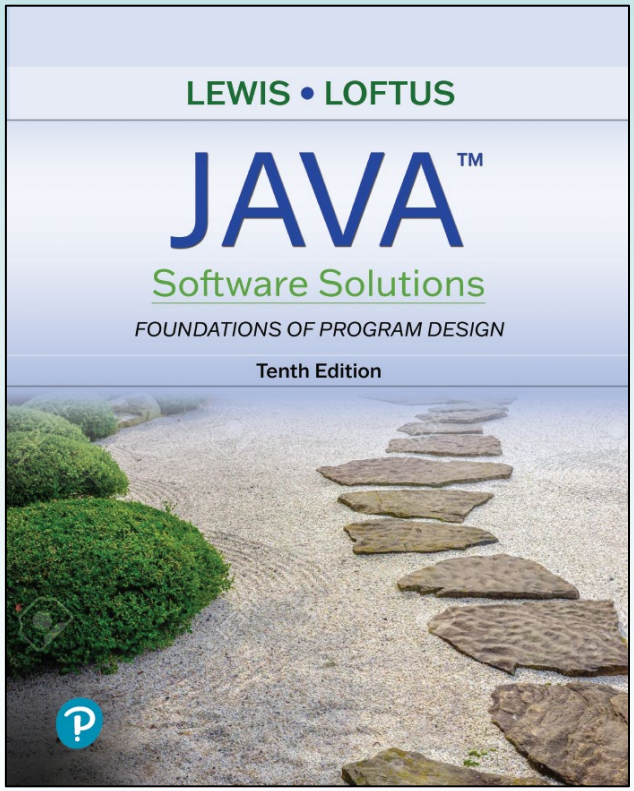
Chapter 8
Arrays



Java Software Solutions

Foundations of Program Design

(rev.05/01/2024
08/12/2026)

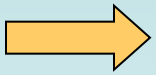
John Lewis
William Loftus

Addison-Wesley
is an imprint of
PEARSON

Arrays

- Arrays are objects that 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

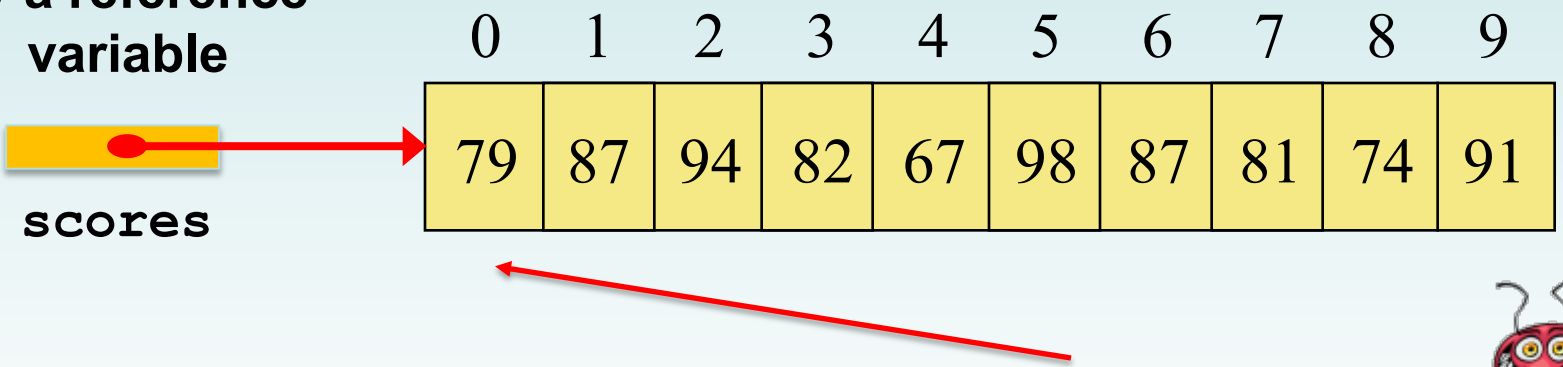
- ☺ **Polygons and Polylines**
- ☺ **Mouse Events and Key Events**

Arrays

An array is an ordered list of values of the same type

An array object is pointed to by a reference variable

Each value held in the array has a numeric *index*



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

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

The reference variable that points to the array is sometimes called the "array name"

Declaring Arrays

- The `scores` array is declared:

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

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

```
int[] scores = new int[10];
scores[0] = 79;
scores[1] = 87;
scores[2] = 94;
scores[3] = 82;
scores[4] = 67;
scores[5] = 98;
scores[6] = 87;
scores[7] = 81;
scores[8] = 74;
scores[9] = 91;

System.out.println("sum = "+(scores[4]+scores[2]));
System.out.println( scores[4]+ " " + scores[2] );

scores[5] = 9;
scores[7] = -4;

System.out.println( scores[5]+scores[7] );
```

```
sum = 161
67 94
5
```

Declaring an Array Reference Variable

- The variable `scores` could be declared:

```
int[] scores;
```

- The type of the variable `scores` is `int[]` (pointer to an array of integers)
  - But, in the above, only the reference variable exists, no array
- Later on in the code, you could do this:

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

- The variable and the object are separate things

Array Expressions

- Get a particular value by using the variable followed by the index in brackets, a *subscripted variable*.
- The expression

```
scores[2]    // "scores sub 2" is like scores2
```

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

- This is a single integer
- It can be used wherever an integer variable can be used

Subscripted Variables

It is as if the declaration

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

gives you 10 **`int`** variables with names

```
scores0, scores1, … , scores9
```

but you must use special names

```
scores[0], scores[1], … , scores[9]
```

Using Arrays

An array element can be used like any variable: assigned a value, printed, or used in a calculation:

```
scores[2] = 89;

int first = 1;

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

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

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

pick = scores[ rand.nextInt(10) ];
```

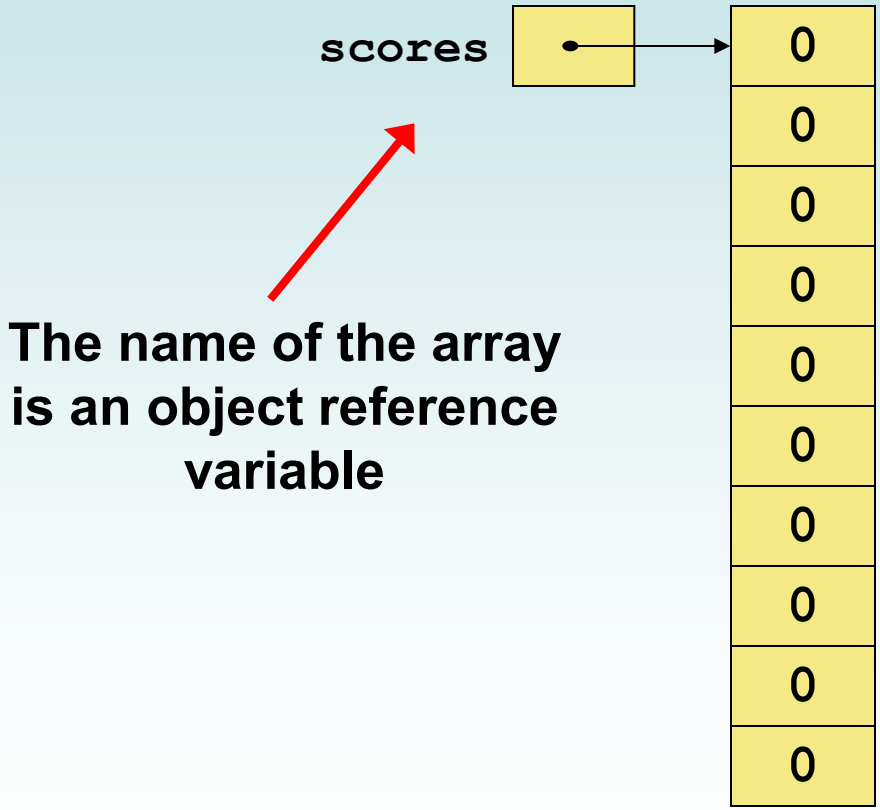
Arrays

- The values held in an array are called *array elements*
- One location in the array is called a *cell*
- An array stores multiple values of the same type
  - the *element type*
- The element type can be a primitive type or an object reference
- You can create an array of integers, an array of characters, an array of (references to) `String` objects, an array of (references to) `Coin` objects, etc.

