Chapter 7: Arrays & More
Lab Exercises

Topics	Lab Exercises
One-Dimensional Arrays	**Tracking Sales ***
	Reversing an Array *
	Adding To and Removing From an Integer List *
Arrays of Objects	**A Shopping Cart***

Om du gör stor projekt ”CardGame”, då skall även nedanstående uppgifter göras.

Om du gör liten projekt ”Shopping at the Green Store” är uppgifterna nedan friviliga.

Variable Length Parameter Lists	Exploring Variable Length Parameter Lists
Two-Dimensional Arrays	Magic Squares
Mouse Events	Drawing Circles with Mouse Clicks
	Moving Circles with the Mouse

Tracking Sales

File *Sales.java* contains a Java program that prompts for and reads in the sales for each of 5 salespeople in a company. It then prints out the id and amount of sales for each salesperson and the total sales. Study the code, then compile and run the program to see how it works. Now modify the program as follows:

1. Compute and print the average sale. (You can compute this directly from the total; no loop is necessary.)
2. Find and print the maximum sale. Print both the id of the salesperson with the max sale and the amount of the sale, e.g., "Salesperson 3 had the highest sale with $4500." Note that you don't need another loop for this; you can do it in the same loop where the values are read and the sum is computed.
3. Do the same for the minimum sale.
4. After the list, sum, average, max and min have been printed, ask the user to enter a value. Then print the id of each salesperson who exceeded that amount, and the amount of their sales. Also print the total number of salespeople whose sales exceeded the value entered.
5. The salespeople are objecting to having an id of 0—no one wants that designation. Modify your program so that the ids run from 1–5 instead of 0–4. **Do not modify the array**—just make the information for salesperson 1 reside in array location 0, and so on.
6. Instead of always reading in 5 sales amounts, at the beginning ask the user for the number of sales people and then create an array that is just the right size. The program can then proceed as before.

```
// ****************************************************************
// Sales.java
//
// Reads in and stores sales for each of 5 salespeople.  Displays
// sales entered by salesperson id and total sales for all salespeople.
//
// ****************************************************************
import java.util.Scanner;

public class Sales
{
    public static void main(String[] args)
    {
           final int SALESPEOPLE = 5;
           int[] sales = new int[SALESPEOPLE];
           int sum;

           Scanner scan = new Scanner(System.in);

           for (int i=0; i<sales.length; i++)
               {
               System.out.print("Enter sales for salesperson " + i + ": ");
               sales[i] = scan.nextInt();
               }

           System.out.println("\nSalesperson   Sales");
           System.out.println("--------------------");
           sum = 0;
           for (int i=0; i<sales.length; i++)
               {
               System.out.println("     " + i + "         " + sales[i]);
                      sum += sales[i];
               }

           System.out.println("\nTotal sales: " + sum);
    }
}
```

Reversing an Array*

Write a program that prompts the user for an integer, then asks the user to enter that many values. Store these values in an array and print the array. Then reverse the array elements so that the first element becomes the last element, the second element becomes the second to last element, and so on, with the old last element now first. Do not just reverse the order in which they are printed; actually change the way they are stored in the array.

1.You can solve this problem in two ways. By creating a temporary array and copy all data from the original array to the temporaty . Than change the original array to the temporary array referens. Print the original array.

2. Solve the same problem but **do not create a second array**; just rearrange the elements within the array you have. (Hint: Swap elements that need to change places.) When the elements have been reversed, print the array again.

Adding To and Removing From an Integer List

File *IntegerList.java* contains a Java class representing a list of integers. The following public methods are provided:

- IntegerList(int size)—creates a new list of *size* elements. Elements are initialized to 0.
- void randomize()—fills the list with random integers between 1 and 100, inclusive.
- void print()—prints the array elements and indices

File *IntegerListTest.java* contains a Java program that provides menu-driven testing for the IntegerList class. Copy both files to your directory, and compile and run IntegerListTest to see how it works.

It is often necessary to add items to or remove items from a list. When the list is stored in an array, one way to do this is to create a new array of the appropriate size each time the number of elements changes, and copy the values over from the old array. However, this is rather inefficient. A more common strategy is to choose an initial size for the array and add elements until it is full, then double its size and continue adding elements until it is full, and so on. (It is also possible to decrease the size of the array if it falls under, say, half full, but we won't do that in this exercise.) The CDCollection class in Listing 7.8 of the text uses this strategy—it keeps track of the current size of the array and the number of elements already stored in it, and method *addCD* calls *increaseSize* if the array is full. Study that example.

1. Add this capability to the IntegerList class. You will need to add an *increaseSize* method plus instance variables to hold the current number of integers in the list and the current size of the array. Since you do not have any way to add elements to the list, you won't need to call *increaseSize* yet.

