Re: Objectives in Methods

- To declare methods, invoke methods, and pass arguments to a method
- To use method overloading and know ambiguous overloading
- To determine the scope of local variables
- To learn the concept of method abstraction
- To know how to use the methods in the `Math` class
- To design and implement methods using stepwise refinement
Objectives in Arrays

° To describe why an array is necessary in programming
° To learn how to declare array reference variables and create arrays
° To initialize the values in an array
° To simplify programming using JDK 1.5 *foreach* loop
° To copy contents from one array to another
° To develop and invoke methods with array arguments and return type
° To sort an array using the selection sort algorithm
° To search elements using the linear or binary search algorithm
° To declare and create multidimensional arrays

Introducing Arrays

° Problem: write a program to read 100 numbers, calculate their average, and find out how many numbers are above the average.

° Observations:
  • The 100 numbers must be stored in variables to accomplish the task
  • are you going to declare 100 variables with the same type?
  • are you doing to repeatedly write almost identical code 100 times?

  **Array + Loops**
Arrays

- Array is a data structure that represents a fixed-size sequential collection of the same types of data (and objects)

| myList[0] | 5.6 |
| myList[1] | 4.5 |
| myList[2] | 3.3 |
| myList[3] | 13.2 |
| myList[4] | 4 |
| myList[5] | 34.33 |
| myList[6] | 34 |
| myList[7] | 45.45 |
| myList[8] | 99.993 |
| myList[9] | 11123 |

Declaring Array Variables

- `datatype[] arrayRefVar;`
  - Example: to declare an array that holds 10 values of type double
to declare an array that holds 100 names of type string

- `datatype arrayRefVar[]; // The style is allowed, but not preferred`
  - Example:
    Example: to declare an array that holds 100 values of type double
Creating Arrays and Default Values

- **Creating arrays**
  
  ```
  arrayRefVar = new datatype[arraySize];
  ```

  Example: to create the array that holds 10 values of type double
to create the array that holds 100 names of type string

- **When an array is created, its elements are assigned the**
  **default value:**

  
  - 0 for the numeric primitive data types,
  - and
  - false for boolean types.

  **Declaring an array does not imply creating the array!**

---

Declaring and Creating in One Step

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

  Examples: double[] myList = new double[10];
  
  String[] myNameList = new String[100];

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

  For example, arrayRefVar.length returns 10
Declaring, creating, initializing by Shorthand Notation

Known as array initializer:

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

This shorthand notation is equivalent to the following statements:

```java
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:

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

Accessing Array Elements and Indexed Variables

° The array elements are accessed through the index.
° The array indices are 0-based, i.e., it starts from 0 to arrayRefVar.length-1.
° Each element in the array is represented using the following syntax, known as an indexed variable: arrayRefVar[index];
° An indexed variable can be used in the same way as a regular variable.

```java
myList[2] = myList[0] + myList[1];
```
Trace Program with Arrays

public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}

Declare array variable values, create an array, and assign its reference to values

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

After the array is created

JDK5: Enhanced foreach Loop

JDK 1.5 introduced a new for loop that enables you to traverse the complete array sequentially without using an index variable. For example, the following code displays all elements in the array myList:

```java
for (double element: myList) {
    System.out.println(element);
}
```

In general, the syntax is

```java
for (elementType element: arrayRefVar) {
    // Process the element
}
```

You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.
Processing Arrays (often with a for loop)

1. Initializing arrays
2. Printing arrays
3. Summing all elements
4. Finding the largest element
5. Finding the smallest index of the largest element

Example: Testing Arrays

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

```java
import javax.swing.JOptionPane;

public class TestArray {
    /** Main method */
    public static void main(String[] args) {
        final int TOTAL_NUMBERS = 6;
        int[] numbers = new int[TOTAL_NUMBERS];
        // Read all numbers and convert string into integer;
        // Find the largest
        // Find the occurrence of the largest number
        // Prepare the result for JOptionPane.showMessageDialog
        // Display the result
    }
}
```

Trace TestArray in NetBeans
Example: Assigning Grades

Objective: read student scores (int), get the best score, and then assign grades based on the 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;

