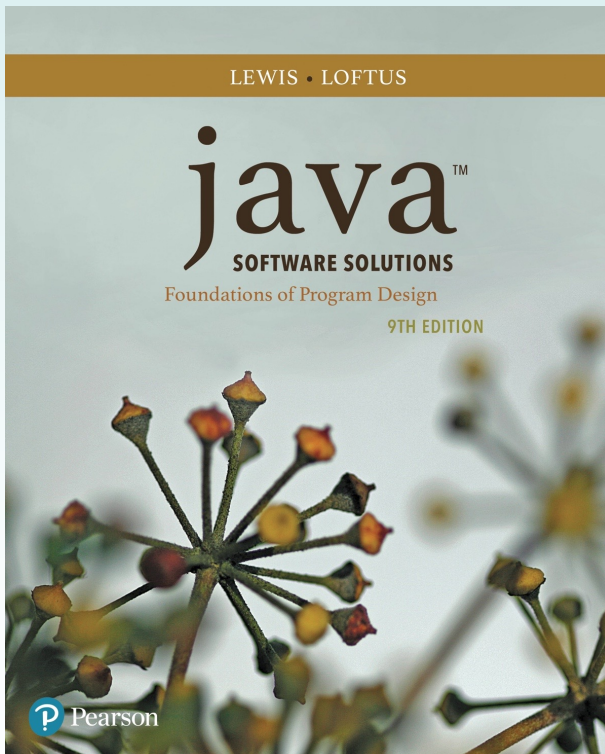


Chapter 8

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
 - polygons and polylines
 - choice boxes

Outline



Declaring and Using Arrays

Arrays of Objects

Variable Length Parameter Lists

Two-Dimensional Arrays

Sorting and Searching

Polygons and Polylines

Array of Color Objects

Choice Boxes

Arrays

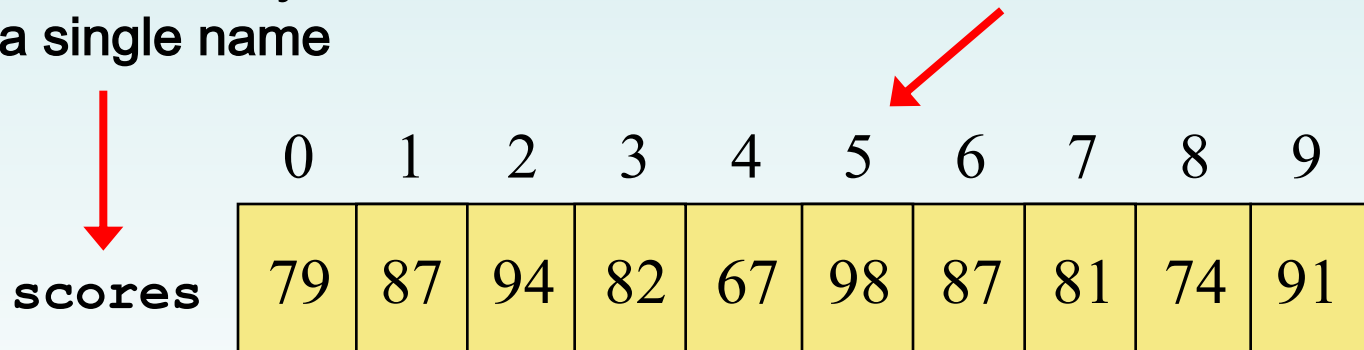
- The `ArrayList` class, introduced in Chapter 5, is used to organize a list of objects
- It is a class in the Java API
- An *array* is a programming language construct used to organize a list of objects
- It has special syntax to access elements
- As its name implies, the `ArrayList` class uses an array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

Arrays

- An array is an ordered list of values:

The entire array
has a single name

Each value has a numeric *index*



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

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

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

Arrays

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

`scores[2]`

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

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

Arrays

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

```
scores[2] = 89;
```

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

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

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

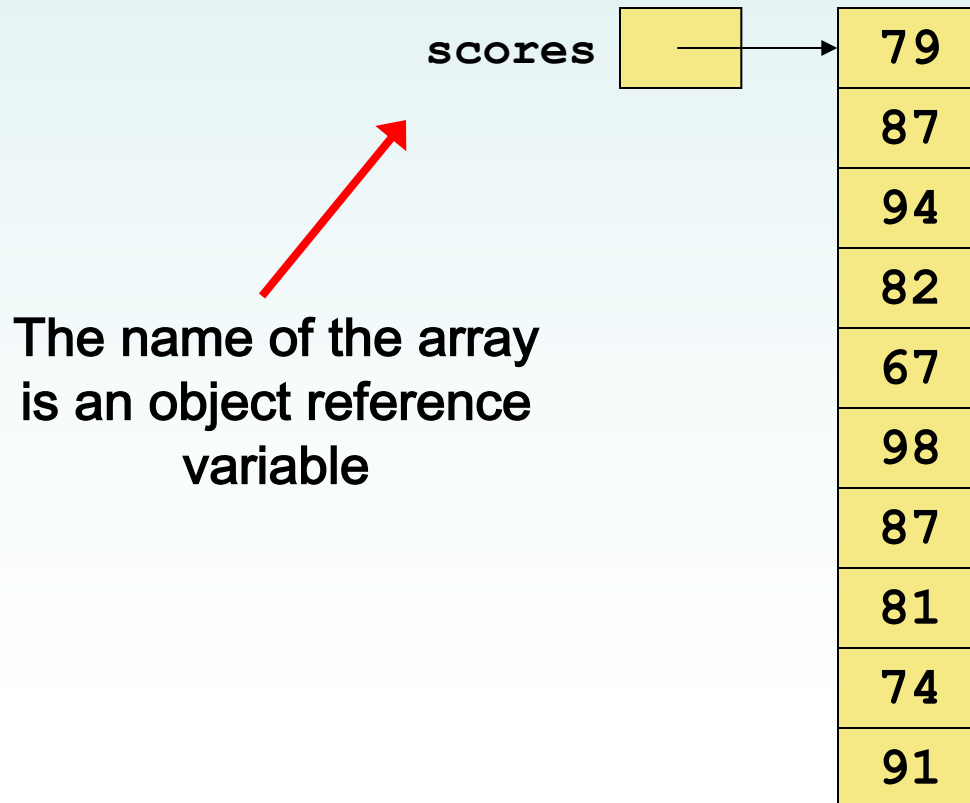
```
pick = scores[rand.nextInt(11)];
```

