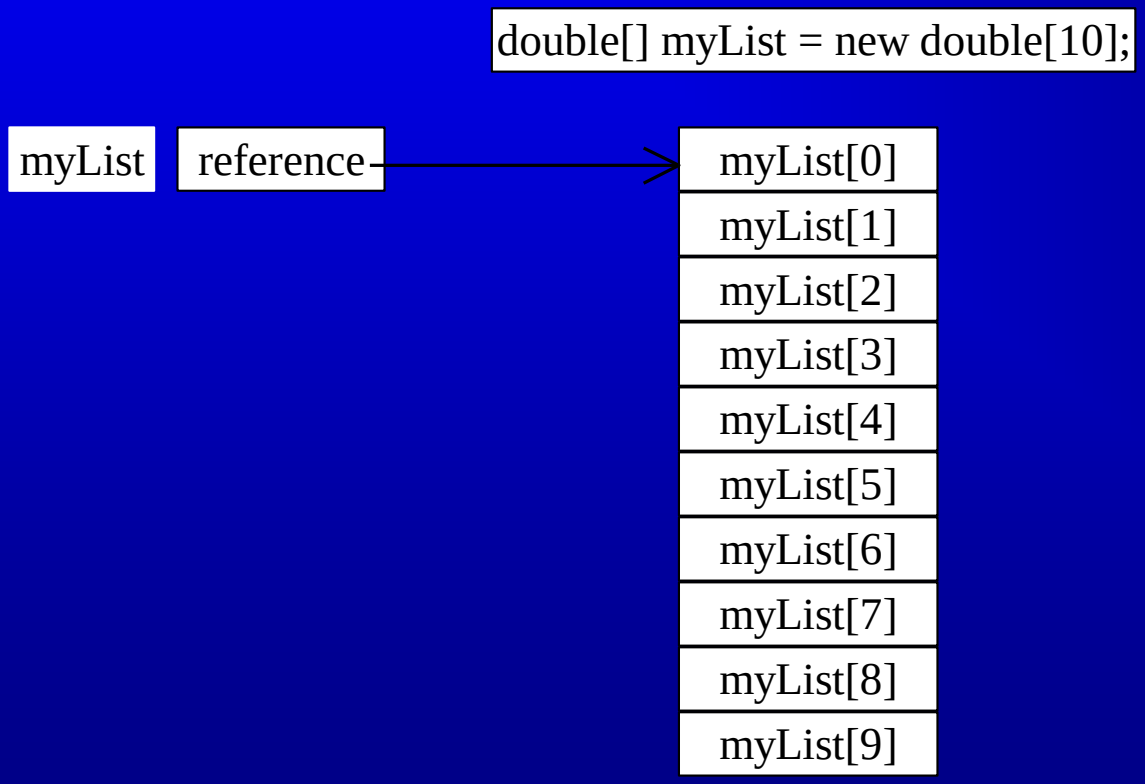
Chapter 5 Arrays

- Introducing Arrays
- Declaring Array Variables, Creating Arrays, and Initializing Arrays
- Passing Arrays to Methods
- Copying Arrays
- Multidimensional Arrays
- Search and Sorting Methods

Introducing Arrays

Array is a data structure that represents a collection of the same types of data.



An Array of 10 Elements of type `double`

Declaring Array Variables

- `datatype[] arrayname;`

  Example:

  ```
  double[] myList;
  ```

- `datatype arrayname[];`

  Example:

  ```
  double myList[];
  ```

Creating Arrays

```
arrayName = new datatype[arraySize];
```

Example:

```
myList = new double[10];
```

`myList[0]` references the first element in the array.

`myList[9]` references the last element in the array.