Arrays

- The array itself is an object that must be instantiated

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



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;

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);
```

- Use this to look at all array elements starting at index 0
- It can't be used to change array values
- See `BasicArray.java`

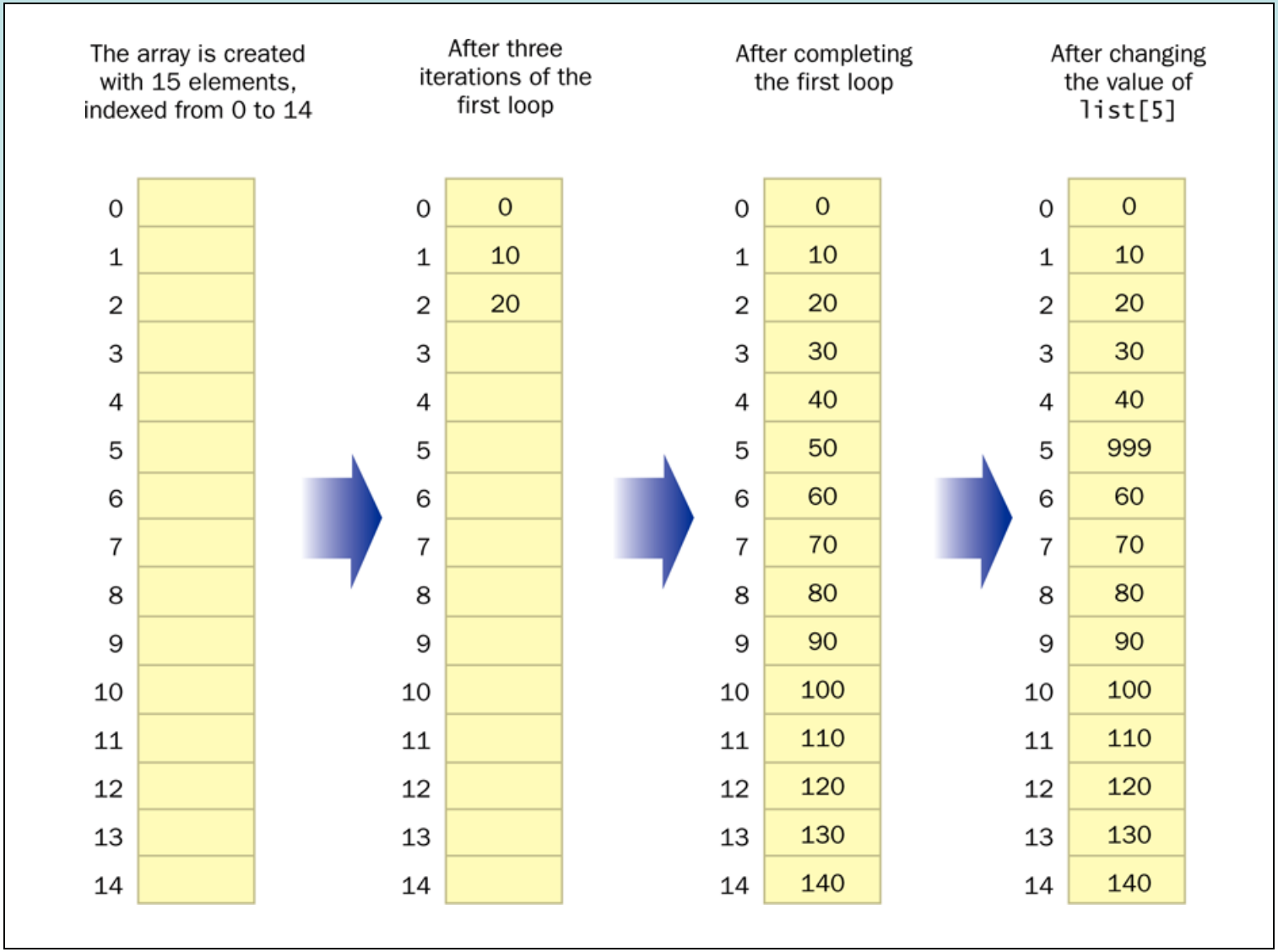
```java
//********************************************************************
//  BasicArray.java       Author: Lewis/Loftus
//
//  Demonstrates basic array declaration and use.
//********************************************************************

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
```

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

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 must specify a valid element
  - 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

- If the array `codes` can hold 100 values, it is indexed from 0 to 99
- If the value of `count` is 100, then the following an throws an exception:

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

- It's common to make *off-by-one errors* with 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 used by:

```
scores.length
```

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

```
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);

      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

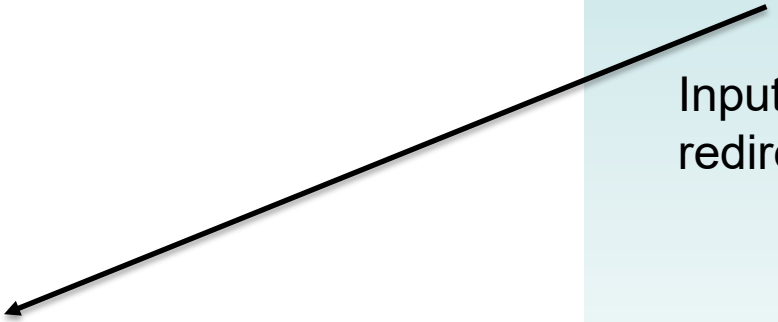
```
The size of the array: 10
Enter number 1: 18.36
Enter number 2: 48.9
Enter number 3: 53.5
Enter number 4: 29.06
Enter number 5: 72.404
Enter number 6: 34.8
Enter number 7: 63.41
Enter number 8: 45.55
Enter number 9: 69.0
Enter number 10: 99.18
The numbers in reverse order:
99.18  69.0  45.55  63.41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

```
import java.util.Scanner;

public class Average
{
   public static void main(String[] args)
   {
      Scanner scan = new Scanner(System.in);
      double[] numbers = new double[200];
      int N = 0; // number of data points
      double sum = 0.0;

      System.out.println("Enter number of values: ");
      N = scan.nextInt();
      for (int index = 0; index < N; index++)
      {
         System.out.print("Enter number " + (index+1) + ": ");
         numbers[index] = scan.nextDouble();
      }

      for (int index = 0; index < N; index++)
          sum += numbers[index];

      System.out.println("The average:" + sum/N );
   }
}
```

Copy and paste this data into Notepad++ then save to a disk file myData.txt

```
100
22 59 57 2 11 80 9 92 57 10 18 96 24 99 19 51 5 60 50 81 61 20 20 34 14
52 30 99 9 30 10 26 17 18 41 53 6 17 55 83 45 16 30 36 75 90 68 93 40 47
62 7 17 100 35 19 38 31 17 4 33 72 27 88 86 5 69 8 6 66 36 80 43 39 72 2
89 53 68 52 27 48 93 7 56 35 43 39 72 90 36 78 100 1 74 47 8 77 59 42
```

```
C:\Users\kjell\OneDrive - CCSU\AA151>javac Average.java

C:\Users\kjell\OneDrive - CCSU\AA151>java Average
Enter number of values:
5
Enter number 1: 1
Enter number 2: 2
Enter number 3: 3
Enter number 4: 4
Enter number 5: 5
The average:3.0