2. Add a method *void addElement(int newVal)* to the IntegerList class that adds an element to the list. At the beginning of *addElement*, check to see if the array is full. If so, call *increaseSize* before you do anything else.

   Add an option to the menu in IntegerListTest to test your new method.

3. Add a method *void removeFirst(int newVal)* to the IntegerList class that removes the first occurrence of a value from the list. If the value does not appear in the list, it should do nothing (but it's not an error). Removing an item should not change the size of the array, but note that the array values do need to remain contiguous, so when you remove a value you will have to shift everything after it down to fill up its space. Also remember to decrement the variable that keeps track of the number of elements.

   Add an option to the menu in IntegerListTest to test your new method.

4. Add a method *removeAll(int newVal)* to the IntegerList class that removes all occurrences of a value from the list. If the value does not appear in the list, it should do nothing (but it's not an error).

   Add an option to the menu in IntegerListTest to test your new method.

```
// ****************************************************************
// IntegerList.java
//
// Define an IntegerList class with methods to create & fill
// a list of integers.
//
// ****************************************************************

public class IntegerList
{
    int[] list; //values in the list

    //-------------------------------------------------------
    //create a list of the given size
    //-------------------------------------------------------
```

```
    public IntegerList(int size)
    {
           list = new int[size];
    }


    //-----------------------------------------------------
    //fill array with integers between 1 and 100, inclusive
    //-----------------------------------------------------
    public void randomize()
    {
           for (int i=0; i<list.length; i++)
               list[i] = (int)(Math.random() * 100) + 1;
    }

    //-----------------------------------------------------
    //print array elements with indices
    //-----------------------------------------------------
    public void print()
    {
           for (int i=0; i<list.length; i++)
               System.out.println(i + ":\t" + list[i]);
    }
}



// ****************************************************************
// IntegerListTest.java
//
// Provide a menu-driven tester for the IntegerList class.
//
// ****************************************************************
import java.util.Scanner;

public class IntegerListTest
{
    static IntegerList list = new IntegerList(10);
    static Scanner scan = new Scanner(System.in);

    //-----------------------------------------------------
    // Create a list, then repeatedly print the menu and do what the
    // user asks until they quit
    //-----------------------------------------------------
    public static void main(String[] args)
    {
           printMenu();
           int choice = scan.nextInt();
           while (choice != 0)
               {
                      dispatch(choice);
                      printMenu();
                      choice = scan.nextInt();
               }
    }

    //--------------------------------------
    // Do what the menu item calls for
    //--------------------------------------
    public static void dispatch(int choice)
    {
```

```
            int loc;
            switch(choice)
                {
                case 0:
                       System.out.println("Bye!");
                       break;
                case 1:
                       System.out.println("How big should the list be?");
                       int size = scan.nextInt();
                       list = new IntegerList(size);
                       list.randomize();
                       break;
                case 2:
                       list.print();
                       break;
                default:
                       System.out.println("Sorry, invalid choice");
                }
    }

    //----------------------------
    // Print the user's choices
    //----------------------------
    public static void printMenu()
    {
            System.out.println("\n   Menu   ");
            System.out.println("   ====");
            System.out.println("0: Quit");
System.out.println("1: Create a new list (** do this first!! **)");
            System.out.println("2: Print the list");
            System.out.print("\nEnter your choice: ");
    }
}
```

A Shopping Cart*

In this exercise you will complete a class that implements a shopping cart as an array of items. The file *Item.java* contains the definition of a class named *Item* that models an item one would purchase. An item has a name, price, and quantity (the quantity purchased). The file *ShoppingCart.java* implements the shopping cart as an array of Item objects.

1. Complete the *ShoppingCart* class by doing the following:
   a. Declare an instance variable *cart* to be an array of Items and instantiate *cart* in the constructor to be an array holding *capacity* Items.
   b. Fill in the code for the *increaseSize* method. Your code should be similar to that in Listing 7.8 of the text but instead of doubling the size just increase it by 3 elements.
   c. Fill in the code for the *addToCart* method. This method should add the item to the cart and update the *totalPrice* instance variable (note this variable takes into account the quantity).
   d. Compile your class.