Declaring and Creating in One Step

- `datatype[] arrayname = new datatype[arraySize];`

```
double[] myList = new double[10];
```

- `datatype arrayname[] = new datatype[arraySize];`

```
double myList[] = new double[10];
```

The Length of Arrays

- Once an array is created, its size is fixed. It cannot be changed. You can find its size using

arrayVariable.length

For example,
myList.length returns 10

Initializing Arrays

- Using a loop:

```
for (int i = 0; i < myList.length; i++)
  myList[i] = i;
```

- Declaring, creating, initializing in one step:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand syntax must be in one statement.

Declaring, creating, initializing Using the Shorthand Notation

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand notation is equivalent to the following statements:

```
double[] myList = new double[4];

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 3.4;

myList[3] = 3.5;
```

CAUTION

Using the shorthand notation, you have to declare, create, and initialize the array all in one statement. Splitting it would cause a syntax error. For example, the following is wrong:

```
double[] myList;

myList = {1.9, 2.9, 3.4, 3.5};
```

Example 5.1
Testing Arrays

- Objective: The program receives 6 numbers from the keyboard, finds the largest number and counts the occurrence of the largest number entered from the keyboard.

  Suppose you entered 3, 5, 2, 5, 5, and 5, the largest number is 5 and its occurrence count is 4.

TestArray

Run

Example 5.2
Assigning Grades

- Objective: read student scores (int) from the keyboard, get the best score, and then assign grades based on the following scheme:
  - Grade is A if score is >= best–10;
  - Grade is B if score is >= best–20;
  - Grade is C if score is >= best–30;
  - Grade is D if score is >= best–40;
  - Grade is F otherwise.

AssignGrade

Run

Passing Arrays to Methods

Java uses *pass by value* to pass parameters to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.

- For a parameter of a primitive type value, the actual value is passed. Changing the value of the local parameter inside the method does not affect the value of the variable outside the method.
- For a parameter of an array type, the value of the parameter contains a