C:\Users\kjell\OneDrive - CCSU\AA151>java Average < myData.txt
Enter number of values:
Enter number 1: Enter number 2: Enter number 3: Enter number 4:
Enter number 84: Enter number 85: Enter number 86: Enter number 87:
. . . .
Enter number 96: Enter number 97: Enter number 98: Enter number 99: Enter number 100:
The average:44.93
```



Input redirection

```
//********************************************************************
//  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);
                                             // upper[0] counts 'A'
      int[] upper = new int[NUMCHARS]; // trick 0: cells init to 0
      int[] lower = new int[NUMCHARS];

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

```
System.out.println("Enter a sentence:");
String line = scan.nextLine();  // trick 1: line is a sequence of ch

//  Count the number of each letter occurrence
for (int ch = 0; ch < line.length(); ch++)  // trick 2: length
{
   current = line.charAt(ch); // trick 3: get char at index ch of line
   if (current >= 'A' && current <= 'Z')  // trick 4:  upper case?
      upper[current-'A']++;  // trick 5: calculate index for upper
   else                      // 'A' - 'A' == 0
      if (current >= 'a' && current <= 'z') // trick 4:  lower case?

         lower[current-'a']++; // trick 5: again
      else
         other++;
}
```

```
        //  Print the results
        System.out.println();
        for (int letter=0; letter < upper.length; letter++)
        {
           System.out.print( (char) (letter + 'A') ); // trick 6:  'A'+1 = 'B'
           System.out.print(": " + upper[letter]);
           System.out.print("\t\t" + (char) (letter + 'a') ); // trick 6:
           System.out.println(": " + lower[letter]);           // 'a'+1 = 'b'

        }

        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
```

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
```

```
import java.util.Scanner;

public class LetterCount
{
   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
      System.out.println("Enter a sentence:");
      String line = scan.nextLine();

      //  Count the number of each letter occurrence
      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++;
      }
      //  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);
   }
}
```

Alternate Array Syntax

- The brackets `[]` can be put after the element type or after the array name
- The following two declarations are equivalent:

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

- The first format is more readable. Use it.

Initializer Lists

- An *initializer list* constructs and fills 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

- 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`

```
//********************************************************************
//  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       Author: Lewis/
//
//  Demonstrates the use of an initi
//********************************

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

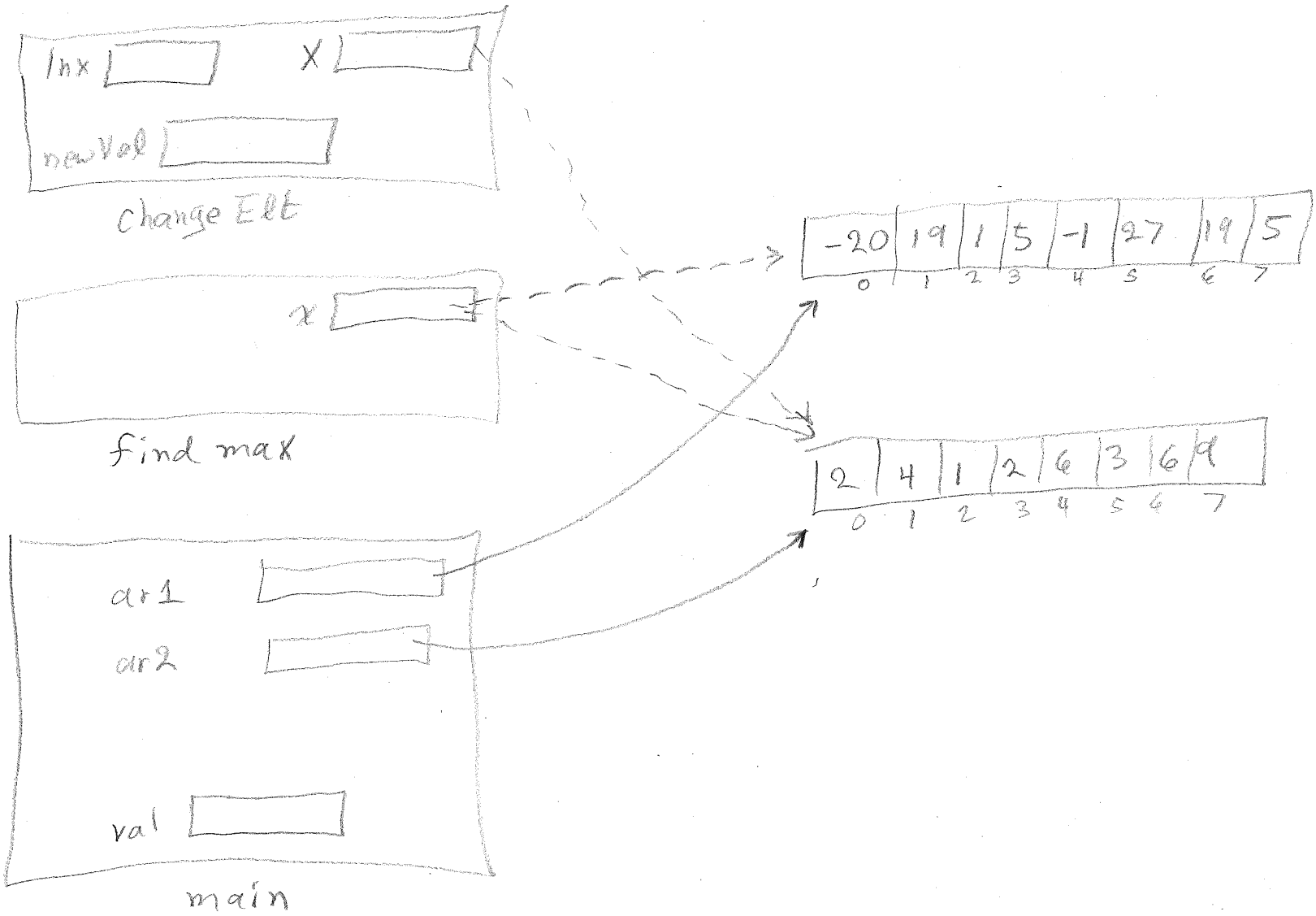
```
Array length: 8
The first few prime numbers are:
2  3  5  7  11  13  17  19
```

Arrays as Parameters

- An array reference can be a parameter to a method
- This makes the formal parameter (in the method) and the actual parameter (in the caller) aliases of each other
  - Both point to the array object
- So, changing an array element within the method changes the original in the caller
- An individual array element can be passed to a method when the type of the formal parameter is the same as the type of the elemen

```
class ArrayDemo // in file ArrayDemo.java
 {
    static int findMax( int[] x )      // x points to an array object
    {
      int max = x[0];
      for ( int index=0; index < x.length; index++ )
        if ( x[index] > max ) max = x[index] ;
      return max ;
    }

    static void changeElt( int[] x, int inx, int newVal )
    {
       x[inx] = newVal;
    }

   public static void main ( String[] args )
    {
      int[] ar1 = { -20, 19, 1, 5, -1, 27, 19, 5 } ;
      int[] ar2 = { 2, 4, 1, 2, 6, 3, 6, 9 } ;
      System.out.println("The first maximum is:  " + findMax( ar1 ) );
      System.out.println("The second maximum is: " + findMax( ar2 ) );
      changeElt( ar2, 3, 777 );
      for (int val : ar2)
         System.out.print(val + "  ");
    }
}
```

```
The first maximum is: 27
The second maximum is: 9
2  4  1  777  6  3  6  9
```


Inx
X
newVal
changeElt
x
findmax
ar1
ar2
val
main
-20 19 1 5 -1 27 19 5
0 1 2 3 4 5 6 7
2 4 1 2 6 3 6 9
0 1 2 3 4 5 6 7

