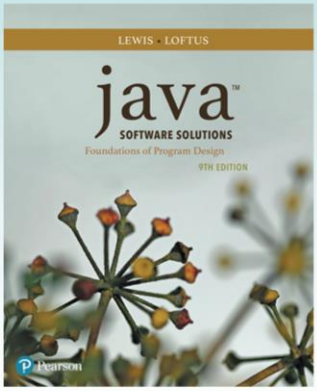
Chapter 8
Arrays



Java Software Solutions
Foundations of Program Design
9th Edition

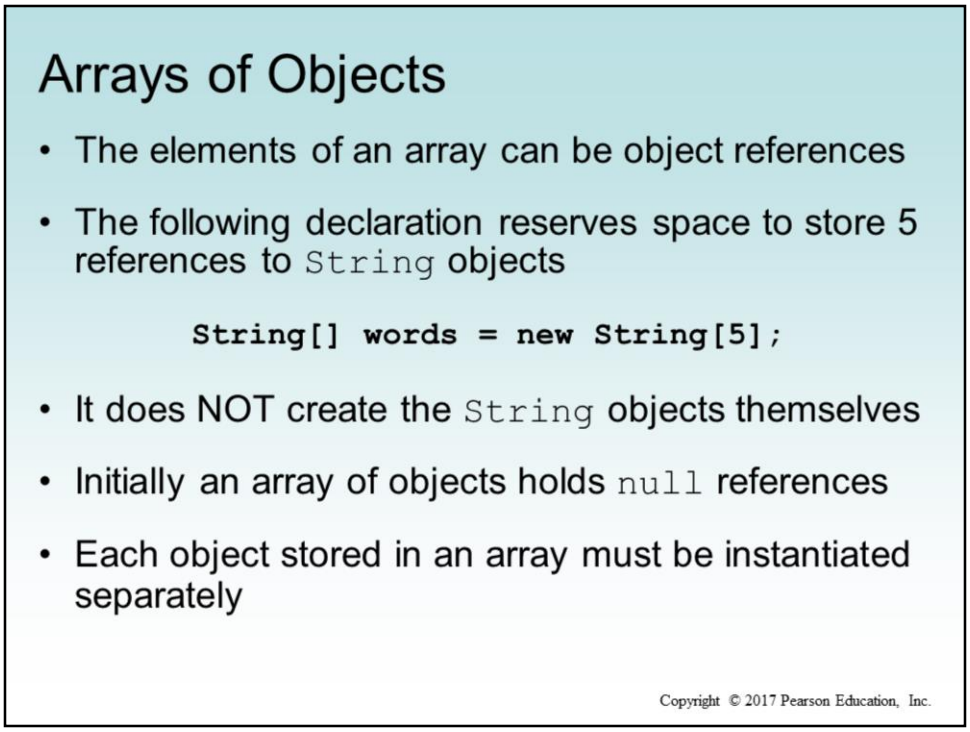
John Lewis
William Loftus

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

Outline

- Declaring and Using Arrays
- **Arrays of Objects**
- Variable Length Parameter Lists
- Two-Dimensional Arrays



-In the previous section, recall we looked at how to create arrays storing primitive variable types

-In this section, let's examine creating arrays storing object variables (or object reference variables)

-The most important thing to remember is that an object variable stores a **reference (or memory address)**

-This reference is to another area in memory that contains the **actual object that is created with the new operator**

-Think of this reference as "pointing to" another area in memory containing the object created with new

-When an array of object variables are created, the array contains a group of such references

-**HOWEVER, the references are not pointing to anything yet, because the actual objects have not been created!**

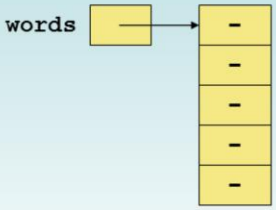
-In Java, when a reference is not pointing to anything it contains the value "null"

-Think of null as meaning "no address specified"

-As we create objects with the new operator, we can assign the address of an object to one of these references!

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]);
```


-Here we see what memory looks like after an array of String objects is created initially with:

String[] words = new String[5];

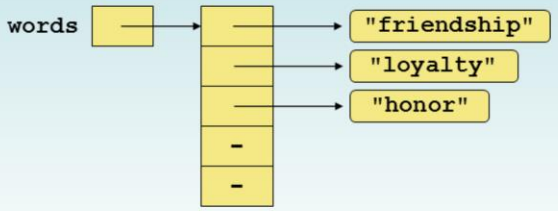
-Note that our array (which is an object itself) simply contains 5 null references

-We haven't actually created the String objects yet

-This is why a run-time error would occur if we tried the example statement above

Arrays of Objects

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



-We can create some String objects with the new operator and then assign them to the references 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"};
```


-Recall that a shortcut to creating a String object with the new operator is to simply specify the string

-In other words, the statement:

String s = new String(“Hello”);

-Is the same as:

String s = “Hello”;

-Therefore, initializing the array in this slide example **creates and assigns String objects to the array in a single step**

-Remember that this only works for the String class

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`

-Unlike the String class, we need to create each object with the new operator with other classes

-The new operator returns the memory address (reference) where the object is created

