

Chapter 6: Arrays and Vectors

- As our programs get more sophisticated, we need assistance organizing large amounts of data
- Chapter 6 focuses on:
 - array declaration and use
 - arrays of objects
 - parameters and arrays
 - multidimensional arrays
 - the `Vector` class
 - additional techniques for managing strings
 - Sorting

Arrays

- An *array* is an ordered list of values
- Each value has a numeric *index*
- An array of size *N* is indexed from zero to *N-1*
- The following array of integers has a size of 10 and is indexed from 0 to 9

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

Arrays

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


```
scores[4]
```

 refers to the value 67 (the 5th value in the array)
- That expression represents a place to store a single integer, can be used wherever an integer variable can
- For example, it can be assigned a value, printed, 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 newly instantiated array of 10 integers
- See `BasicArray.java`

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, they 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*
- Its common to inadvertently introduce *off-by-one errors* when using arrays

Bounds Checking

- Each array object has a public constant called `length` that stores the size of the array
- It is referenced through 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` and `LetterCount.java`

Array Declarations Revisited

- The brackets of the array type can be associated with the element type or with the name of the array
- Therefore


```
float[] prices;
```

 and


```
float prices[];
```

 are essentially equivalent
- The first format is usually 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[] letter_grades = {'A', 'B', 'C',
                        'D', 'F'};
```

Initializer Lists

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

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 follow the parameter passing rules of that element's type

Arrays of Objects

- The elements of an array can be object references
- The declaration


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

 reserves space to store 25 references to `String` objects
- It does NOT create the `String` objects themselves
- Each object stored in an array must be instantiated separately

Arrays of Objects

- See `GradeRange.java`
- 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`

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 and 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
- An expanded version:
 - scan the list to find the smallest value
 - put it in the first position
 - find the next smallest value
 - put it 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`

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
- An expanded version:
 - 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 `SortGrades.java`

Comparing Sorts

- Both Selection and Insertion Sorts are similar in efficiency
- They 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
- That is approximately n^2 number of comparisons for a list of size n
- We therefore say that these sorts are of *order* n^2
- Other sorts are more efficient: *order* $n \log_2 n$

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 this to develop a generic sort for a set of objects

Multidimensional 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, therefore each row can have a different length

Multidimensional Arrays

- An initializer list can be used to create and set up a multidimensional array
- Each element in the list is itself an initializer list
- Note that each array dimension has its own length constant
- See `TwoDArray.java` and `SodaSurvey.java`

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
- Service methods are used to interact with a vector
- The `Vector` class is part of the `java.util` package
- See `Beatles.java`

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

- A vector 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