```java
import javax.swing.JOptionPane;
public class AssignGrade {
    /** Main method */
    public static void main(String[] args) {
        // Get number of students to
        // Convert string into integer
        int numberOfStudents = Integer.parseInt(numberOfStudentsString);
        int[] scores = new int[numberOfStudents];   // Array scores
        int best = 0;   // The best score
        char grade; // The grade
        // Read scores from the dialog and convert string into integer
        // find the best score
        // Declare and initialize output string
        // Assign and display grades
        // Display the result
    }
}
```
Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

```java
list2 = list1; // it can copy primitive data type variables, not arrays
```

### Copying Arrays (True Duplications)

Using a loop:
```java
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];

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

Another approach: `arraycopy` method in the java.lang.System class

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

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

Caution: the target array must have been created with its own memory space allocated!

---

Passing Arrays to Methods

```
public static void printArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
}
```

Invoke the method:
```
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```
Pass By Value

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 reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.

Example

```java
public class Test {
    public static void main(String[] args) {
        int x = 1; // x represents an int value
        int[] y = new int[10]; // y represents an array of int values

        m(x, y); // Invoke m with arguments x and y

        System.out.println("x is " + x); // what is output
        System.out.println("y[0] is " + y[0]); // what is output
    }

    public static void m(int number, int[] numbers) {
        number = 1001; // Assign a new value to number
        numbers[0] = 5555; // Assign a new value to numbers[0]
    }
}
```
Call Stack

When invoking \( m(x, y) \), the values of \( x \) and \( y \) are passed to \( \text{number} \) and \( \text{numbers} \). Since \( y \) contains the reference value to the array, \( \text{numbers} \) now contains the same reference value to the same array.

Swapping Example: Passing Arrays as Arguments

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

```java
public class TestPassArray {
    /** Main method */
    public static void main(String[] args) {
        int[] a = {1, 2};
        swap(a[0], a[1]); // what values before and after?
        swapFirstTwoInArray(a); // what values before and after?
    }

    /** Swap two variables – Listing 5.2*/
    public static void swap(int n1, int n2) {
        int temp = n1;
        n1 = n2;
        n2 = temp;
    }

    /** Swap the first two elements in the array */
    public static void swapFirstTwoInArray(int[] array) {
        int temp = array[0];
        array[0] = array[1];
        array[1] = temp;
    }
}
```
Returning an Array from a Method

An array can be the argument of a method, but also a return value.

Example:

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Trace it!

Example: Counting the Occurrences of Each Letter

Problem: generate 100 lowercase letters randomly using the getRandomLowerCaseLetter method of the RandomCharacter class in Listing 5.6, and assign them to an array of characters. Then, count the occurrences of each letter in the array using

```java
Public class CountLettersInArray{
    public static void main(String args[]){
        // Declare and create an array
        char[] chars = createArray();
        // Display the array
        System.out.println("The lowercase letters are:");
        displayArray(chars);
        // Count the occurrences of each letter
        int[] counts = countLetters(chars);
        // Display counts
        System.out.println();
        System.out.println("The occurrences of each letter are:");
        displayCounts(counts);
    }
}
```
Example: Counting the Occurrences of Each Letter

How to count the occurrences of each letter in the array?

```java
public static int[] countLetters(char[] chars) {
    // declare and create an array of 26 integers
    int[] counts = new int[26];
    // for each letter in the array, count it
    for (int i = 0; i < chars.length; i++)
        counts[chars[i] - 'a'] ++;
    return counts;
}
```

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 is a common task in computer programming.
- Two commonly used approaches are **linear search** and **binary search**.

```
list [0] [1] [2] ...
key  Compare key with list[i] for i = 0, 1, ...
```
### Linear Search Animation

- The linear search compares the key *sequentially* with each element in the array.
- Return the index if matched, or -1 if exhausted.

<table>
<thead>
<tr>
<th>Key</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
<tr>
<td>3</td>
<td>6 4 1 9 7 3 2 8</td>
</tr>
</tbody>
</table>

### From Idea to Solution

```java
/** The method for finding a key in the list */
public static int linearSearch(int[] list, int key) {
    for (int i = 0; i < list.length; i++)
        if (key == list[i])
            return i;
    return -1;
}
```

```java
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```
**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 array.

