Lecture 21
Selection sort

• Suppose that you want to sort a list in ascending order. Selection sort finds the smallest number in the list and swaps it with the first element. It then finds the smallest number remaining and swaps it with the second element, and so on, until only a single number remains. Figure 6.12 shows how to sort the list \{2, 9, 5, 4, 8, 1, 6\} using selection sort.
Select 1 (the smallest) and swap it with 2 (the first) in the list.

Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.

Select 4 (the smallest) and swap it with 5 (the first) in the remaining list.

Select 6 (the smallest) and swap it with 8 (the first) in the remaining list.

Select 8 (the smallest) and swap it with 9 (the first) in the remaining list.

Since there is only one element remaining in the list, the sort is completed.
public class SelectionSort {
    /** The method for sorting the numbers */
    public static void selectionSort(double[] list) {
        for (int i = 0; i < list.length - 1; i++) {
            // Find the minimum in the list[i..list.length-1]
            double currentMin = list[i];
            int currentMinIndex = i;

            for (int j = i + 1; j < list.length; j++) {
                if (currentMin > list[j]) {
                    currentMin = list[j];
                    currentMinIndex = j;
                }
            }

            // Swap list[i] with list[currentMinIndex] if necessary
            if (currentMinIndex != i) {
                list[currentMinIndex] = list[i];
                list[i] = currentMin;
            }
        }
    }
}
Insertion Sort

• Suppose that you want to sort a list in ascending order. The insertion-sort algorithm sorts a list of values by repeatedly inserting a new element into a sorted sublist until the whole list is sorted. Figure 6.13 shows how to sort the list \{2, 9, 5, 4, 8, 1, 6\} using insertion sort.
Step 1: Initially, the sorted sublist contains the first element in the list. Insert 9 into the sublist.

\[
\begin{array}{cccccccc}
2 & 9 & 5 & 4 & 8 & 1 & 6
\end{array}
\]

Step 2: The sorted sublist is \{2, 9\}. Insert 5 into the sublist.

\[
\begin{array}{cccccccc}
2 & 9 & 5 & 4 & 8 & 1 & 6
\end{array}
\]

Step 3: The sorted sublist is \{2, 5, 9\}. Insert 4 into the sublist.

\[
\begin{array}{cccccccc}
2 & 5 & 9 & 4 & 8 & 1 & 6
\end{array}
\]

Step 4: The sorted sublist is \{2, 4, 5, 9\}. Insert 8 into the sublist.

\[
\begin{array}{cccccccc}
2 & 4 & 5 & 9 & 8 & 1 & 6
\end{array}
\]

Step 5: The sorted sublist is \{2, 4, 5, 8, 9\}. Insert 1 into the sublist.

\[
\begin{array}{cccccccc}
2 & 4 & 5 & 8 & 9 & 1 & 6
\end{array}
\]

Step 6: The sorted sublist is \{1, 2, 4, 5, 8, 9\}. Insert 6 into the sublist.

\[
\begin{array}{cccccccc}
1 & 2 & 4 & 5 & 8 & 9 & 6
\end{array}
\]

Step 7: The entire list is now sorted.

\[
\begin{array}{cccccccc}
1 & 2 & 4 & 5 & 6 & 8 & 9
\end{array}
\]

**Figure 6.13** Insertion sort repeatedly inserts a new element into a sorted sublist.

**Figure 6.14** A new element is inserted into a sorted sublist.
public class InsertionSort {
    /** The method for sorting the numbers */
    public static void insertionSort(double[] list) {
        for (int i = 1; i < list.length; i++) {
            /** Insert list[i] into a sorted sublist list[0..i-1] so that list[0..i] is sorted. */
            double currentElement = list[i];
            int k;
            for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
                list[k + 1] = list[k];
            }
            // Insert the current element into list[k + 1]
            list[k + 1] = currentElement;
        }
    }
}
The Arrays Class

double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers); // Sort the whole array

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars, 1, 3); // Sort part of the array

int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("(1) Index is " +
    java.util.Arrays.binarySearch(list, 11));

You can use the equals method to check whether two arrays are equal. Two arrays are equal if they have the same contents. In the following code, list1 and list2 are equal, but list2 and list3 are not.

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)); // true
System.out.println(java.util.Arrays.equals(list2, list3)); // false

You can also use the toString method to return a string that represents all elements in the array. This is a quick and simple way to display all elements in the array. For example, the following code

    int[] list = {2, 4, 7, 10};
    System.out.println(Arrays.toString(list));

displays [2, 4, 7, 10].
Two-Dimensional Array Basics

The syntax for declaring a two-dimensional array is:

```
elementType[][] arrayRefVar;
```

or

```
elementType arrayRefVar[][];  // Allowed, but not preferred
```

As an example, here is how you would declare a two-dimensional array variable `matrix` of `int` values:

```
int[][] matrix;
```

or

```
int matrix[][];  // This style is allowed, but not preferred
```

You can create a two-dimensional array of 5-by-5 `int` values and assign it to `matrix` using this syntax:

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

matrix[2][1] = 7;

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

Equivalent

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;
Obtaining the Lengths of Two-Dimensional Arrays

• A two-dimensional array is actually an array in which each element is a one-dimensional array. The length of an array \( x \) is the number of elements in the array, which can be obtained using \( x.length \). \( x[0] \), \( x[1] \), \ldots, and \( x[x.length-1] \) are arrays. Their lengths can be obtained using \( x[0].length \), \( x[1].length \), \ldots, and \( x[x.length-1].length \).
Ragged Arrays

Each row in a two-dimensional array is itself an array. Thus, the rows can have different lengths. An array of this kind is known as a *ragged array*. Here is an example of creating a ragged array:

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

As you can see, `triangleArray[0].length` is 5, `triangleArray[1].length` is 4, `triangleArray[2].length` is 3, `triangleArray[3].length` is 2, and `triangleArray[4].length` is 1.

