Chapter 6: Arrays and Vectors

Presentation slides for

Java Software Solutions

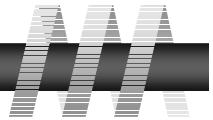
Foundations of Program Design

Second Edition

by John Lewis and William Loftus

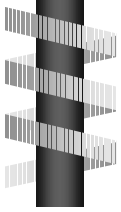
Java Software Solutions is published by Addison-Wesley

Arrays and Vectors



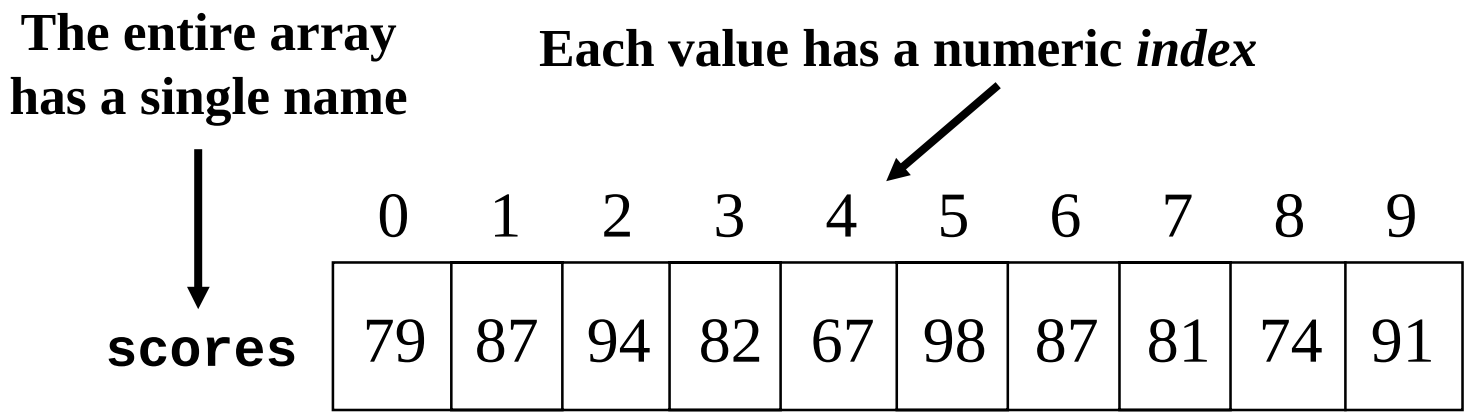
- **Arrays and vectors are objects that help us organize large amounts of information**

- **Chapter 6 focuses on:**
  - **array declaration and use**
  - **arrays of objects**
  - **sorting elements in an array**
  - **multidimensional arrays**
  - **the `Vector` class**
  - **using arrays to manage graphics**



Arrays

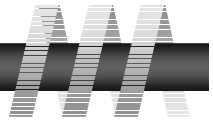
- An *array* is an ordered list of values



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

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

Arrays

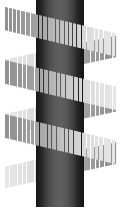


- A particular value in an array is referenced using the array name followed by the index in brackets
- For example, the expression

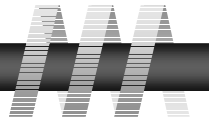
```
scores[2]
```

refers to the value 94 (which is the 3rd value in the array)

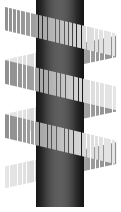
- That expression represents a place to store a single integer, and can be used wherever an integer variable can
- For example, it can be assigned a value, printed, or used in a calculation



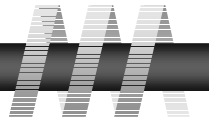
Arrays



- An array stores multiple values of the same type
- That type can be primitive types or objects
- Therefore, we can create an array of integers, or an array of characters, or an array of String objects, etc.
- In Java, the array itself is an object
- Therefore the name of the array is a object reference variable, and the array itself is instantiated separately



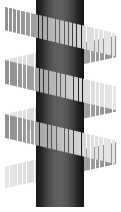
Declaring Arrays



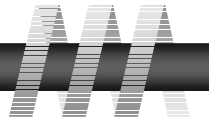
- The `scores` array could be declared as follows:

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

- Note that the type of the array does not specify its size, but each object of that type has a specific size
- The type of the variable `scores` is `int[]` (an array of integers)
- It is set to a new array object that can hold 10 integers
- See **`BasicArray.java`** (page 270)

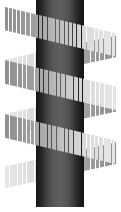


Declaring Arrays

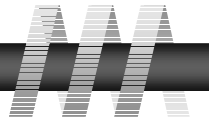


- **Some examples of array declarations:**

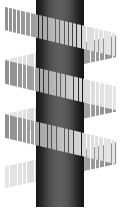
```
float[] prices = new float[500];

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

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



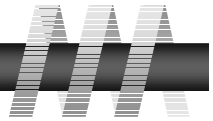
Bounds Checking



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



Bounds Checking



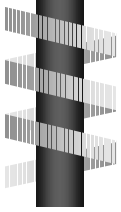
- For example, if the array `codes` can hold 100 values, it can only be indexed using the numbers 0 to 99
- If `count` has the value 100, then the following reference will cause an `ArrayOutOfBoundsException`:

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

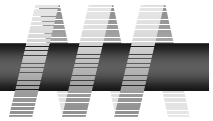
- It's common to introduce *off-by-one errors* when using arrays

problem

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



Bounds Checking



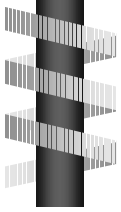
- Each array object has a public constant called `length` that stores the size of the array

- It is referenced using the array name (just like any other object):

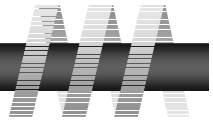
```
scores.length
```

- Note that `length` holds the number of elements, not the largest index

- See **ReverseNumbers.java** (page 272)
- See **LetterCount.java** (page 274)



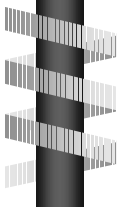
Array Declarations Revisited



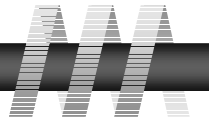
- The brackets of the array type can be associated with the element type or with the name of the array
- Therefore the following declarations are equivalent:

```
float[] prices;
float prices[];
```

- The first format is generally more readable

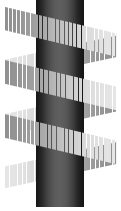


Initializer Lists

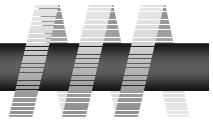


- **An *initializer list* can be used to instantiate and initialize an array in one step**
- **The values are delimited by braces and separated by commas**
- **Examples:**

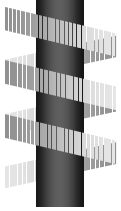
```
int[] units = {147, 323, 89, 933, 540,
               269, 97, 114, 298, 476};

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



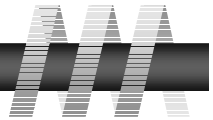
Initializer Lists



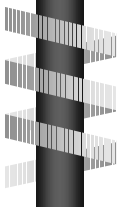
- **Note that when an initializer list is used:**
  - **the `new` operator is not used**
  - **no size value is specified**
- **The size of the array is determined by the number of items in the initializer list**
- **An initializer list can only be used in the declaration of an array**
- **See `Primes.java` (page 278)**



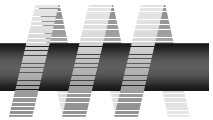
Arrays as Parameters



- **An entire array can be passed to a method as a parameter**
- **Like any other object, the reference to the array is passed, making the formal and actual parameters aliases of each other**
- **Changing an array element in the method changes the original**
- **An array element can be passed to a method as well, and will follow the parameter passing rules of that element's type**



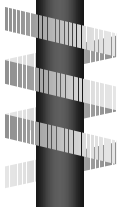
Arrays of Objects



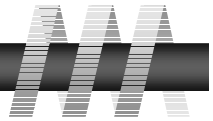
- The elements of an array can be object references
- The following declaration reserves space to store 25 references to `String` objects

```
String[] words = new String[25];
```

- It does NOT create the `String` objects themselves
- Each object stored in an array must be instantiated separately
- See `GradeRange.java` (page 280)