- If the key is less than the middle element…
- If the key is equal to the middle element, return the index
- If the key is greater than the middle element …
- If the key is not found finally, return \(- (\text{insertion point} + 1)\)

**Binary Search, cont.**

key is 11

key < 50

<table>
<thead>
<tr>
<th>low</th>
<th>mid</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
</tr>
<tr>
<td>[6]</td>
<td>[7]</td>
<td>[8]</td>
</tr>
<tr>
<td>[9]</td>
<td>[10]</td>
<td>[11]</td>
</tr>
<tr>
<td>[12]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

key < 50

list

| 2 | 4 | 7 | 10 | 11 | 45 | 50 | 59 | 60 | 66 | 69 | 70 | 79 |
Binary Search, cont.

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>45</td>
<td>50</td>
<td>59</td>
<td>60</td>
<td>66</td>
<td>69</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

key is 51

key > 50

What is the condition that the key is not found?

From Idea to Solution

/** Use binary search to find the key in the list */

public static int binarySearch(int[] list, int key) {
    int low = 0;
    int high = list.length - 1;

    while (high >= low) { // what if “>=” replaced by “>”
        int mid = (low + high) / 2;
        if (key < list[mid])
            high = mid - 1;
        else if (key == list[mid])
            return mid; // return the index if matching
        else
            low = mid + 1;
    }

    return -(1 + low); // return -(insertion point + 1) otherwise
The Arrays Class and Arrays.binarySearch Method

Java has a Arrays Class which contains various methods for searching, sorting, comparing, etc.

Java provides several overloaded binarySearch methods for searching a key in an array of int, double, char, short, long, and float in the java.util.Arrays class.

For example, the following code searches the keys in an array of numbers and an array of characters.

```java
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("Index is " + java.util.Arrays.binarySearch(list, 11));
// what is the return?

char[] chars = {‘a’, ‘c’, ‘g’, ‘x’, ‘y’, ‘z’};
System.out.println("Index is " + java.util.Arrays.binarySearch(chars, ‘t’));
// what is the return?
```

* For the binarySearch method to work, the array must be pre-sorted in increasing order. How to do sorting?

Sorting Arrays

* Sorting, like searching, is a common task in computer programming.
* For instance, if you wanted to display the grades in alphabetical order.
* Two simple, intuitive algorithms: selection sort and insertion sort.
  • More: Bubble sort, Gap sort, ...

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

What do you think now?
```
Selection Sort

Selection sort finds the largest number in the list and places it last. It then finds the largest number remaining and places it next to last, and so on until the list contains only a single number.

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

From Idea to Solution

```
for (int i = list.length - 1; i >= 1; i--) {
    select the largest element in list[0..i];
    swap the largest with list[i], if necessary;
    // list[i] is in place.
    // The next iteration applies on list[0..i-1]
}
```

```
From Idea to Solution, Cont.

```java
for (int i = list.length - 1; i >= 1; i--) {
    select the largest element in list[0..i];
    swap the largest with list[i], if necessary;
    // list[i] is in place.
    // The next iteration applies on list[0..i-1]
}
```

Expand

```java
// Find the maximum in the list[0..i]
double currentMax = list[0];
for (int j = 1; j <= i; j++) {
    if (currentMax < list[j]) {
        currentMax = list[j];
        currentMaxIndex = j;
    }
}
```

From Idea to Solution, Cont.

```java
for (int i = list.length - 1; i >= 1; i--) {
    select the largest element in list[0..i];
    swap the largest with list[i], if necessary;
    // list[i] is in place.
    // The next iteration applies on list[0..i-1]
}
```

Expand

```java
// Swap list[i] with list[currentMaxIndex] if necessary;
if (currentMaxIndex != i) {
    list[currentMaxIndex] = list[i];
    list[i] = currentMax;
}
Wrap it in a Method