If you don’t know the values in a ragged array in advance, but do know the sizes—say, the same as before—you can create a ragged array using the following syntax:

```java
int[][] triangleArray = new int[5][];
triangleArray[0] = new int[5];
triangleArray[1] = new int[4];
triangleArray[2] = new int[3];
triangleArray[3] = new int[2];
triangleArray[4] = new int[1];
```

You can now assign values to the array. For example,

```java
triangleArray[0][3] = 50;
triangleArray[4][0] = 45;
```

**Note**
The syntax `new int[5][]` for creating an array requires the first index to be specified. The syntax `new int[] []` would be wrong.
What is the output of the following code?

```java
int[][] array = new int[5][6];
int[] x = {1, 2};
array[0] = x;
System.out.println("array[0][1] is " + array[0][1]);
```

Which of the following statements are valid?

```java
int[][] r = new int[2];
int[] x = new int[];
int[][] y = new int[3][];
int[][] z = {{1, 2}};
int[][] m = {{1, 2}, {2, 3}};
int[][] n = {{1, 2}, {2, 3}, };
Suppose an array `matrix` is created as follows:

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

The following are some examples of processing two-dimensional arrays.

1. **Initializing arrays with input values.** The following loop initializes the array with user input values:

   ```java
   java.util.Scanner input = new Scanner(System.in);
   System.out.println("Enter " + matrix.length + " rows and " +
   matrix[0].length + " columns:");
   for (int row = 0; row < matrix.length; row++) {
       for (int column = 0; column < matrix[row].length; column++) {
           matrix[row][column] = input.nextInt();
       }
   }
   ```

2. **Initializing arrays with random values.** The following loop initializes the array with random values between 0 and 99:

   ```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);
       }
   }
   ```

3. **Printing arrays.** To print a two-dimensional array, you have to print each element in the array using a loop like the following:

   ```java
   for (int row = 0; row < matrix.length; row++) {
       for (int column = 0; column < matrix[row].length; column++) {
           System.out.print(matrix[row][column] + " ");
       }
   
   System.out.println();
   ```
4. Summing all elements. Use a variable named `total` to store the sum. Initially `total` is 0. Add each element in the array to `total` using a loop like this:

```java
int total = 0;
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        total += matrix[row][column];
    }
}
```

5. Summing elements by column. For each column, use a variable named `total` to store its sum. Add each element in the column to `total` using a loop like this:

```java
for (int column = 0; column < matrix[0].length; column++) {
    int total = 0;
    for (int row = 0; row < matrix.length; row++)
        total += matrix[row][column];
    System.out.println("Sum for column " + column + " is "+ total);
}
```

6. Which row has the largest sum? Use variables `maxRow` and `indexOfMaxRow` to track the largest sum and index of the row. For each row, compute its sum and update `maxRow` and `indexOfMaxRow` if the new sum is greater.

```java
int maxRow = 0;
int indexOfMaxRow = 0;

// Get sum of the first row in maxRow
for (int column = 0; column < matrix[0].length; column++) {
    maxRow += matrix[0][column];
}

for (int row = 1; row < matrix.length; row++) {
    int totalOfThisRow = 0;
    for (int column = 0; column < matrix[row].length; column++)
        totalOfThisRow += matrix[row][column];
    if (totalOfThisRow > maxRow) {
        maxRow = totalOfThisRow;
        indexOfMaxRow = row;
    }
}

System.out.println("Row " + indexOfMaxRow + " has the maximum sum of " + maxRow);
```
7. Random shuffling. Shuffling the elements in a one-dimensional array was introduced in Section 6.2.6. How do you shuffle all the elements in a two-dimensional array? To accomplish this, for each element `matrix[i][j]`, randomly generate indices `i1` and `j1` and swap `matrix[i][j]` with `matrix[i1][j1]`, as follows:

```java
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        int i1 = (int)(Math.random() * matrix.length);
        int j1 = (int)(Math.random() * matrix[i].length);

        // Swap matrix[i][j] with matrix[i1][j1]
        int temp = matrix[i][j];
        matrix[i][j] = matrix[i1][j1];
        matrix[i1][j1] = temp;
    }
}
```
7.5 Show the printout of the following code:

```java
int[][] array = {{1, 2}, {3, 4}, {5, 6}};
for (int i = array.length - 1; i >= 0; i--) {
    for (int j = array[i].length - 1; j >= 0; j--)
        System.out.print(array[i][j] + " ");
    System.out.println();
}
```

7.6 Show the printout of the following code:

```java
int[][] array = {{1, 2}, {3, 4}, {5, 6}};
int sum = 0;
for (int i = 0; i < array.length; i++)
    sum += array[i][0];
System.out.println(sum);
```