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 Object References

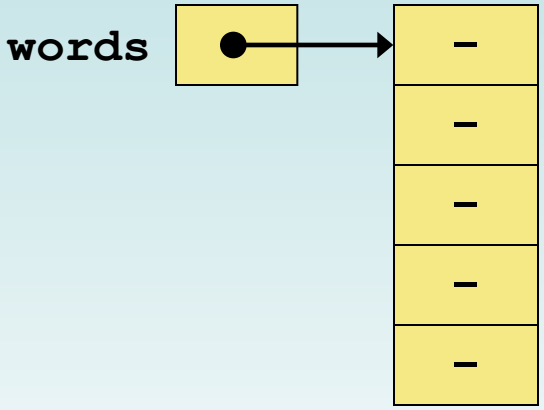
- The elements of an array can be object references
- The following declaration reserves space to store 5 references to `String` objects

```
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 Object References

- **`String[] words = new String[5];`**
- The `words` array when initially declared:



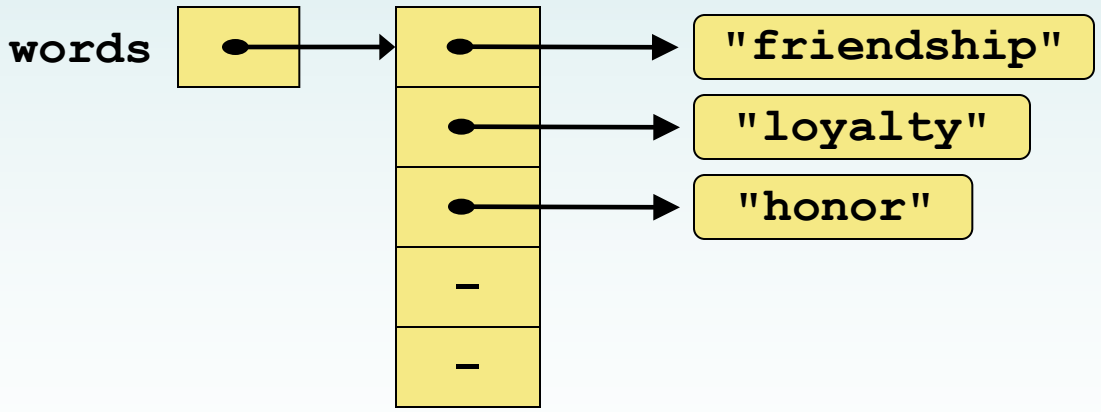
- At this point, the following line of code would throw a `NullPointerException`:

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

Arrays of Object References

Create some `String` objects and store them in the array:

```
words[0] = new String("friendship");
words[1] = new String("loyalty");
words[2] = new String("honor");
```



```
public class ArrayOfStrings
{
   public static void main( String[] args )
   {
    String[] words = new String[5];

    words[0] = new String("friendship");
    words[1] = new String("loyalty");
    words[2] = new String("honor");
    words[3] = new String("skill");
    words[4] = new String("talent");

    for ( String st : words )
      System.out.print( st + ", " );


   }
}
```

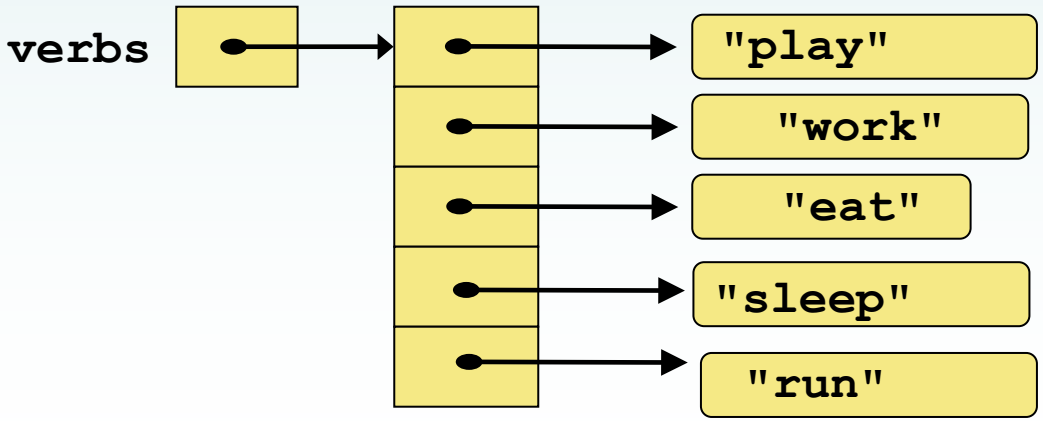
Challenges:

- add strings
- null pointers
- swap two strings

Arrays of Object References

- `String` objects can be created using literals
- The following declaration creates an array object called `verbs` and fills it with five `String` references created using string literals

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



Arrays of Object References

- The following example creates an array of `Grade` objects, each object holds a String and a numeric lower bound
- The letter grades include characters '+' and '-', so must be stored as strings instead of `char`
- See `GradeRange.java`
- See `Grade.java`

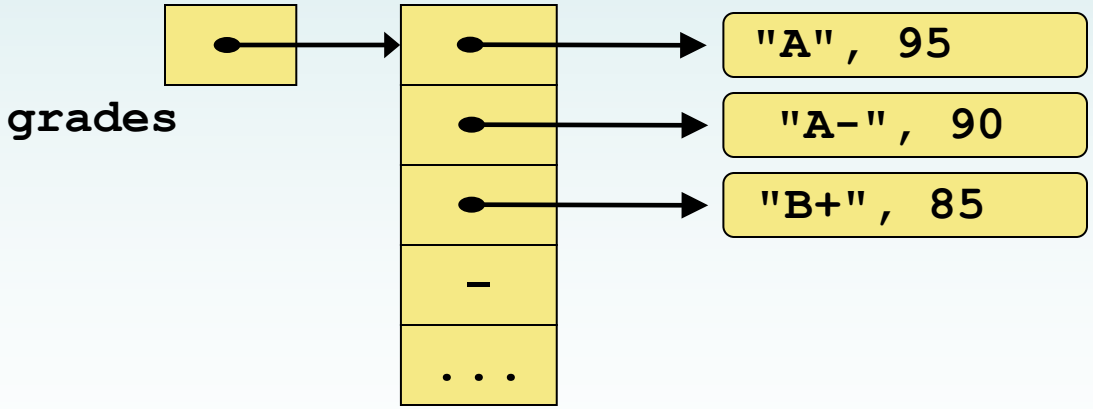
```
//********************************************************************
//  GradeRange.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an array of objects.
//********************************************************************

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);   // automatic toString()
   }
}
```

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
```

The initializer works as if the individual cells of the array were assigned object references one by one:

```
grades[0] = new Grade("A",  95);
grades[1] = new Grade("A-", 90);
grades[2] = new Grade("B+", 85);
```



```
//********************************************************************
//  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;
   }
```

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

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

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

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

```
public class Grade   // in file Grade.java
{
   private String name;
   private int lowerBound;

   public Grade(String grade, int cutoff)
   {
      name = grade;
      lowerBound = cutoff;
   }

   public String toString()
   {
      return name + "\t" + lowerBound;
   }
   public void setName(String grade)
   {
      name = grade;
   }

   public void setLowerBound(int cutoff)
   {
      lowerBound = cutoff;
   }
   public String getName()
   {
      return name;
   }

   public int getLowerBound()
   {
      return lowerBound;
   }
}
```