Arrays

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

Arrays

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



Declaring Arrays

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

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

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

Declaring Arrays

- Some other examples of array declarations:

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

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

```
boolean[] flags;
```

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

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

Using Arrays

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

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

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

Basic Array Example

The array is created with 15 elements, indexed from 0 to 14

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	



After three iterations of the first loop

0	0
1	10
2	20
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	



After completing the first loop

0	0
1	10
2	20
3	30
4	40
5	50
6	60
7	70
8	80
9	90
10	100
11	110
12	120
13	130
14	140



After changing the value of list[5]

0	0
1	10
2	20
3	30
4	40
5	999
6	60
7	70
8	80
9	90
10	100
11	110
12	120
13	130
14	140

Quick Check

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

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

Quick Check

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

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

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

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

Bounds Checking

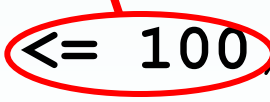
- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element
- That is, the index value must be in range 0 to N-1
- The Java interpreter throws an `ArrayIndexOutOfBoundsException` if an array index is out of bounds
- This is called automatic *bounds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

Bounds Checking

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

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

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

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

Bounds Checking

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

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

Initializer Lists

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

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

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

Initializer Lists

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

Arrays as Parameters

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

Outline

Declaring and Using Arrays



Arrays of Objects

Variable Length Parameter Lists

Two-Dimensional Arrays

Sorting and Searching

Polygons and Polylines

Array of Color Objects

Choice Boxes

Arrays of Objects

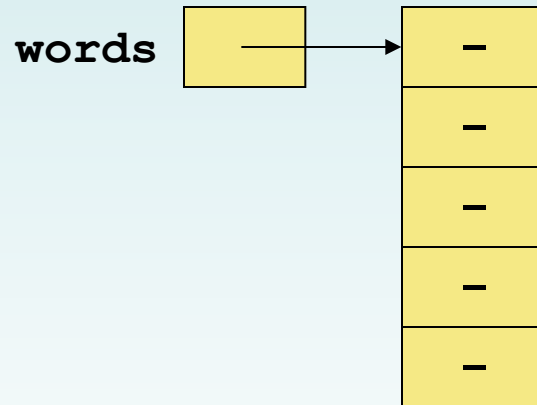
- The elements of an array can be object references
- The following declaration reserves space to store 5 references to `String` objects

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

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

Arrays of Objects

- The `words` array when initially declared:

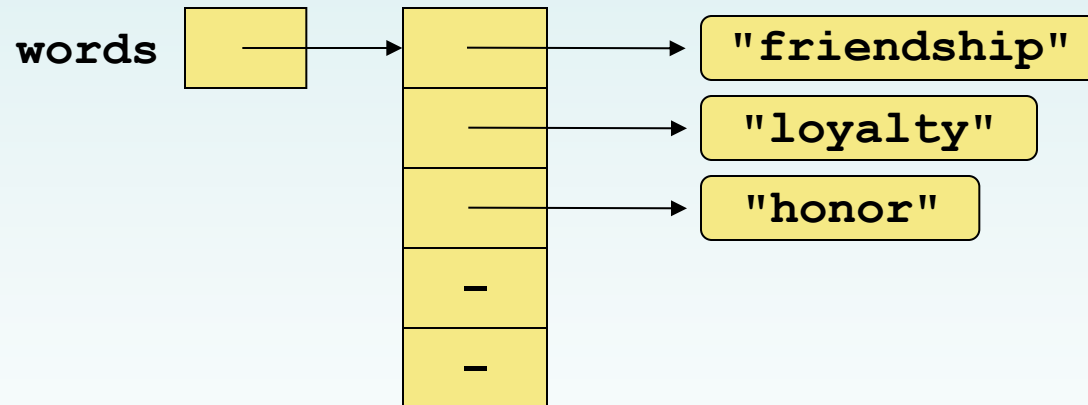


- At this point, the following line of code would throw a `NullPointerException`:

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


Arrays of Objects

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



Arrays of Objects

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

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

Arrays of Objects

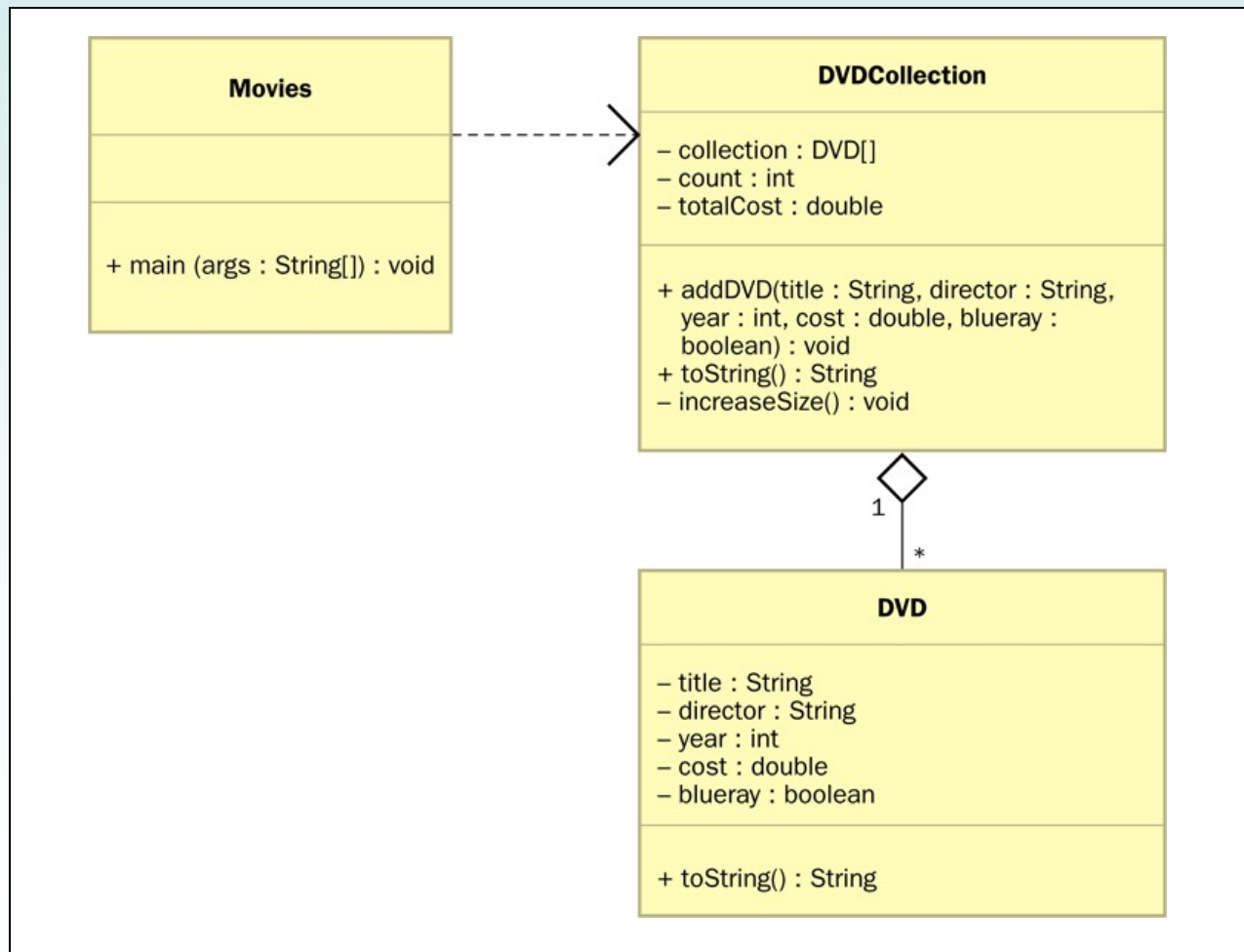
- The following example creates an array of `Grade` objects, each with a string representation and a numeric lower bound
- The letter grades include plus and minus designations, so must be stored as strings instead of `char`
- **See** `GradeRange.java`
- **See** `Grade.java`