```java
/** The method for sorting the numbers */
public static void selectionSort(double[] list) {
    for (int i = list.length - 1; i >= 1; i--) {
        // Find the maximum in the list[0..i]
        double currentMax = list[0];
        int currentMaxIndex = 0;
        for (int j = 1; j <= i; j++) {
            if (currentMax < list[j]) {
                currentMax = list[j];
                currentMaxIndex = j;
            }
        }
        // Swap list[i] with list[currentMaxIndex] if necessary;
        if (currentMaxIndex != i) {
            list[currentMaxIndex] = list[i];
            list[i] = currentMax;
        }
    }
}
```

Invoke it
selectionSort(yourList)

Insertion Sort
The insertion sort algorithm sorts a list of values by repeatedly inserting an unsorted element into a sorted sublist until the whole list is sorted.

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

```
2 9 5 4 8 1 6
2 5 9 4 8 1 6
2 4 5 8 9 1 6
1 2 4 5 8 9
```
How to Insert? (6.8.2)

The insertion sort algorithm sorts a list of values by repeatedly inserting an unsorted element into a sorted sublist until the whole list is sorted.

```
[0] [1] [2] [3] [4] [5] [6]
list 2 5 9 4
Step 1: Save 4 to a temporary variable currentElement
```

```
[0] [1] [2] [3] [4] [5] [6]
list 2 5 9
Step 2: Move list[2] to list[3]
```

```
[0] [1] [2] [3] [4] [5] [6]
list 2 5 9
Step 3: Move list[1] to list[2]
```

```
[0] [1] [2] [3] [4] [5] [6]
list 2 4 5 9
Step 4: Assign currentElement to list[1]
```

From the Idea to the Solution

```java
def insertion_sort(list):
    for i in range(1, len(list)):
        # insert list[i] into a sorted sublist list[0] ... list[i-1] so that the resulted sublist list[0] ... list[i] is sorted
        current_element = list[i]
        for k in range(i, 0, -1):
            if list[k-1] > current_element:
                list[k] = list[k-1]
            else:
                break
        list[k] = current_element
```

Invoke it

```
insertion_sort(your_list)
```
The Arrays.sort Method

Java provides several overloaded sort methods for sorting an array of int, double, char, short, long, and float in the java.util.Arrays class.

For example, the following code sorts an array of numbers and an array of characters.

```java
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);

cchar[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars);
```

The Arrays.equals Method

We can use the equals method to check if two arrays are equal (they have the same contents, instead of the same reference; which one is stronger?).

```java
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 10};
int[] list3 = {4, 2, 7, 10};
System.out.println(java.util.Arrays.equals(list1, list2));
System.out.println(java.util.Arrays.equals(list1, list3));
```
When we would ask for a two-dimensional (or multi-dimensional) array? A matrix for example.

<table>
<thead>
<tr>
<th>Name</th>
<th>Exam 1</th>
<th>Exam 2</th>
<th>Exam 3</th>
<th>Homework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name 1</td>
<td>75</td>
<td>85</td>
<td>95</td>
<td>100</td>
</tr>
<tr>
<td>Name 2</td>
<td>50</td>
<td>40</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>Name 3</td>
<td>90</td>
<td>95</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Name 4</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Name 10</td>
<td>100</td>
<td>20</td>
<td>90</td>
<td>80</td>
</tr>
</tbody>
</table>

How can we represent the data structure that holds this 11 x 5 table of String values? or 5 x 11 table?

How would we assign a new Homework grade to student "Name 1" of 60 percent?

How would we access the Exam 3 grade for student "Name 2"?

```java
// Declare array ref var
dataType[][] matrix;

// Create array and assign its reference to variable
refVar = new dataType[11][5]; // or dataType[5][11]

// Combine declaration and creation in one statement
dataType[][] refVar = new dataType[11][5];
```
### Two-dimensional Array Illustration

```
0 1 2 3 4
0
1
2
3
4
```

```
0 1 2 3 4
0
1 2 3 4
0 1 2 3 4
```

```
matrix = new int[5][5];
matrix[2][1] = 7;
```

```
What is matrix[1, 2]?
```

```
matrix.length? 5
matrix[0].length? 5
```

```
array = new int[5][5],
array[2][1] = 7;
```

```
array.length? 4
array[0].length? 3
```

### Declaring, Creating, and Initializing Using Shortcut

You can also use an array initializer 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}
};
```

```
array.length? 4
array[0].length? 3
```

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

Same as

```
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;
```
Two-dimensional Arrays

How can we represent the data structure that holds this 10 x 5 table of String values?