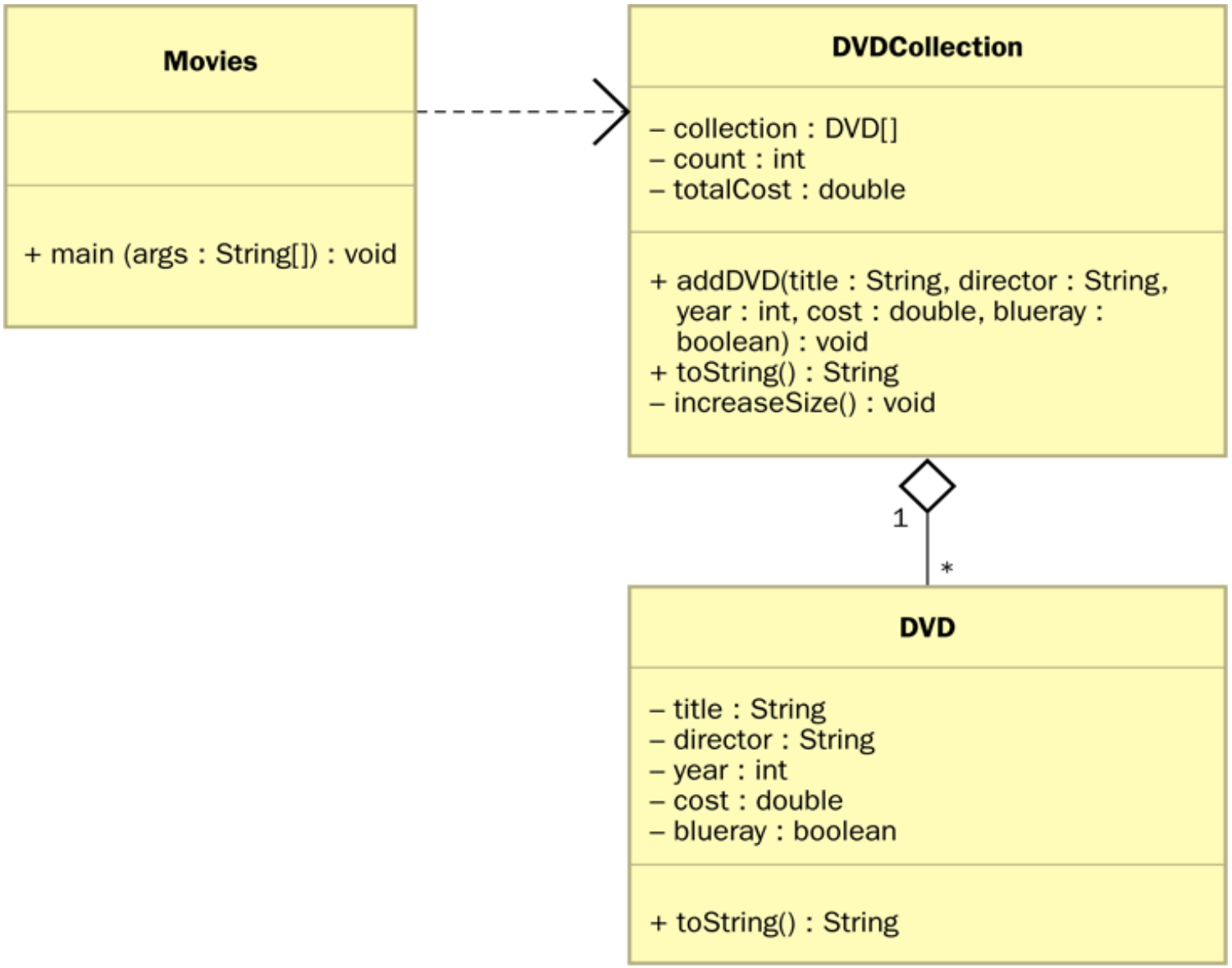
```
//********************************************************************
//  GradeRange.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an array of objects.
//********************************************************************

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);
   }
}
```

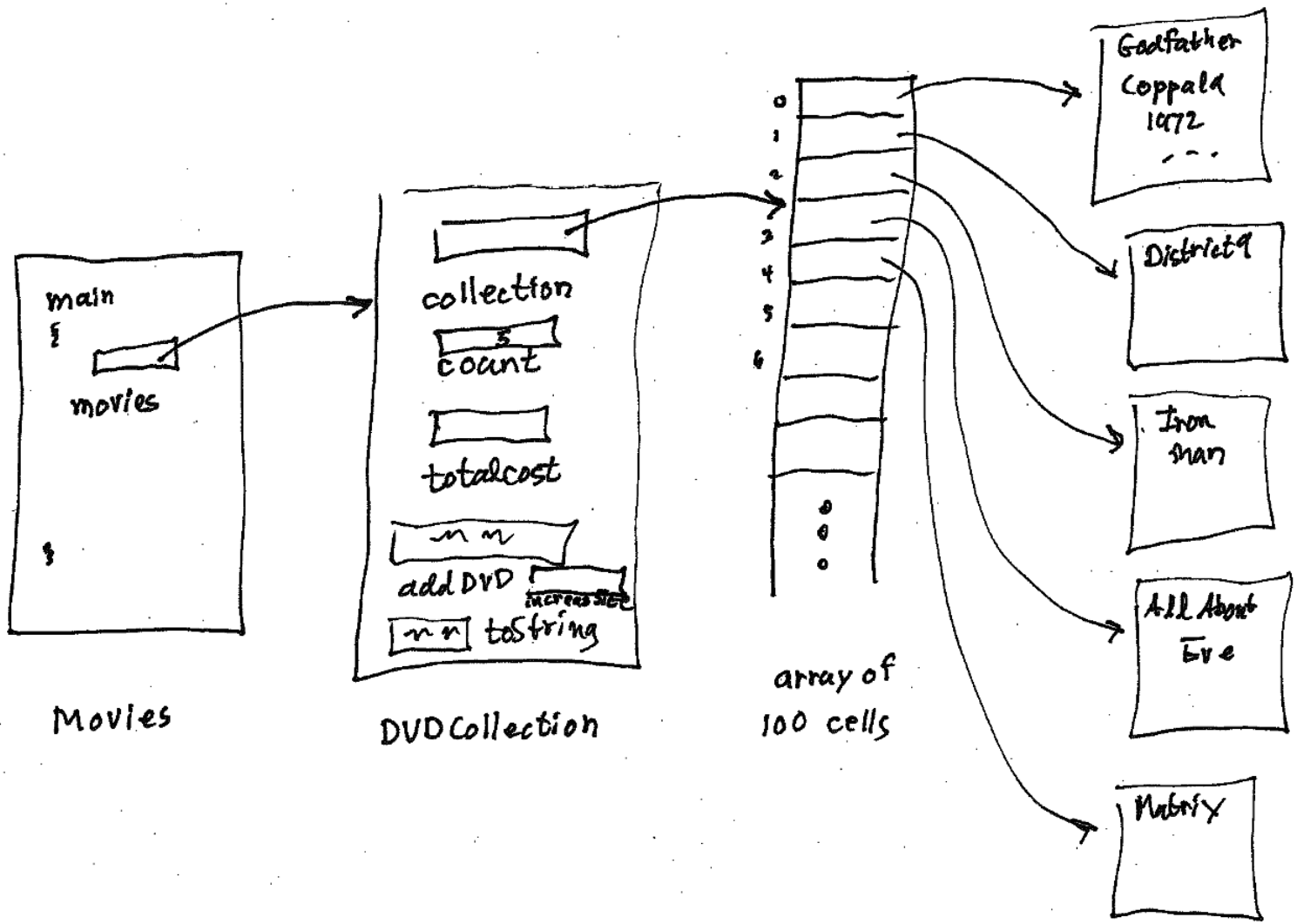
Challenges:

- change lower bound
- add a grade
- find grade for 78

Arrays of Object References

- Example: manage 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`





```
//********************************************************************
//  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;

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;
   }
```

```
   //-----------------------------------------------------------------
   //  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;
   }
}
```

```
import java.text.NumberFormat;

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

   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;
   }

   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;
   }
}
```

```
public class DVDtester
{
  public static void main( String[] args )
  {
    DVD[] dvd = new DVD[10];
    dvd[0] = new DVD( "Casablanca", "Michael Curtiz",
                      1942, 19.95, false );
    dvd[1] = new DVD( "Iron Man", "Jon Favreau", 2008, 15.95, false );
    System.out.println( dvd[0] );
    System.out.println( dvd[1] );
  }
}
```

```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);
    }
}
```