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`

Arrays of Objects

- A UML diagram for the `Movies` program:



Command-Line Arguments

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

```
java StateEval pennsylvania texas arizona
```

- See `NameTag.java`

Outline

Declaring and Using Arrays

Arrays of Objects



Variable Length Parameter Lists

Two-Dimensional Arrays

Sorting and Searching

Polygons and Polylines

Array of Color Objects

Choice Boxes

Variable Length Parameter Lists

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

```
// one call to average three values
```

```
mean1 = average(42, 69, 37);
```

```
// another call to average seven values
```

```
mean2 = average(35, 43, 93, 23, 40, 21, 75);
```


Variable Length Parameter Lists

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

Variable Length Parameter Lists

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

Indicates a variable length parameter list

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

element type array name

Variable Length Parameter Lists

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

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

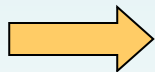
- The varying number of parameters must come last in the formal arguments
- A method cannot accept two sets of varying parameters
- Constructors can also be set up to accept a variable number of parameters
- **See** `VariableParameters.java`
- **See** `Family.java`

Outline

Declaring and Using Arrays

Arrays of Objects

Variable Length Parameter Lists



Two-Dimensional Arrays

Sorting and Searching

Polygons and Polylines

Array of Color Objects

Choice Boxes

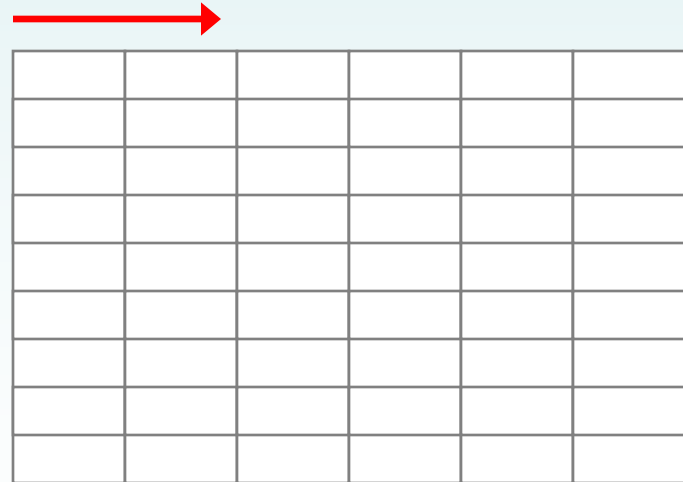
Two-Dimensional Arrays

- 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

one
dimension



two
dimensions



Two-Dimensional Arrays

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

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

- A array element is referenced using two index values:

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

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

Two-Dimensional Arrays

Expression	Type	Description
<code>table</code>	<code>int[][]</code>	2D array of integers, or array of integer arrays
<code>table[5]</code>	<code>int[]</code>	array of integers
<code>table[5][12]</code>	<code>int</code>	integer

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

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



Sorting and Searching

Polygons and Polylines

Array of Color Objects

Choice Boxes

Sorting

- *Sorting* is the process of arranging a list of items in a particular order
- The sorting process is based on specific criteria:
 - sort test scores in ascending numeric order
 - sort a list of people alphabetically by last name
- There are many algorithms, which vary in efficiency, for sorting a list of items
- We will examine two specific algorithms:
 - Selection Sort
 - Insertion Sort

Selection Sort

- The strategy of Selection Sort:
 - select a value and put it in its final place in the 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 in their proper places

Selection Sort

Scan right starting with 3.
1 is the smallest. Exchange 1 and 3.



Scan right starting with 9.
2 is the smallest. Exchange 9 and 2.



Scan right starting with 6.
3 is the smallest. Exchange 6 and 3.



Scan right starting with 6.
6 is the smallest. Exchange 6 and 6.



Swapping

- The processing of the selection sort algorithm includes the *swapping* of two values
- Swapping requires three assignment statements and a temporary storage location
- To swap the values of `first` and `second`:

```
temp = first;  
first = second;  
second = temp;
```

Selection Sort

- Let's look at an example that sorts an array of `Contact` objects
- The `selectionSort` method is a static method in the `Sorting` class
- See `PhoneList.java`
- See `Sorting.java`
- See `Contact.java`

Insertion Sort

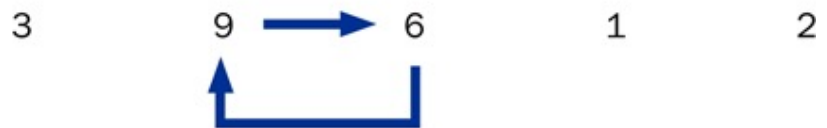
- The strategy of Insertion Sort:
 - pick any item and insert it into its proper place in a sorted sublist
 - repeat until all items have been inserted
- In more detail:
 - consider the first item to be a sorted sublist (of one item)
 - insert the second item into the sorted sublist, shifting the first item as needed to make room to insert the new one
 - insert the third item into the sorted sublist (of two items), shifting items as necessary
 - repeat until all values are inserted into their proper positions

Insertion Sort

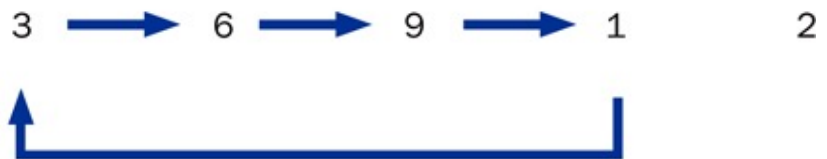
3 is sorted.
Shift nothing. Insert 9.