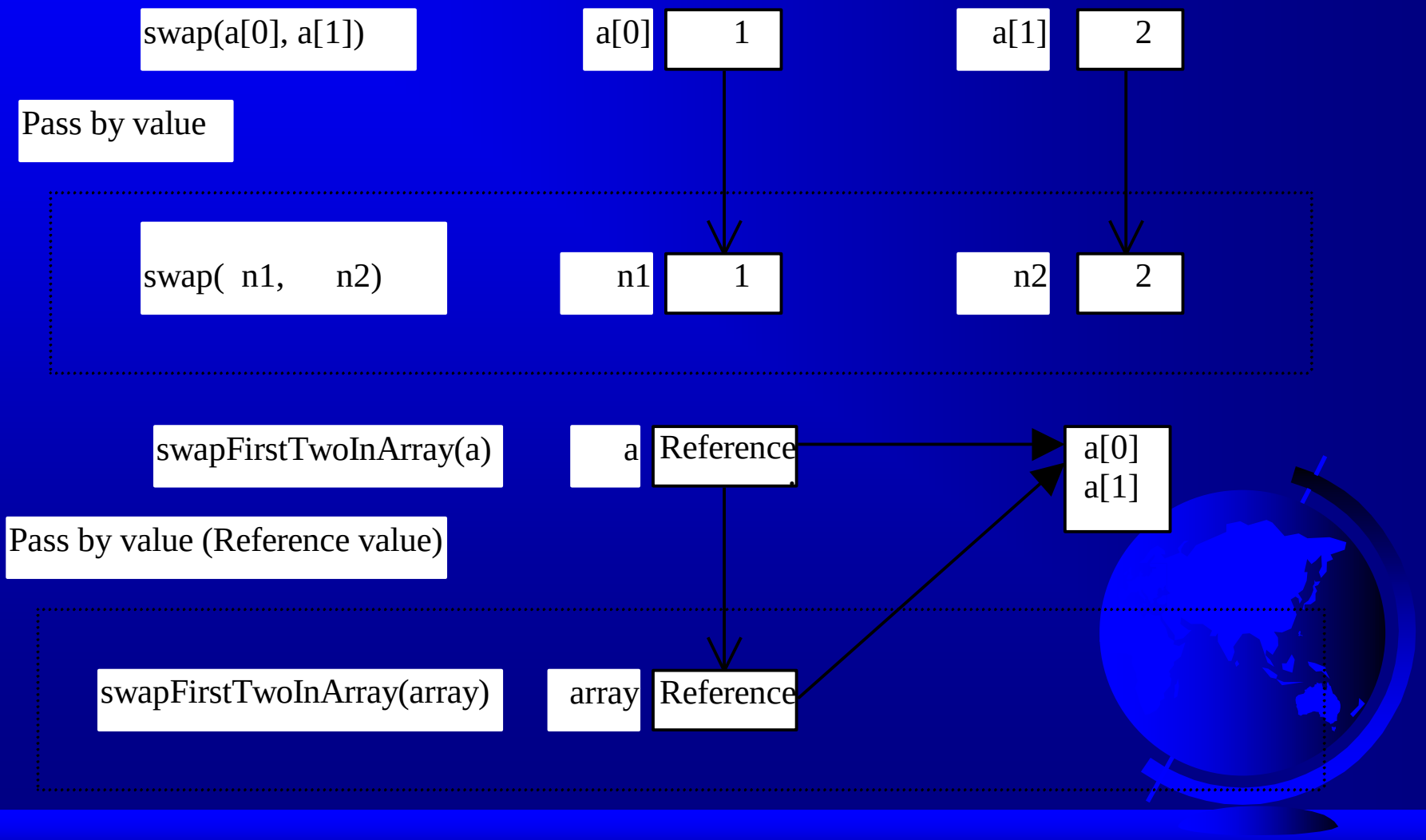
Example 5.3
Passing Arrays as Arguments

- Objective: Demonstrate differences of passing primitive data type variables and array variables.

TestPassArray

Run

Example 5.3, cont.



Example 5.4 Computing Deviation Using Arrays

$$mean = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$deviation = \sqrt{\frac{\sum_{i=1}^{n} (x_i - mean)^2}{n-1}}$$

Deviation

Run

Example 5.5
Counting Occurrence of Each Letter

- Generate 100 lowercase letters randomly and assign to an array of characters.
- Count the occurrence of each letter in the array.
- Find the mean and standard deviation of the counts.