Output

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

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
```

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

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

import java.text.NumberFormat;

public class DVDCollection
{
   private DVD[] collection;
   private int count; // number of DVDs currently in collection
   private double totalCost;

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

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++;
}
```

```
//-----------------------------------------------------------------
//  Returns a very long string 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";

   // only include array cells that have a dvd
   for (int dvd = 0; dvd < count; dvd++)
      report += collection[dvd].toString() + "\n";

   return report;
}
```

```
    //-----------------------------------------------------------------
    //  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]; // a completely new array

       // copy pointers from old array into new one
       for (int dvd = 0; dvd < collection.length; dvd++)
          temp[dvd] = collection[dvd];

       // now point collection at the new array (old one becomes garbage)
       collection = temp;
    }
}
```

```
import java.text.NumberFormat;

public class DVDCollection
{
   private DVD[] collection;
   private int count; // number of DVDs currently in collection
   private double totalCost;
   public DVDCollection()
   {
      collection = new DVD[100];   count = 0;  totalCost = 0.0;
   }

   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++;
   }

   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";

      // only include array cells that have a dvd
      for (int dvd = 0; dvd < count; dvd++)
         report += collection[dvd].toString() + "\n";

      return report;
   }

   private void increaseSize()
   {
      DVD[] temp = new DVD[collection.length * 2]; // a completely new array

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

      collection = temp;
   }
}
```

☺ 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` object references to the `main` method of the `StateEval` program:

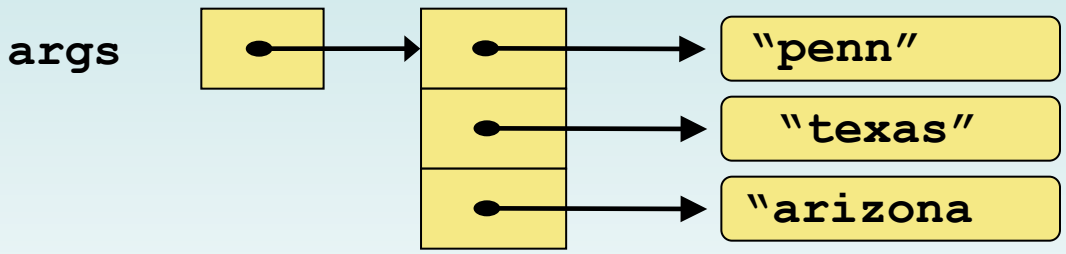
```
java StateEval pennsylvania texas arizona
```

- See `NameTag.java` (on disk)

☺ Array of Object References

```
java StateEval penn texas Arizona
```

```
public static void main(String[] args)
```



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

☺ Numerical Arguments

`main(String[] args )`

`args.length` — number of arguments

```
int choice = Integer.parseInt(args[0]);
```

converts the String pointed to by `args[0]` to an `int`

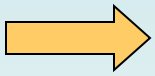
```
double val =
    Double.parseDouble(args[0]);
```

converts the String pointed to by `args[0]` to a `double`

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

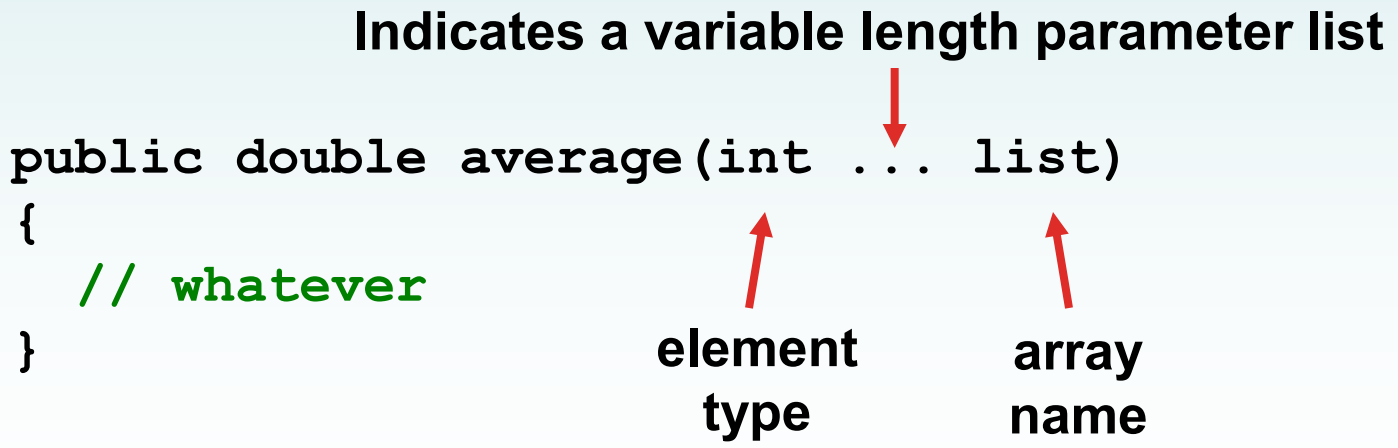
```
// one call to average three values
mean1 = average(42, 69, 37);

// 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



☺ 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`

```
//********************************************************************
//  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(                     )
   {
      Family lewis = new Fa                     "Sharon", "Justin", "Kayla",
         "Nathan", "Samanth

      Family camden = new F                     n", "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
```

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

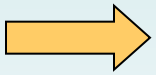
```
    //-----------------------------------------------------------------
    //  Returns a string representation of this family.
    //-----------------------------------------------------------------
    public String toString()
    {
        String result = "";

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

        return result;
    }
}
```

Outline

Declaring and Using Arrays

Arrays of Objects

- ☺ **Variable Length Parameter Lists**

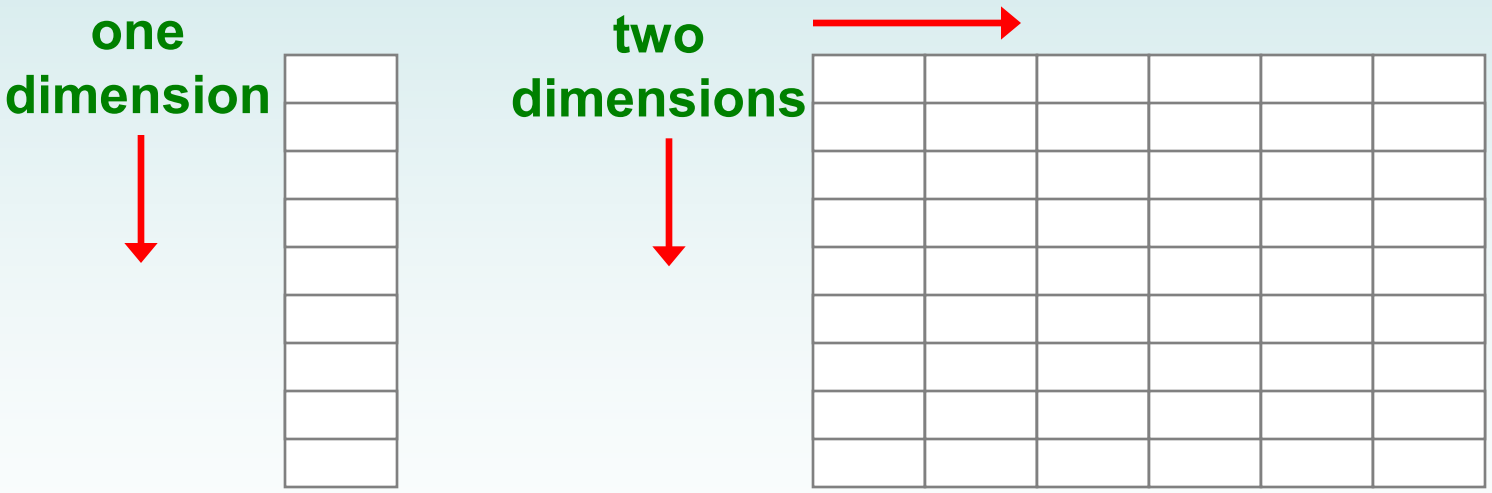


Two-Dimensional Arrays

- ☺ **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

- 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, row, and column:

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

- An 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[][]`	Reference to a 2D array of integers, or array of integer arrays
`table[5]`	`int[]`	Reference to an array of integers
`table[5][12]`	`int`	An integer