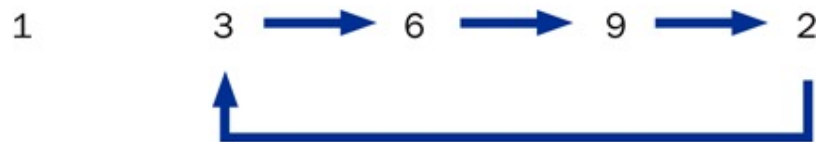
3 and 9 are sorted.
Shift 9 to the right. Insert 6.



3, 6 and 9 are sorted.
Shift 9, 6, and 3 to the right. Insert 1.



1, 3, 6 and 9 are sorted.
Shift 9, 6, and 3 to the right. Insert 2.



All values are sorted.



Comparing Sorts

- The Selection and Insertion sort algorithms 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
- 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 $n \log_2 n$*

$O(N)$, $O(N^2)$, etc.

- What does order N^2 mean?
 - often abbreviate this as $O(N^2)$
- Suppose the processing time is $4N^2 - N + 5$
 - where N is the size of the problem
- If we look how this time grows with large N , the N^2 is the largest part (the part with the largest exponent)
 - terms with smaller exponents are not significant
- We ignore the constant multiplier in front of the N^2 and say the time is $O(N^2)$
 - we are looking for how fast the time grows

Selection Sort Performance

- To sort N elements takes $N-1$ comparisons the first time through the array
 - then $N-2$ comparisons the second time
 - $N-3$ comparisons the third time through
 - ...
 - 1 comparison the last time through
- Total is $1 + 2 + 3 + \dots + (N-1) = N^2/2 - N/2$
- We ignore the $1/2$ multiplier and call this an $O(N^2)$ algorithm
- Time is proportional to N^2

Growth Rates

- Compare these
- Sorting can be $N\log_2(N)$ or N^2
- Searches can be N or $\log_2(N)$
- Note difference
- Note: \log_2 is the number of bits in a quantity and need not be an integer

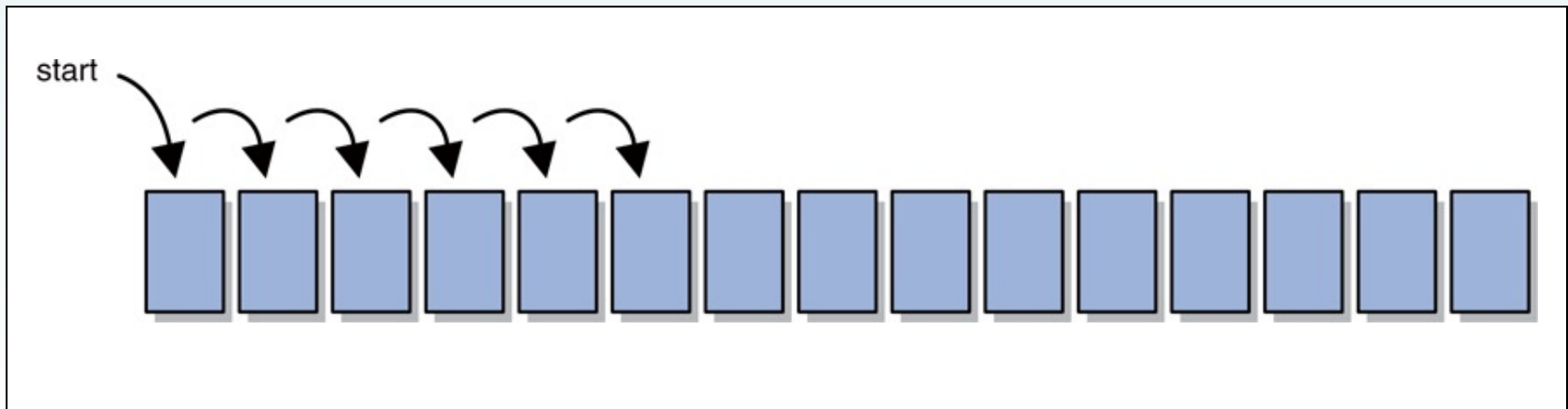
N	$\log_2(N)$	$N\log_2(N)$	N^2
2	1	2	4
4	2	8	16
8	3	24	64
16	4	64	256
32	5	160	1024
64	6	384	4096
128	7	896	16384
256	8	2064	65536
500	8.966	4482.9	250000
10^6	19.93	1.993×10^7	10^{12}

Searching

- *Searching* is the process of finding a *target element* within a group of items called the *search pool*
- The target may or may not be in the search pool
- We want to perform the search efficiently, minimizing the number of comparisons
- Let's look at two classic searching approaches: linear search and binary search

Linear Search

- A linear search begins at one end of a list and examines each element in turn
- Eventually, either the item is found or the end of the list is encountered