CountLettersInArray

Run

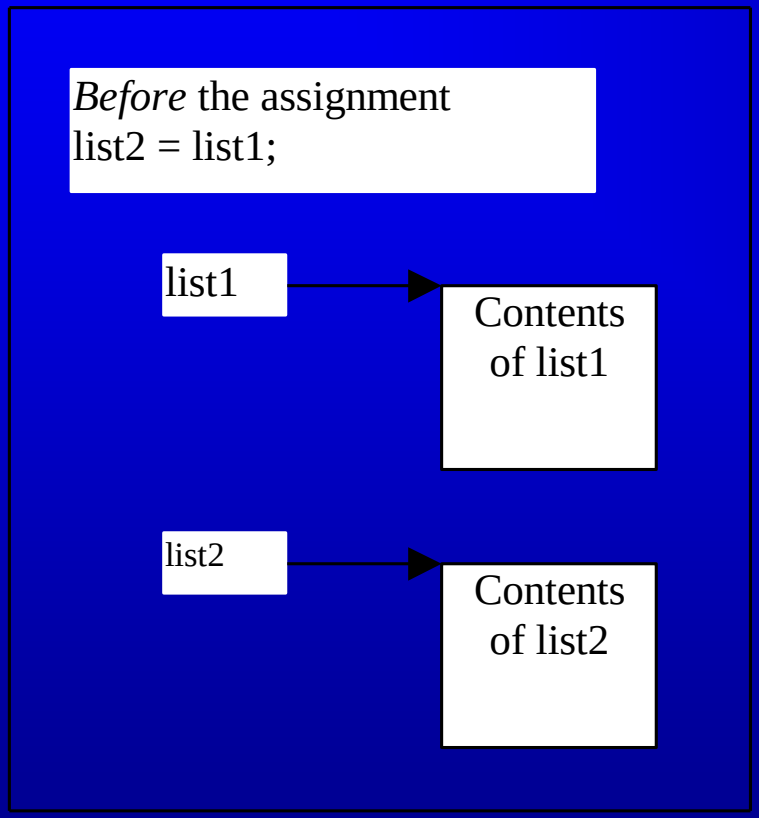
Example 5.6
Copying Arrays

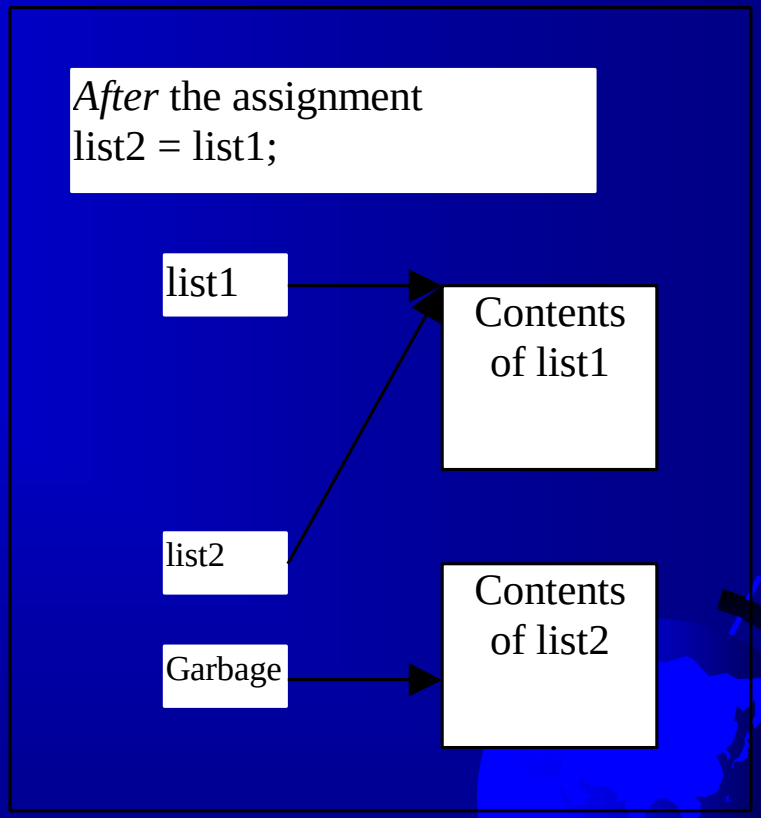
In this example, you will see that a simple assignment cannot copy arrays in the following program. The program simply creates two arrays and attempts to copy one to the other, using an assignment statement.

TestCopyArray

Run

Copying Arrays





Copying Arrays

Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new
  int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++)
   targetArray[i] = sourceArray[i];
```

The `arraycopy` Utility

```
arraycopy(sourceArray, src_pos,
  targetArray, tar_pos, length);
```

Example:

```
System.arraycopy(sourceArray, 0,
  targetArray, 0, sourceArray.length);
```

Multidimensional Arrays

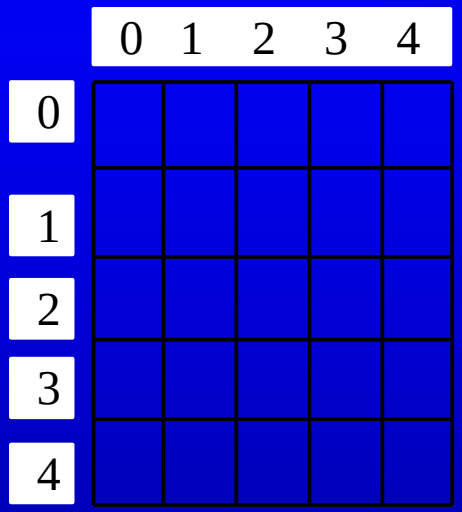
Declaring Variables of Multidimensional Arrays and Creating Multidimensional Arrays

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

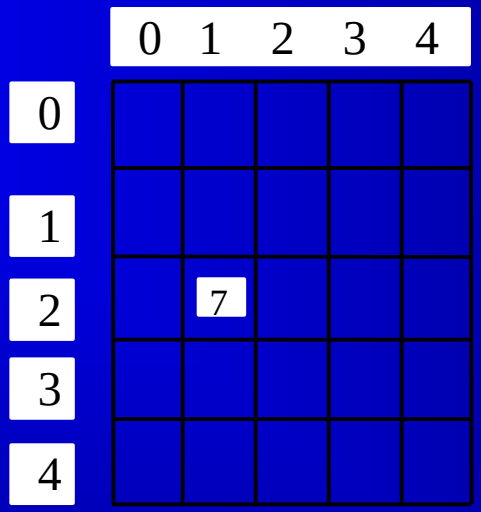
or

```
int matrix[][] = new int[10][10];
matrix[0][0] = 3;

for (int i=0; i<matrix.length; i++)
  for (int j=0; j<matrix[i].length; j++)
  {
    matrix[i][j] = (int)(Math.random()*1000);
  }

double[][] x;
```

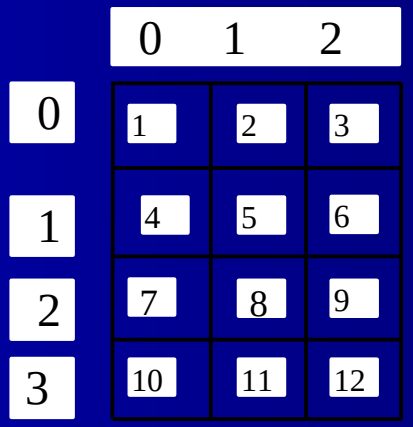
Multidimensional Array Illustration




```
matrix = new int[5][5];
```




```
matrix[2][1] = 7;
```




```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Declaring, Creating, and Initializing Using Shorthand Notations

You can also use a shorthand notation to declare, create and initialize a two-dimensional array. For example,

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

This is equivalent to the following statements:

```
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

Lengths of Multidimensional Arrays

```
int[][] array = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9},
   {10, 11, 12}
};

array.length
array[0].length
array[1].length
array[2].length
```

Ragged Arrays

Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a ragged array*. For example,