```
public class TwoDArray
{
   //-----------------------------------------------------------------
   //  Creates a 2D array of integers
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      int[][] table = new int[5][10]; // 5 rows, 10 columns

      System.out.println( "table[3][4]= " + table[3][4] );
      table[3][4] = 99;
      System.out.println( "table[3][4]= " + table[3][4] );
      int sum = table[3][4] + table[2][1];
      System.out.println( sum );
   }
}
```

```
table[3][4]= 0
table[3][4]= 99
99
```

```
//********************************************************************
//  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]; // 5 rows, 10 columns

      // 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();
      }
   }
}
```

```
public class TwoDArray2
{

   public static void main(String[] args)
   {
      int[][] table = {{2,4,5,8}, {-1,7,7,9}, {6,2,1,5}}; //3 rows, 4 cols
      int sum = 0;

      // compute the sum
      for (int row=0; row < table.length; row++)
         for (int col=0; col < table[row].length; col++)
            sum += table[row][col];

      System.out.println( "sum = " + sum );
   }
}
```

sum = 55

```
public class TwoDArray3
{

   public static void main(String[] args)
   {
      int[][] table = {{2,4,5,8}, {-1,7,7,9}, {6,2,1,5}}; //3 rows, 4 cols
      int[] sum = {0,0,0}; // {sum of row 0, sum of row 1, sum of row 2}

      // compute the sum of each row
      for (int row=0; row < table.length; row++)
         for (int col=0; col < table[row].length; col++)
            sum[row] += table[row][col];

      for (int val : sum )
         System.out.print( val + ""  );
   }
}
```

19 22 14

Soda Survey

- Suppose a soda manufacturer has 4 new flavors.
- 10 people try each new flavor and rates them on a scale of 1 to 5.
- The data is in an array.
- Each row is for one flavor. Each int in a row is the rating one taster gave.
- E.g. soda #3 was rated 4 by taster number 3

```
int[][] scores = { {3, 4, 5, 2, 1, 4, 3, 2, 4, 4}, // soda #1
                   {2, 4, 3, 4, 3, 3, 2, 1, 2, 2}, // soda #2
                   {3, 5, 4, 5, 5, 3, 2, 5, 5, 5}, // soda #3
                   {1, 1, 1, 3, 1, 2, 1, 3, 2, 4} }; // soda #4
```

```
//********************************************************************
//  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}, // soda #1
                         {2, 4, 3, 4, 3, 3, 2, 1, 2, 2}, // soda #2
                         {3, 5, 4, 5, 5, 3, 2, 5, 5, 5}, // soda #3
                         {1, 1, 1, 3, 1, 2, 1, 3, 2, 4} }; // soda #4

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

      int[] sodaSum   = new int[SODAS];  // sum of scores for each soda
      int[] personSum = new int[PEOPLE]; // sum of scores give by each
                                         // person
```

continue

continue

```
        // visit every cell of 2D array. Add its value to 2 sums.
        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");

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

        // compute and print the average rating each person gave
        System.out.println ();
        for (int person=0; person < PEOPLE; person++)
           System.out.println("Person #" + (person+1) + ": " +
                              fmt.format((float)personSum[person]/SODAS));
    }
}
```

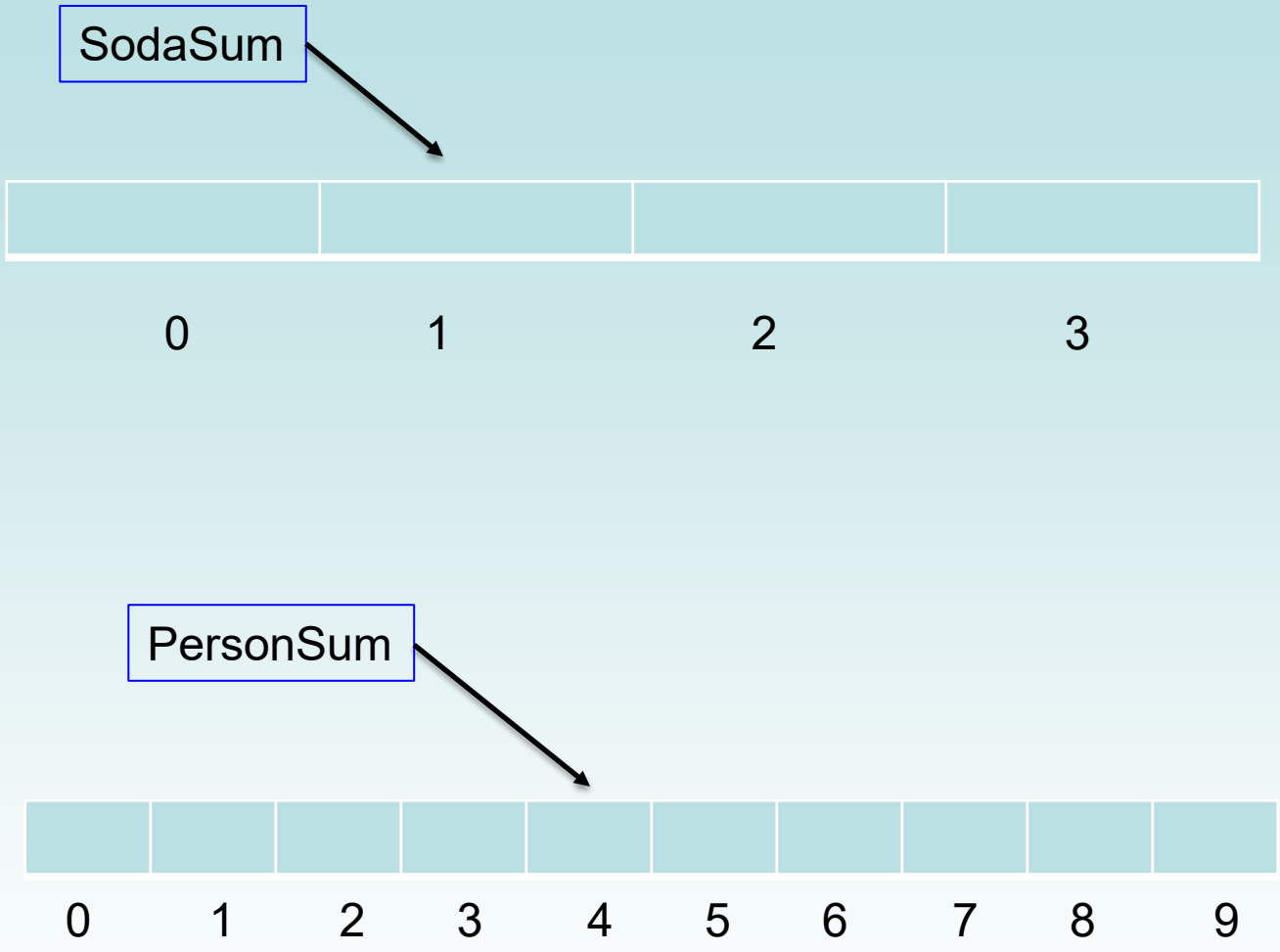
continue