How would we assign a new Homework grade to student "Name 1" of 60 percent?

How would we access the Exam 3 grade for student "Name 2"?

```java
int score = (new Integer (grades[2][3])).intValue(); //Ch.10, page357
```

Lengths of Two-dimensional Arrays

```java
int[][] x = new int[3][4];
```

```
x
x[0] x[0][0] x[0][1] x[0][2] x[0][3] x[0].length is 4
x[1][0] x[1][1] x[1][2] x[1][3] x[1].length is 4
```

ArrayIndexOutOfBoundsException
Processing 2D Arrays

- Initializing arrays with random values
  ```java
  for (int row = 0; row < matrix.length; row++)
    for (int column = 0; column < matrix[row].length; column++)
      matrix[row][column] = (int)(Math.random() * 100);
  ```

- Printing arrays
- Summing all array elements
- Summing elements by rows
  ```java
  for (int column = 0; column < matrix[0].length; column++){
      int sum = 0;
      for (row = 0; row < matrix.length; row++)
        sum += matrix[row][column];
      System.out.println("Sum for column " + column + " is " + sum);
  }
  ```

Example: Grading Multiple-Choice Test

- Objective: write a program that grades multiple-choice test and shows the number of correct answers of each student.

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 D A E A D
```
**Code: Grading Multiple-Choice Test**

```java
public class GradeExam {
    /** Main method */
    public static void main(String args[]) {
        // Students' answers to the questions
        char[][] answers = {
            {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
            ..........
        };
        // Key to the questions
        // Grade all answers
        for (int i = 0; i < answers.length; i++) {
            int correctCount = 0;
            // Grade each student
            for (int j = 0; j < answers[i].length; j++) {
                if (answers[i][j] == keys[j])
                    correctCount++; // we can define a correctCount array to keep the records too
            }
            System.out.println("Student "+i+"'s correct count is "+correctCount);
        }
    }
}
```

**Multidimensional Arrays**

Occasionally, you will need to represent $n$-dimensional data structures. In Java, you can create $n$-dimensional arrays for any integer $n$.

The way to declare and create two-dimensional arrays can be generalized to declare and create $n$-dimensional arrays for $n \geq 3$.

For example, a class with 10 students, 5 exams, each exam has 2 parts.

```java
double[][][] scores = new double[10][5][2];
```
Exercise 6.18 Bubble Sort

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

The bubble-sort algorithm makes several iterations through the array.

On each iteration, successive neighboring pairs are compared. If a pair is in decreasing order, its values are swapped; otherwise, the values remain unchanged.

The technique is called a bubble sort or sinking sort because the smaller values gradually "bubble" their way to the top and the larger values sink to the bottom.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Sorted Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 5, 4, 8, 1, 6, 9</td>
</tr>
<tr>
<td>2</td>
<td>2, 4, 5, 1, 6, 8, 9</td>
</tr>
<tr>
<td>3</td>
<td>2, 4, 1, 5, 6, 8, 9</td>
</tr>
<tr>
<td>4</td>
<td>2, 1, 4, 5, 6, 8, 9</td>
</tr>
<tr>
<td>5</td>
<td>1, 2, 4, 5, 6, 8, 9</td>
</tr>
<tr>
<td>6</td>
<td>1, 2, 4, 5, 6, 8, 9</td>
</tr>
</tbody>
</table>

```java
boolean changed = true;
do {
    changed = false;
    for (int j = 0; j < list.length - 1; j++)
        if (list[j] > list[j + 1]) {
            swap list[j] with list[j + 1];
            changed = true;
        }
} while (changed);
```

Reading/Homework

° Chapter 6 of the textbook: 6.1 – 6.5, 6.7 - 6.11
° Preview Chapter 7
° Do review questions: 6.4 – 6.6, 6.10, 6.20, 6.21, 6.27