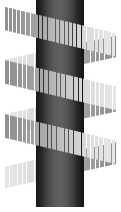
Command-Line Arguments



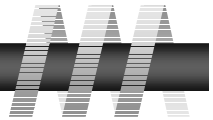
- The signature of the `main` method indicates that it takes an array of `String` objects as a parameter
- These values come from command-line arguments that are provided when the interpreter is invoked
- For example, the following invocation of the interpreter passes an array of three `String` objects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

```
> java DoIt pennsylvania texas california
```

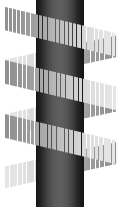
- These strings are stored at indexes 0-2 of the parameter
- See **`NameTag.java`** (page 281)



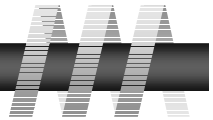
Arrays of Objects



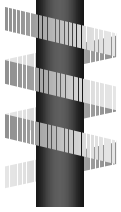
- Objects can have arrays as instance variables
- Therefore, fairly complex structures can be created simply with arrays and objects
- The software designer must carefully determine an organization of data and objects that makes sense for the situation
- See `Tunes.java` (page 282)
- See `CDCollection.java` (page 284)
- See `CD.java` (page 286)



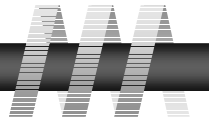
Sorting



- **Sorting is the process of arranging a list of items into a particular order**
- **There must be some value on which the order is based**
- **There are many algorithms for sorting a list of items**
- **These algorithms vary in efficiency**
- **We will examine two specific algorithms:**
  - **Selection Sort**
  - **Insertion Sort**

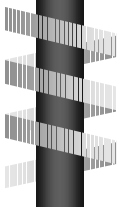


Selection Sort

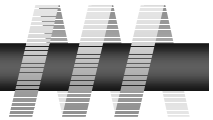


- **The approach of Selection Sort:**
  - **select one value and put it in its final place in the sort list**
  - **repeat for all other values**

- **In more detail:**
  - **find the smallest value in the list**
  - **switch it with the value in the first position**
  - **find the next smallest value in the list**
  - **switch it with the value in the second position**
  - **repeat until all values are placed**



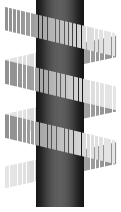
Selection Sort



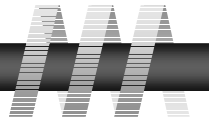
- **An example:**

```
original:          3   9   6   1   2
smallest is 1:     1   9   6   3   2
smallest is 2:     1   2   6   3   9
smallest is 3:     1   2   3   6   9
smallest is 6:     1   2   3   6   9
```

- **See `SortGrades.java` (page 289)**
- **See `Sorts.java` (page 290) -- the `selectionSort` method**

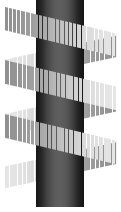


Insertion Sort

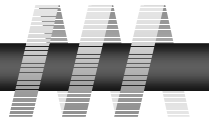


- **The approach of Insertion Sort:**
  - **Pick any item and insert it into its proper place in a sorted sublist**
  - **repeat until all items have been inserted**

- **In more detail:**
  - **consider the first item to be a sorted sublist (of one item)**
  - **insert the second item into the sorted sublist, shifting items as necessary to make room to insert the new addition**
  - **insert the third item into the sorted sublist (of two items), shifting as necessary**
  - **repeat until all values are inserted into their proper position**



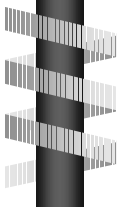
Insertion Sort



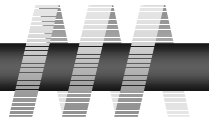
- **An example:**

```
original:     3   9   6   1   2
insert 9:     3   9   6   1   2
insert 6:     3   6   9   1   2
insert 1:     1   3   6   9   2
insert 2:     1   2   3   6   9
```

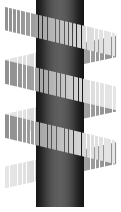
- **See `Sorts.java` (page 290) -- the `insertionSort` method**