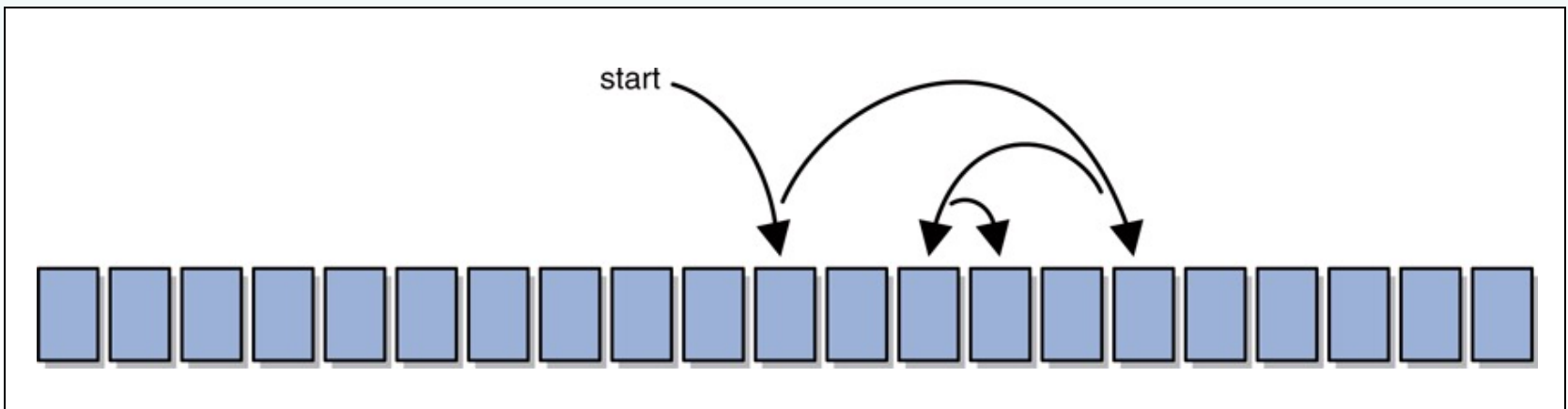


Binary Search

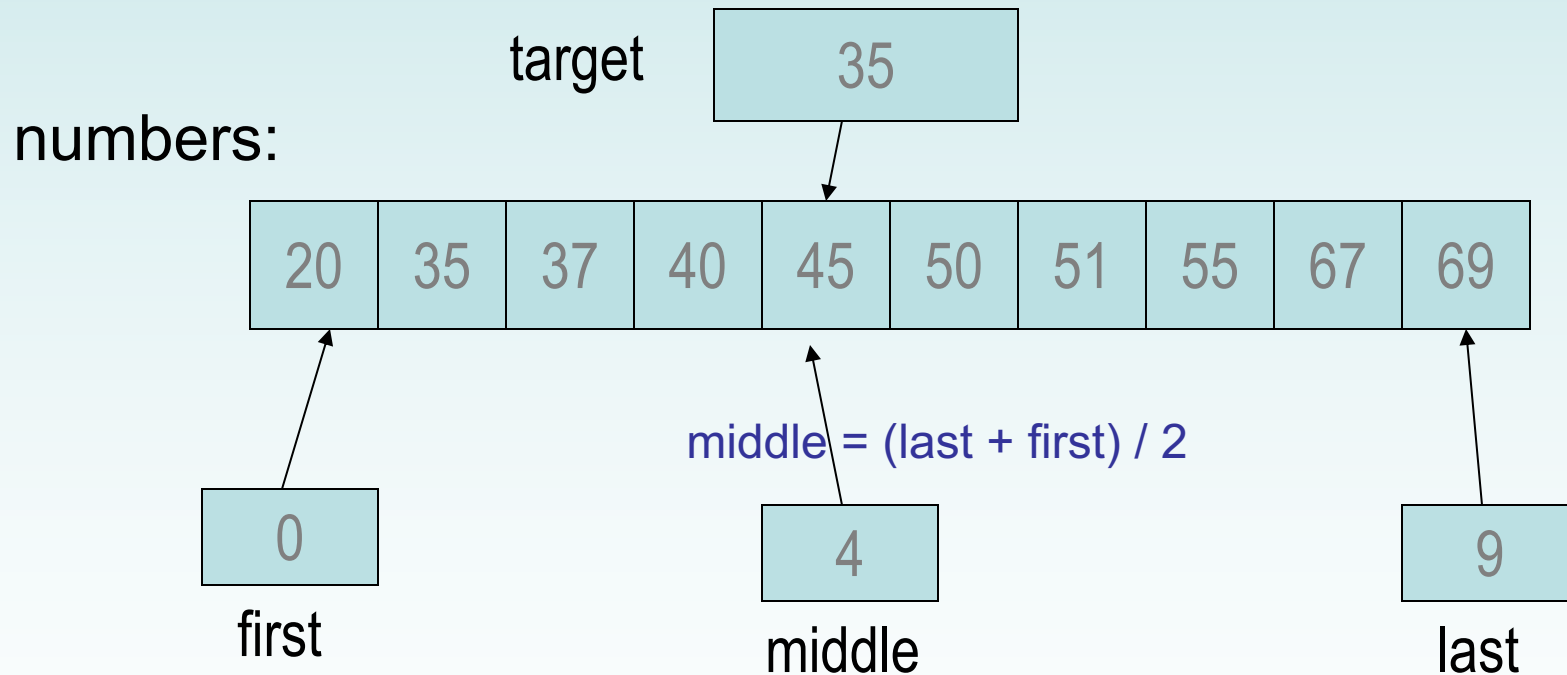
- A *binary search* assumes the list of items in the search pool is sorted
- It eliminates a large part of the search pool with a single comparison
- A binary search first examines the middle element of the list -- if it matches the target, the search is over
- If it doesn't, only one half of the remaining elements need be searched
- Since they are sorted, the target can only be in one half of the other

Binary Search

- The process continues by comparing the middle element of the remaining *viable candidates*
- Each comparison eliminates approximately half of the remaining data
- Eventually, the target is found or the data is exhausted, see next slides