```
int[][] matrix = {
   {1, 2, 3, 4, 5},
   {2, 3, 4, 5},
   {3, 4, 5},
   {4, 5},
   {5}
};
```

Example 5.7
Adding and Multiplying Two Matrices

- Objective: Use two-dimensional arrays to create two matrices, and then add and multiply the two matrices.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} & a_{14}+b_{14} & a_{15}+b_{15} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} & a_{24}+b_{24} & a_{25}+b_{25} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} & a_{34}+b_{34} & a_{35}+b_{35} \\ a_{41}+b_{41} & a_{42}+b_{42} & a_{43}+b_{43} & a_{44}+b_{44} & a_{45}+b_{45} \\ a_{51}+b_{51} & a_{52}+b_{52} & a_{53}+b_{53} & a_{54}+b_{54} & a_{55}+b_{55} \end{pmatrix}$$

TestMatrixOperation

Run

Example 5.7 (cont) Adding and Multiplying Two Matrices

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} \end{pmatrix}$$

$$c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j} + a_{i4} \times b_{4j} + a_{i5} \times b_{5j}$$

Example 5.8
Grading Multiple-Choice Test

- Objective: write a program that grades multiple-choice test.

Students' Answers to the Questions:

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

Key to the Questions:

	0	1	2	3	4	5	6	7	8	9
Key	D	B	D	C	C	D	A	E	A	D

Grade Exam

Run

Example 5.9
Calculating Total Scores

- Objective: write a program that calculates the total score for students in a class. Suppose the scores are stored in a three-dimensional array named scores. The first index in scores refers to a student, the second refers to an exam, and the third refers to the part of the exam. Suppose there are 7 students, 5 exams, and each exam has two parts--the multiple-choice part and the programming part. So, scores[i][j][0] represents the score on the multiple-choice part for the i's student on the j's exam. Your program displays the total score for each student, .

TotalScore

Run

Searching Arrays

Searching is the process of looking for a specific element in an array; for example, discovering whether a certain score is included in a list of scores. Searching, like sorting, is a common task in computer programming. There are many algorithms and data structures devoted to searching. In this section, two

Linear Search

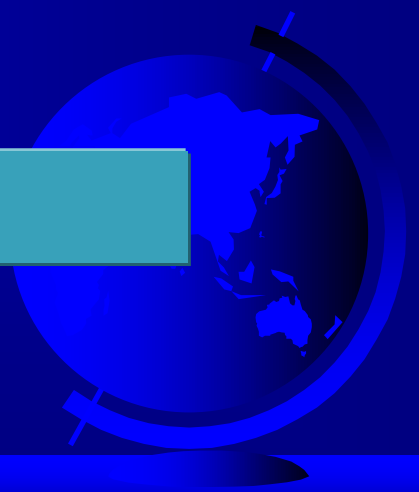