```
      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
```

```
                    column
      0  1   2   3   4  5  6  7  8   9
{ {3, 4, 5, 2, 1, 4, 3, 2, 4, 4},   0    // soda #1
  {2, 4, 3, 4, 3, 3, 2, 1, 2, 2},   1    // soda #2
  {3, 5, 4, 5, 5, 3, 2, 5, 5, 5},    2   // soda #3
  {1, 1, 1, 3, 1, 2, 1, 3, 2, 4} };  3   // soda #4
                                     row
```



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*

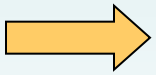
Outline

Declaring and Using Arrays

Arrays of Objects

- ☺ **Variable Length Parameter Lists**

Two-Dimensional Arrays

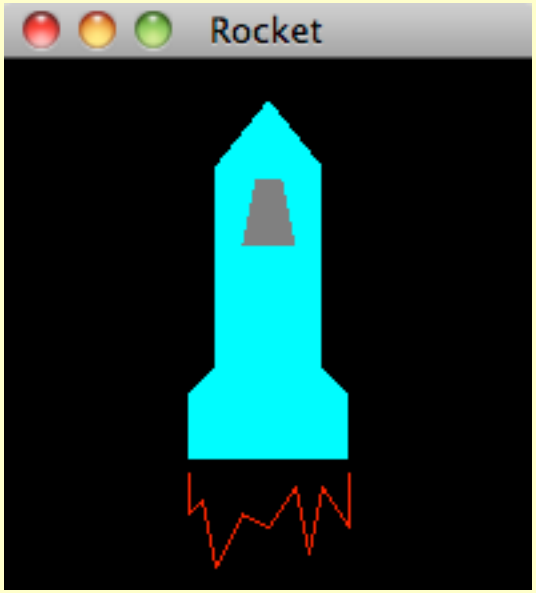


- ☺ **Polygons and Polylines**
- ☺ **Mouse Events and Key Events**

☺ 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`

```
//********************************************************************
//  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);
   }
}
```

```
//********************************************************************
//  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:

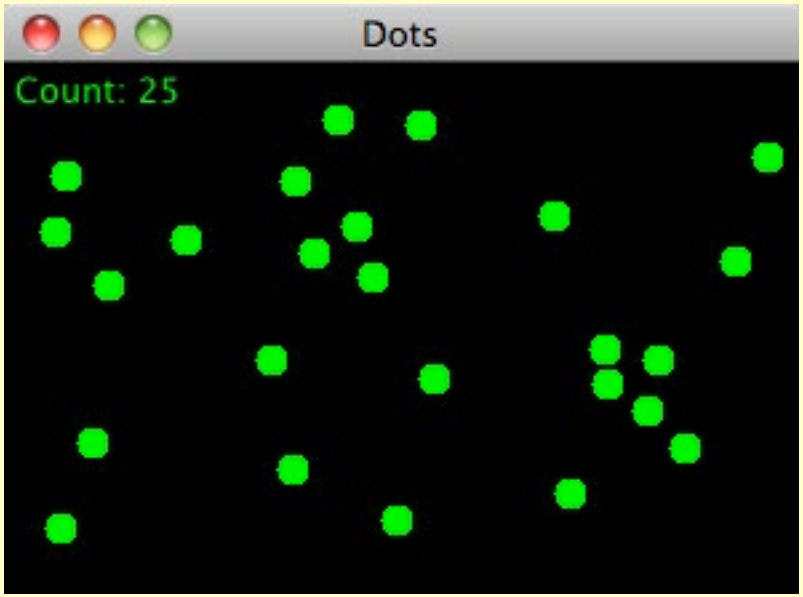
mouse moved	the mouse is moved
mouse dragged	the mouse is moved while the mouse button is pressed down

- 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

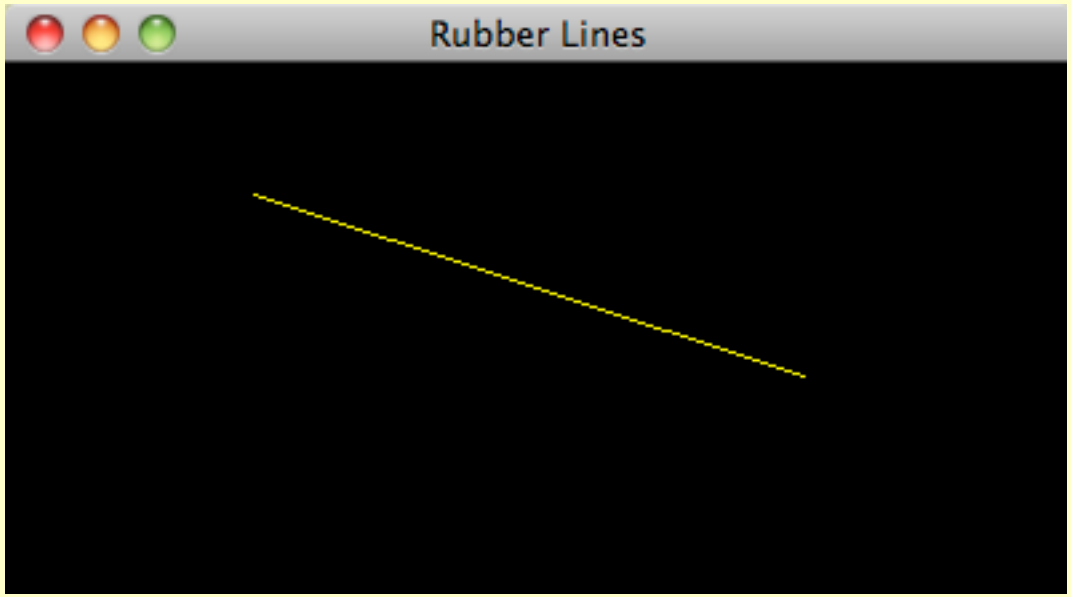
continue

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

```
//********************************************************************
//  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);
   }
}
```

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

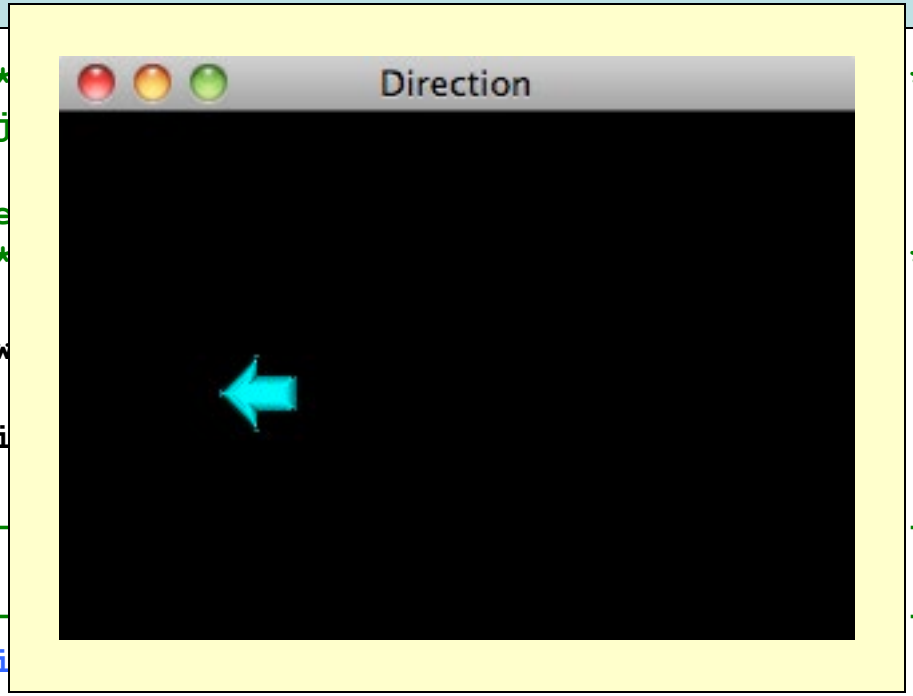
key pressed	a key on the keyboard is pressed down
key released	a key on the keyboard is released
key typed	a key on the keyboard is pressed down and released

- 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`

```
//********************************************************************
//  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);
   }
}
```

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