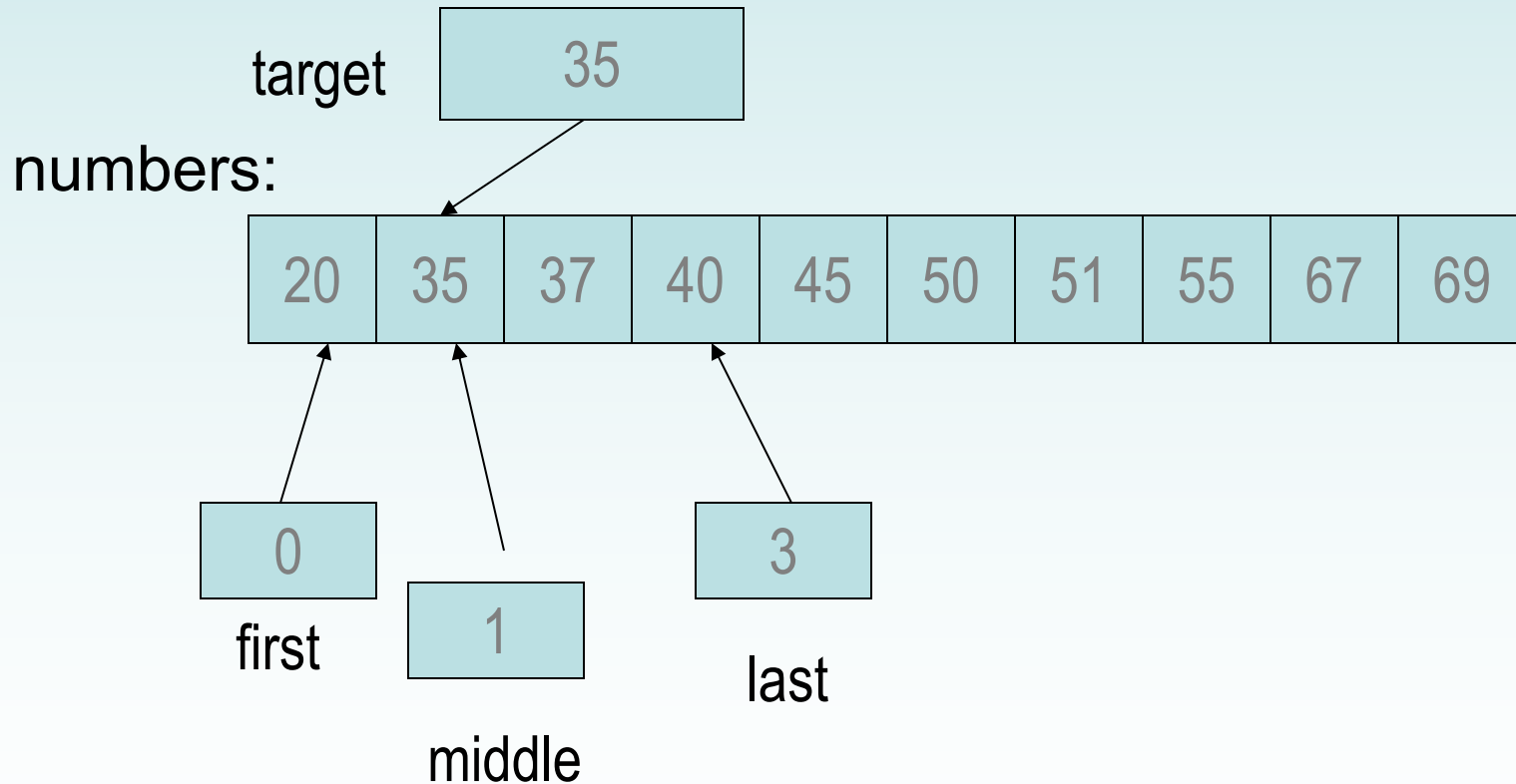


Binary Search, Target Present 1/2



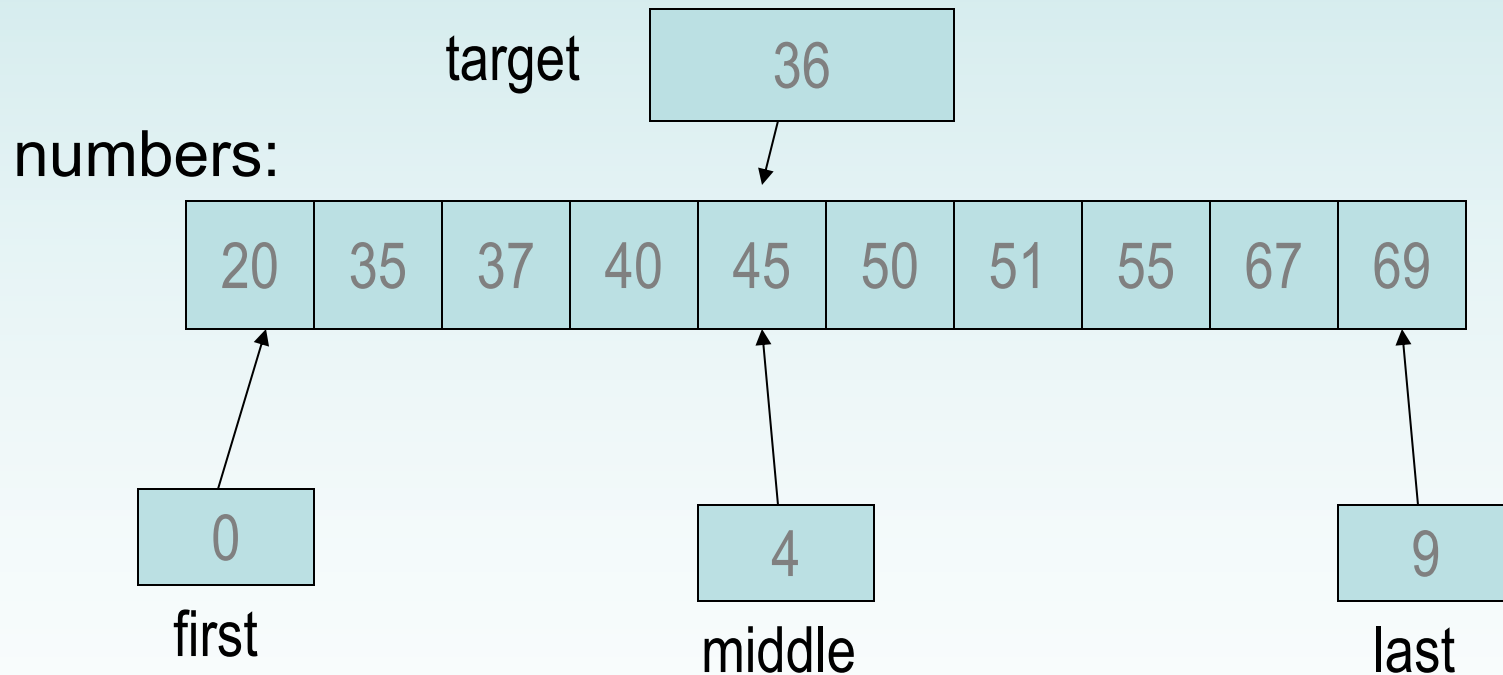
target is less than middle element, numbers[middle],
so set last = (4 - 1) = 3 and try again