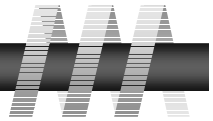
Sorting Objects



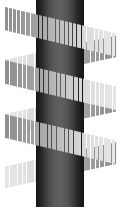
- Integers have an inherent order, but the order of a set of objects must be defined by the person defining the class
- Recall that a Java interface can be used as a type name and guarantees that a particular class has implemented particular methods
- We can use the `Comparable` interface to develop a generic sort for a set of objects
- See `SortPhoneList.java` (page 294)
- See `Contact.java` (page 295)
- See `Sorts.java` (page 290)



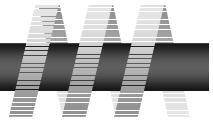
Comparing Sorts



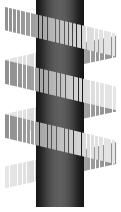
- **Both Selection and Insertion sorts are similar in efficiency**
- **The both have outer loops that scan all elements, and inner loops that compare the value of the outer loop with almost all values in the list**
- **Therefore approximately n^2 number of comparisons are made to sort a list of size n**
- **We therefore say that these sorts are of *order* n^2**
- **Other sorts are more efficient: *order* $n \log_2 n$**



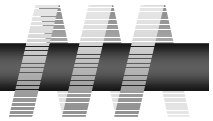
Two-Dimensional Arrays



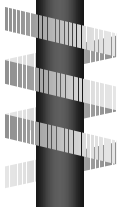
- **A *one-dimensional array* stores a simple list of values**
- **A *two-dimensional array* can be thought of as a table of values, with rows and columns**
- **A two-dimensional array element is referenced using two index values**
- **To be precise, a two-dimensional array in Java is an array of arrays**
- **See `TwoDArray.java` (page 299)**



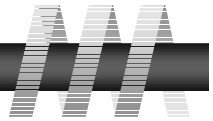
Multidimensional Arrays



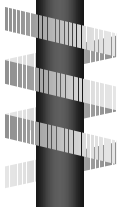
- An array can have as many dimensions as needed, creating a multidimensional array
- Each dimension subdivides the previous one into the specified number of elements
- Each array dimension has its own `length` constant
- Because each dimension is an array of array references, the arrays within one dimension could be of different lengths



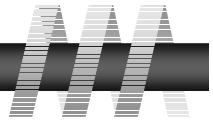
The `Vector` Class



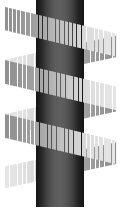
- **An object of class `Vector` is similar to an array in that it stores multiple values**
- **However, a vector**
  - **only stores objects**
  - **does not have the indexing syntax that arrays have**
- **The methods of the `Vector` class are used to interact with the elements of a vector**
- **The `Vector` class is part of the `java.util` package**
- **See `Beatles.java` (page 304)**



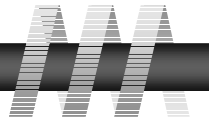
The `Vector` Class



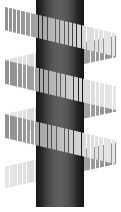
- **An important difference between an array and a vector is that a vector can be thought of as a dynamic, able to change its size as needed**
- **Each vector initially has a certain amount of memory space reserved for storing elements**
- **If an element is added that doesn't fit in the existing space, more room is automatically acquired**



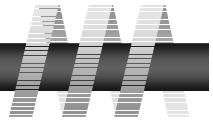
The Vector Class



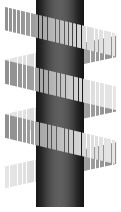
- The `Vector` class is implemented using an array
- Whenever new space is required, a new, larger array is created, and the values are copied from the original to the new array
- To insert an element, existing elements are first copied, one by one, to another position in the array
- Therefore, the implementation of `Vector` in the API is not very efficient for inserting elements



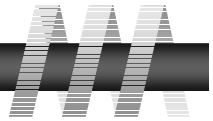
Polygons and Polylines



- **Arrays are often helpful in graphics processing**
- **Polygons and polylines are shapes that are defined by values stored in arrays**
- **A polyline is similar to a polygon except that its endpoints do not meet, and it cannot be filled**
- **See `Rocket.java` (page 307)**
- **There is also a separate `Polygon` class that can be used to define and draw a polygon**



Saving Drawing State



- Each time the `repaint` method is called on an applet, the window is cleared prior to calling `paint`
- An array or vector can be used to store the objects drawn, and redraw them as necessary
- See `Dots2.java` (page 310)