page.drawLine (point1.x, point1.y, point2.x, point2.y);
}

//*****************************************************************
//  Represents the listener for all mouse events.
//*****************************************************************
private class LineListener implements MouseListener, MouseMotionListener
{
   //--------------------------------------------------------------
   //  Captures the initial position at which the mouse button is
   //  pressed.
   //--------------------------------------------------------------
   public void mousePressed (MouseEvent event)
   {
      point1 = event.getPoint();
   }
```

continue

continue

```
      //-----------------------------------------------------------
      //  Gets the current position of the mouse as it is dragged and
      //  redraws the line to create the rubberband effect.
      //-----------------------------------------------------------
      public void mouseDragged (MouseEvent event)
      {
         point2 = event.getPoint();
         repaint();
      }

      //-----------------------------------------------------------
      //  Provide empty definitions for unused event methods.
      //-----------------------------------------------------------
      public void mouseClicked (MouseEvent event) {}
      public void mouseReleased (MouseEvent event) {}
      public void mouseEntered (MouseEvent event) {}
      public void mouseExited (MouseEvent event) {}
      public void mouseMoved (MouseEvent event) {}
   }
}
```

Key Events

- A *key event* is generated when the user types on the keyboard

key pressed	a key on the keyboard is pressed down
key released	a key on the keyboard is released
key typed	a key on the keyboard is pressed down and released

- Listeners for key events are created by implementing the `KeyListener` interface

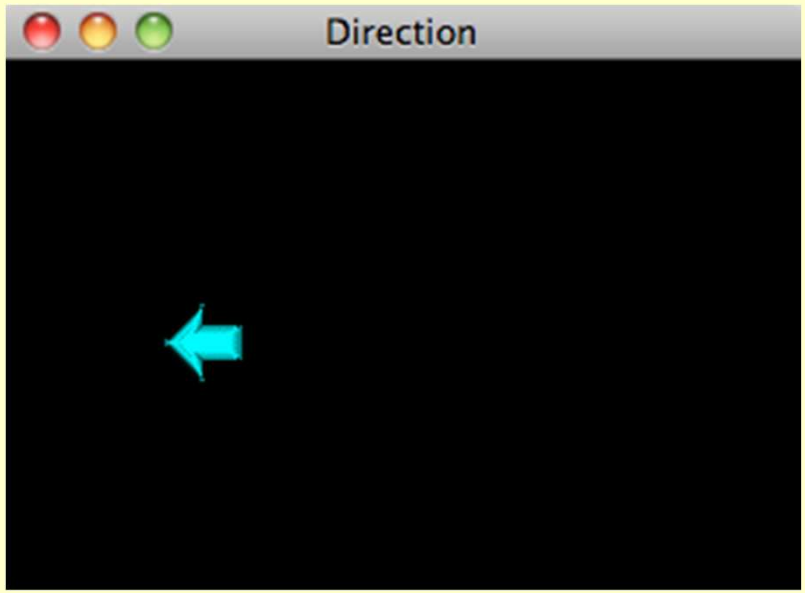
- A `KeyEvent` object is passed to the appropriate method when a key event occurs

Key Events

- The component that generates a key event is the one that has the current *keyboard focus*
- Constants in the `KeyEvent` class can be used to determine which key was pressed
- The following example "moves" an image of an arrow as the user types the keyboard arrow keys
- See `Direction.java`
- See `DirectionPanel.java`

```java
//*****************************************************************
//  Direction.java       Author: Lewis/Loftus
//
//  Demonstrates key events.
//*****************************************************************

import javax.swing.JFrame;

public class Direction
{
   //-----------------------------------------------------------------
   //  Creates and displays the application frame.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame frame = new JFrame ("Direction");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add (new DirectionPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

```
//*************                                                    ***************
//  Direction.j
//
//  Demonstrate
//*************                                                    ***************

import javax.sw

public class Di
{
   //----------                                                    ---------------
   //  Creates
   //----------                                                    ---------------
   public stati
   {
      JFrame frame = new JFrame ("Direction");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add (new DirectionPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```



```java
//*****************************************************************
//  DirectionPanel.java       Author: Lewis/Loftus
//
//  Represents the primary display panel for the Direction program.
//*****************************************************************

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DirectionPanel extends JPanel
{
   private final int WIDTH = 300, HEIGHT = 200;
   private final int JUMP = 10;  // increment for image movement

   private final int IMAGE_SIZE = 31;

   private ImageIcon up, down, right, left, currentImage;
   private int x, y;
```

continue

continue

```
//-----------------------------------------------------------------
//  Constructor: Sets up this panel and loads the images.
//-----------------------------------------------------------------
public DirectionPanel()
{
   addKeyListener (new DirectionListener());

   x = WIDTH / 2;
   y = HEIGHT / 2;

   up = new ImageIcon ("arrowUp.gif");
   down = new ImageIcon ("arrowDown.gif");
   left = new ImageIcon ("arrowLeft.gif");
   right = new ImageIcon ("arrowRight.gif");

   currentImage = right;

   setBackground (Color.black);
   setPreferredSize (new Dimension(WIDTH, HEIGHT));
   setFocusable(true);
}
```

continue

continue

```
//-----------------------------------------------------------------
//  Draws the image in the current location.
//-----------------------------------------------------------------
public void paintComponent (Graphics page)
{
   super.paintComponent (page);
   currentImage.paintIcon (this, page, x, y);
}

//*****************************************************************
//  Represents the listener for keyboard activity.
//*****************************************************************
private class DirectionListener implements KeyListener
{
   //--------------------------------------------------------------
   //  Responds to the user pressing arrow keys by adjusting the
   //  image and image location accordingly.
   //--------------------------------------------------------------
   public void keyPressed (KeyEvent event)
   {
      switch (event.getKeyCode())
      {
         case KeyEvent.VK_UP:
            currentImage = up;
            y -= JUMP;
            break;
```

continue

continue

```
            case KeyEvent.VK_DOWN:
               currentImage = down;
               y += JUMP;
               break;
            case KeyEvent.VK_LEFT:
               currentImage = left;
               x -= JUMP;
               break;
            case KeyEvent.VK_RIGHT:
               currentImage = right;
               x += JUMP;
               break;
         }

         repaint();
      }

      //-----------------------------------------------------------
      //  Provide empty definitions for unused event methods.
      //-----------------------------------------------------------
      public void keyTyped (KeyEvent event) {}
      public void keyReleased (KeyEvent event) {}
   }
}
```

Summary

- Chapter 8 has focused on:
  - array declaration and use
  - bounds checking and capacity
  - arrays that store object references
  - variable length parameter lists
  - multidimensional arrays
  - polygons and polylines
  - mouse events and keyboard events