Binary Search, Target Present 2/2



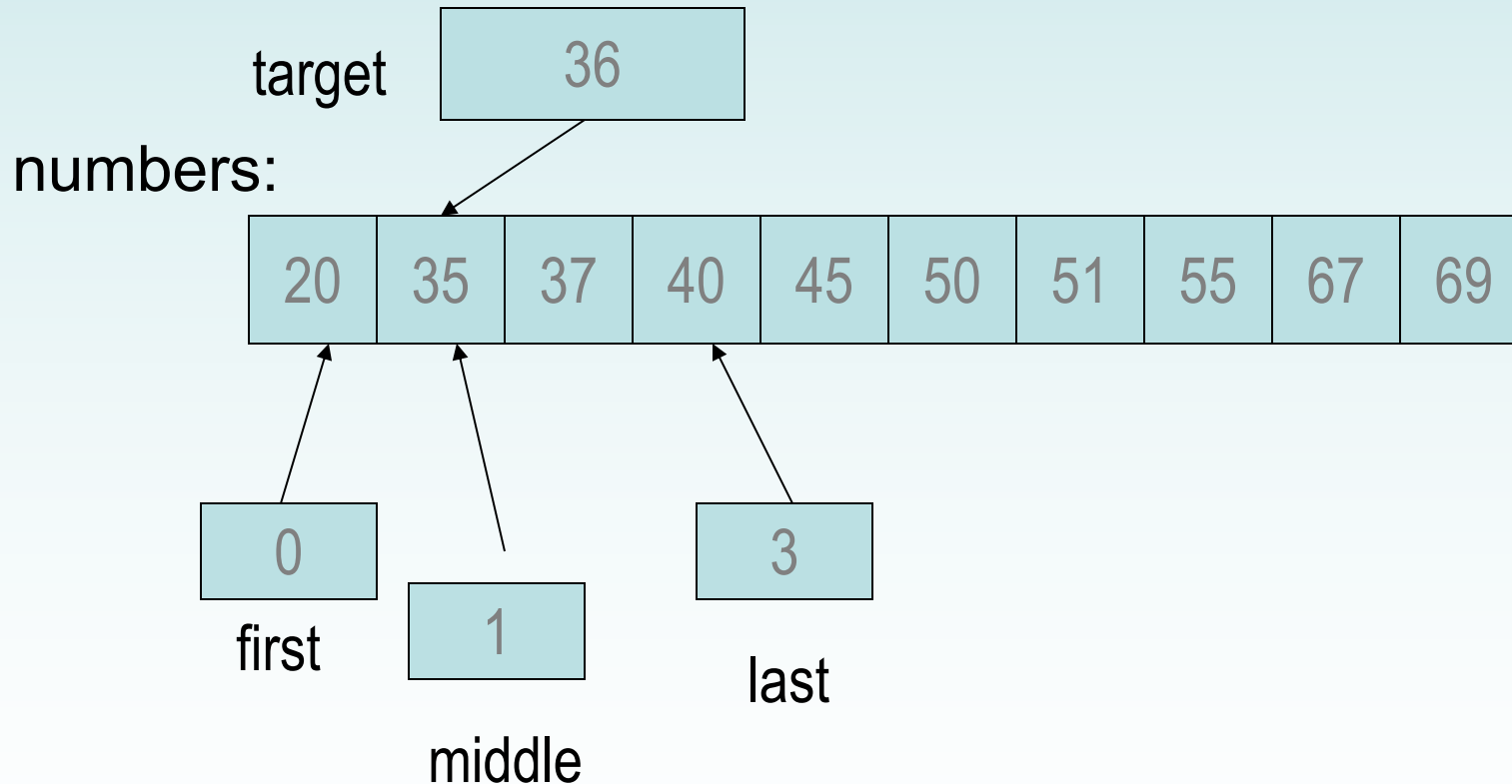
target equals middle element - stop, we have found it

Binary Search, Target Absent, 1/4



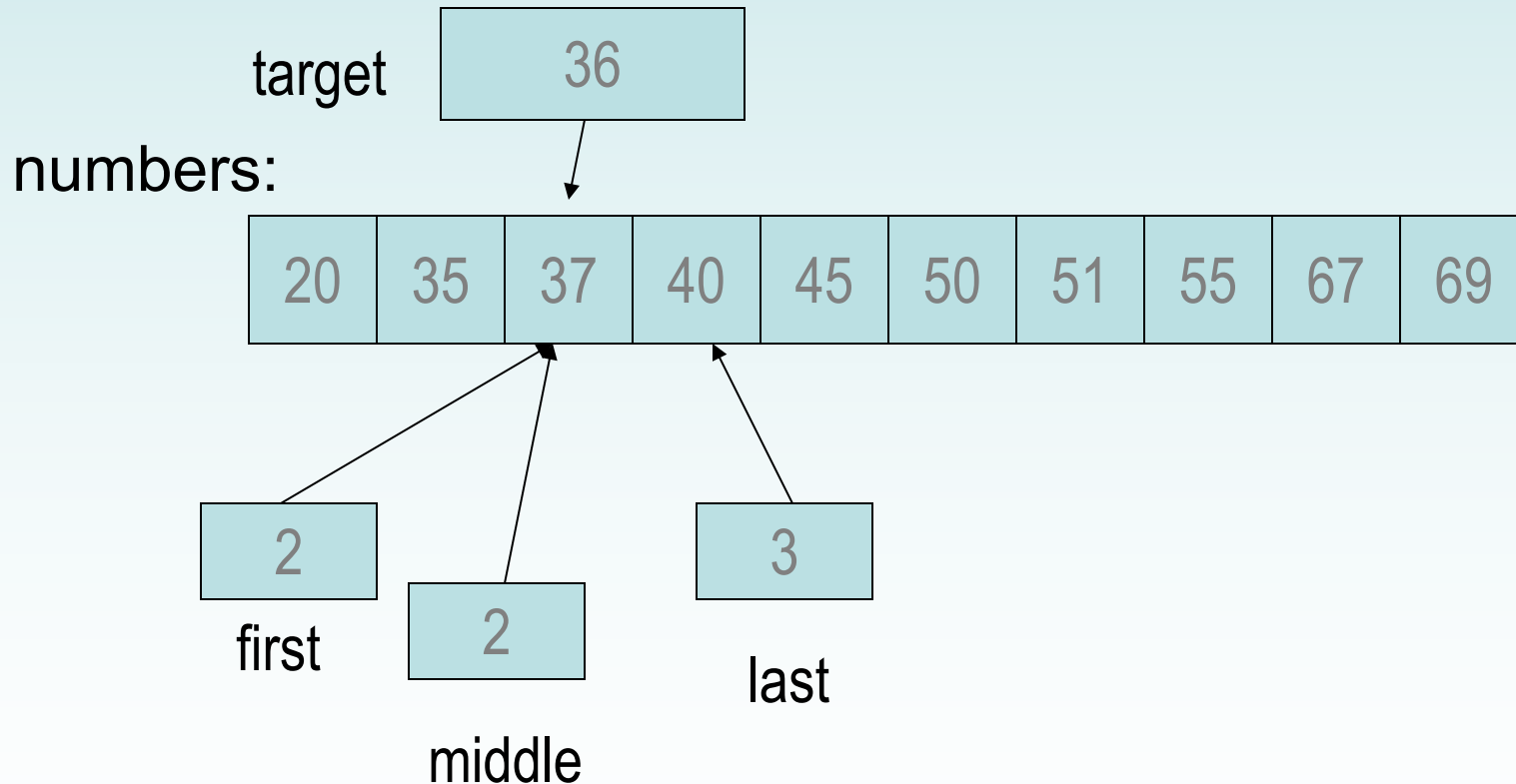
target is less than middle element, numbers[middle], so set
 $\text{last} = (4 - 1) = 3$ and try again