-We can assign this result to one of the elements in the array of objects

-The next example in the slide demonstrates this process

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


-Note how the array of Grade objects is created

-The array is initialized by creating each object with the new operator

-The reference returned by each new operator is assigned to an array element

-After this creation, we use a for-each loop to print each grade object created using our array!

```
//*********************                    *****************************
//  GradeRange.java       A                 oftus
//
//  Demonstrates the use of                 bjects.
//*********************                    *****************************

public class GradeRange
{
   //-----------------                      -----------------------------
   //  Creates an array of                   and prints them.
   //-----------------                      -----------------------------
   public static void main
   {
      Grade[] grades =
      {
         new Grade("A", 95)                    -", 90),
         new Grade("B+", 87),                  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
```

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

-This next example creates a class that contains (or “knows” about) an array of objects (DVDs)

-DVDCollection class contains an array of DVDs

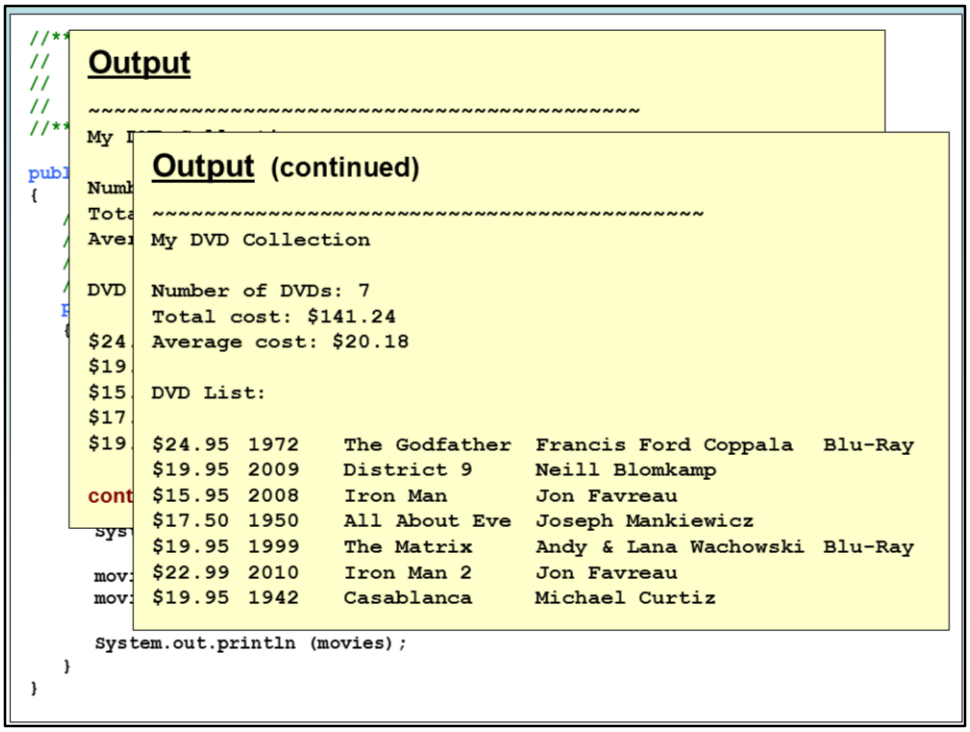
```
//********************************************************************
//  Movies.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an array of objects.
//********************************************************************

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
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;
   private double totalCost;

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

continue

-Here the array of DVD objects is declared and created in the constructor

-HOWEVER, no actual DVD objects are created yet, so each location initially contains "null"

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

-This method actually creates the DVD object and assigns it to one of the array locations

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;
   }
}
```

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

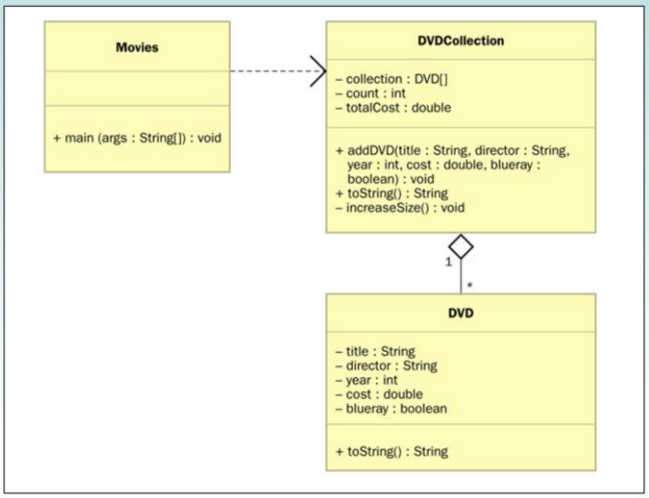
continue

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:



-Note the diamond indicates aggregation which is translated the DVDCollection **has DVDs**

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`

-"command-line arguments" refers to extra information specified **after the name of a program when executing**

-We've been using Eclipse to execute our program automatically for us

-We could also execute our program from a window in our operating system where we can enter commands

-Such windows are typically called "command prompts"

-In the Windows operating system, we can type "DOS" commands in such a window

-In Windows, you can type "cmd" in the search bar to bring up this "command-line window"

-One of the commands we could type is the name of our program followed by extra information

-If we list additional information (separated by spaces) after the name, these can be read into our program

-This additional information is known as command-line arguments and are read into the program as Strings

-This is why we see an array of Strings in the main method of our driver classes!

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


-This program demonstrates using the command-line arguments in a program

-We can specify command-line arguments in Eclipse by executing our program through Run->Run Configurations…

-Specify your “Main” class (the one containing the main method), then select the “Arguments” tab

-Type your argument(s) in the “Program arguments” area in this tab

-Multiple arguments are specified with a space between each argument

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