The linear search approach compares the key element, key, with each element in the array list[]. The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found. If a match is made, the linear search returns the index of the element in the array that matches the key. If

Example 5.10
Testing Linear Search

- Objective: Implement and test the linear search method by creating an array of 10 elements of `int` type randomly and then display this array. Prompt the user to enter a key for testing the linear search.

LinearSearch

Run



Binary Search

For binary search to work, the elements in the array must already be ordered. Without loss of generality, assume that the array is in ascending order.

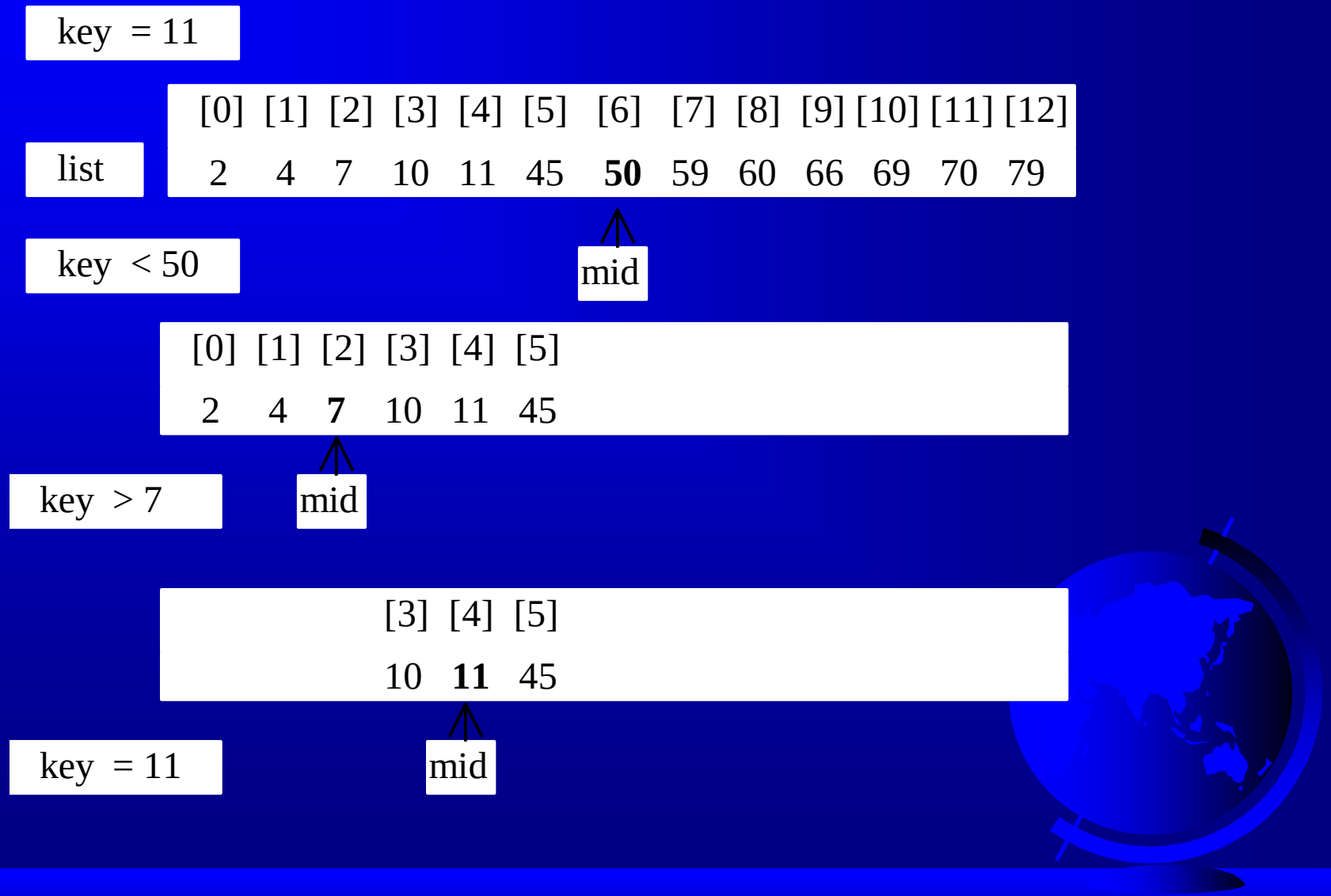
e.g. 2 4 7 10 11 45 50 59 60 66 69 70 79

The binary search first compares the key with the element in the middle of the

Binary Search, cont.

- If the key is less than the middle element, you only need to search the key in the first half of the array.
- If the key is equal to the middle element, the search ends with a match.
- If the key is greater than the middle element, you only need to search the key in the second half of the array.

Binary Search, cont.



Example 5.11
Testing Binary Search

- Objective: Implement and test the binary search method. The program first creates an array of 10 elements of int type. It displays this array and then prompts the user to enter a key

BinarySearch

Run

Example 5.12
Using Arrays in Sorting

- Objective: Use the `selectionSort` method to write a program that will sort a list of double floating-point numbers.

int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted

Sort it to produce 1, 2, 4, 5, 6, 8, 9

2, 9, 5, 4, 8, 1, 6

SelectionSort

Run

Example 5.12: (Cont) Using Arrays in Sorting

int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted

Find the largest element in myList and swap it with the last element in myList.

2, *9*, 5, 4, 8, 1, *6* => 2, *6*, 5, 4, 8, 1, *9* *(size = 7)*

2, 6, 5, 4, *8*, *1* => 2, 6, 5, 4, *1*, *8* *(size = 6)*

2, *6*, 5, 4, *1* => 2, *1*, 5, 4, *6* *(size = 5)*

2, 1, *5*, *4* => 2, 1, *4*, *5*

2, 1, *4* => 2, 1, *4*,

2, 1 => 1, 2

Sort it to produce 1, 2, 4, 5, 6, 8, 9

Exercise 5.5: Bubble Sort

```
int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted
```

Pass 1: 2, 5, 4, 8, 1, 6, 9

Pass 2: 2, 4, 5, 1, 6, 8, 9

Pass 3: 2, 4, 1, 5, 6, 8, 9

Pass 4: 2, 1, 4, 5, 6, 8, 9

Pass 5: 1, 2, 4, 5, 6, 8, 9

Pass 6: 1, 2, 4, 5, 6, 8, 9