Binary Search, Target Absent, 2/4



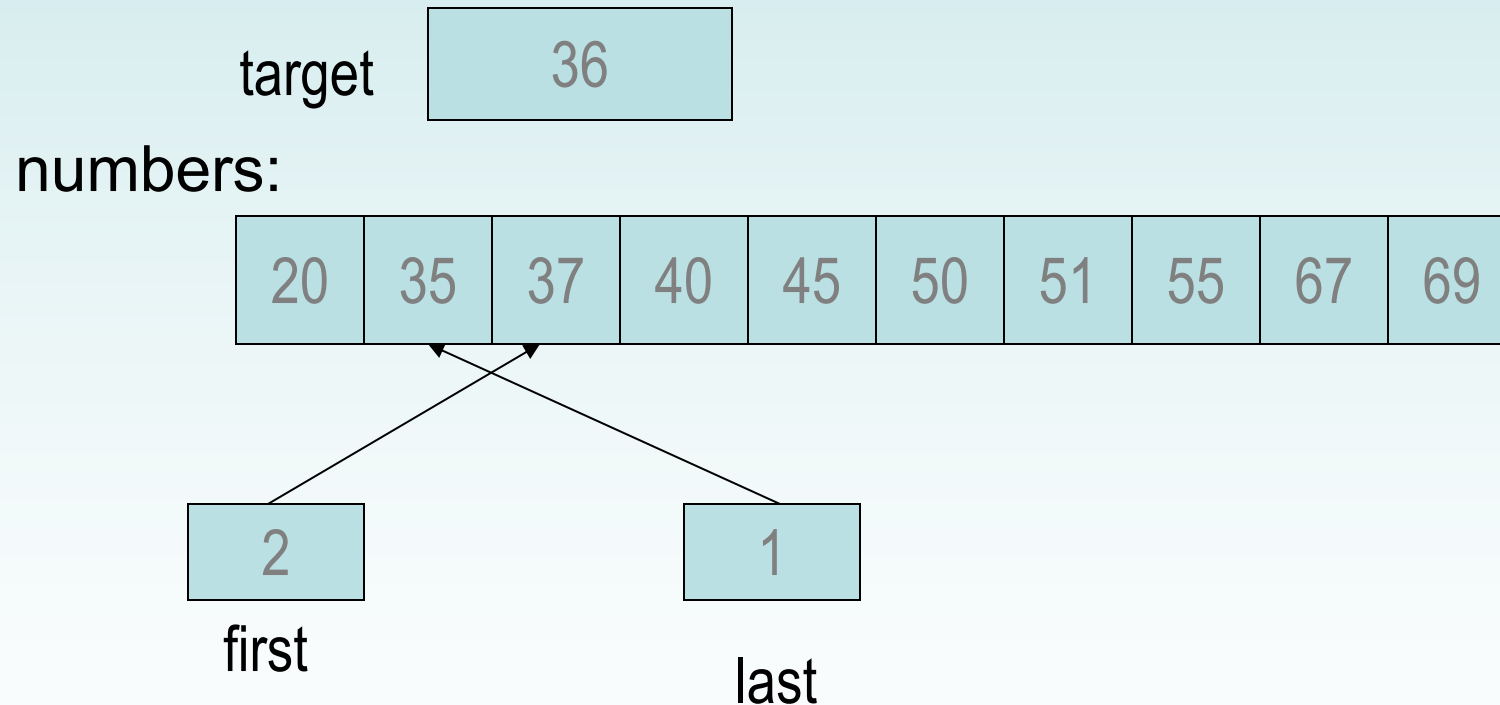
target larger than middle element, so set first = 1 + 1 = 2 and try again

Binary Search, Target Absent, 3/4



target less than middle element, so set last = 2 - 1 = 1 and try again

Binary Search, Target Absent, 4/4



first > last so we know target is not in the array

Searching Example

- The search methods are implemented as static methods in the `Searching` class
- **See** `PhoneList2.java`
- **See** `Searching.java`

Outline

Declaring and Using Arrays

Arrays of Objects

Variable Length Parameter Lists

Two-Dimensional Arrays

Sorting and Searching



Polygons and Polylines

Array of Color Objects

Choice Boxes

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`

Outline

Declaring and Using Arrays

Arrays of Objects

Variable Length Parameter Lists

Two-Dimensional Arrays

Sorting and Searching

Polygons and Polylines



Array of Color Objects

Choice Boxes

Arrays of Color Objects

- Let's look at an example that uses an array of `Color` objects
- When the mouse button is clicked, a colored dot is displayed
- A double-click clears the window
- **See** `Dots.java`

Outline

Declaring and Using Arrays

Arrays of Objects

Variable Length Parameter Lists

Two-Dimensional Arrays

Sorting and Searching

Polygons and Polylines

Array of Color Objects



Choice Boxes

Choice Boxes

- A *choice box* lets the user select one of several options from a drop down menu
- The `JukeBox` example allows the user to select a song from a choice box
- Play and Stop buttons control the song playback
- See `JukeBox.java`

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
 - sorting and searching
 - polygons and polylines
 - choice boxes