2. Write a program that simulates shopping. The program should have a loop that continues as long as the user wants to shop. Each time through the loop read in the name, price, and quantity of the item the user wants to add to the cart. After adding an item to the cart, the cart contents should be printed. After the loop print a "Please pay ..." message with the total price of the items in the cart.

```
// *************************************************************
```

```
//   Item.java
//
//   Represents an item in a shopping cart.
// ************************************************************

import java.text.NumberFormat;

public class Item
{
    private String name;
    private double price;
    private int quantity;

    // -------------------------------------------------------
    //  Create a new item with the given attributes.
    // -------------------------------------------------------
    public Item (String itemName, double itemPrice, int numPurchased)
    {
           name = itemName;
           price = itemPrice;
           quantity = numPurchased;
    }

    // -------------------------------------------------------
    //   Return a string with the information about the item
    // -------------------------------------------------------
    public String toString ()
    {
           NumberFormat fmt = NumberFormat.getCurrencyInstance();

           return (name + "\t" + fmt.format(price) + "\t" + quantity + "\t"
                      + fmt.format(price*quantity));
    }

    // -----------------------------------------------
    //   Returns the unit price of the item
    // -----------------------------------------------
    public double getPrice()
    {
           return price;
    }

    // -----------------------------------------------
    //   Returns the name of the item
    // -----------------------------------------------
    public String getName()
    {
           return name;
    }

    // -----------------------------------------------
    //   Returns the quantity of the item
    // -----------------------------------------------
    public int getQuantity()
    {
           return quantity;
    }
}
```

```
// ********************************************************************
//   ShoppingCart.java
//
//   Represents a shopping cart as an array of items
// ********************************************************************

import java.text.NumberFormat;

public class ShoppingCart
{
    private int itemCount;      // total number of items in the cart
    private double totalPrice;  // total price of items in the cart
    private int capacity;       // current cart capacity

    // -----------------------------------------------------------
    //  Creates an empty shopping cart with a capacity of 5 items.
    // -----------------------------------------------------------
    public ShoppingCart()
    {
           capacity = 5;
           itemCount = 0;
           totalPrice = 0.0;
    }

    // -----------------------------------------------------------
    //  Adds an item to the shopping cart.
    // -----------------------------------------------------------
    public void addToCart(String itemName, double price, int quantity)
    {
    }

    // -----------------------------------------------------------
    //  Returns the contents of the cart together with
    //  summary information.
    // -----------------------------------------------------------
    public String toString()
    {
           NumberFormat fmt = NumberFormat.getCurrencyInstance();

           String contents = "\nShopping Cart\n";
           contents += "\nItem\t\tUnit Price\tQuantity\tTotal\n";

           for (int i = 0; i < itemCount; i++)
               contents += cart[i].toString() + "\n";

           contents += "\nTotal Price: " + fmt.format(totalPrice);
           contents += "\n";

           return contents;
    }

    // -----------------------------------------------------------
    //  Increases the capacity of the shopping cart by 3
    // -----------------------------------------------------------
    private void increaseSize()
    {
    }
}
```

Exploring Variable Length Parameter Lists

The file *Parameters.java* contains a program to test the variable length method *average* from Section 7.5 of the text. Note that *average* must be a static method since it is called from the static method *main*.

1. Compile and run the program. You must use the -source 1.5 option in your compile command.
2. Add a call to find the average of a single integer, say 13. Print the result of the call.
3. Add a call with an empty parameter list and print the result. Is the behavior what you expected?
4. Add an interactive part to the program. Ask the user to enter a sequence of at most 20 nonnegative integers. Your program should have a loop that reads the integers into an array and stops when a negative is entered (the negative number should not be stored). Invoke the average method to find the average of the integers in the array (send the array as the parameter). Does this work?
5. Add a method *minimum* that takes a variable number of integer parameters and returns the minimum of the parameters. Invoke your method on each of the parameter lists used for the average function.

```
//********************************************************
//  Parameters.java
//
//  Illustrates the concept of a variable parameter list.
//********************************************************

import java.util.Scanner;

public class Parameters
{
    //-----------------------------------------------
    //  Calls the average and minimum methods with
    //  different numbers of parameters.
    //-----------------------------------------------
    public static void main(String[] args)
    {
        double mean1, mean2;

        mean1 = average(42, 69, 37);
        mean2 = average(35, 43, 93, 23, 40, 21, 75);

        System.out.println ("mean1 = " + mean1);
        System.out.println ("mean2 = " + mean2);
    }

    //-------------------------------------------
    //  Returns the average of its parameters.
    //-------------------------------------------
    public static 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;
    }
}
```

Magic Squares

One interesting application of two-dimensional arrays is *magic squares*. A magic square is a square matrix in which the sum of every row, every column, and both diagonals is the same. Magic squares have been studied for many years, and there are some particularly famous magic squares. In this exercise you will write code to determine whether a square is magic.

File *Square.java* contains the shell for a class that represents a square matrix. It contains headers for a constructor that gives the size of the square and methods to read values into the square, print the square, find the sum of a given row, find the sum of a given column, find the sum of the main (or other) diagonal, and determine whether the square is magic. The read method is given for you; you will need to write the others. Note that the read method takes a Scanner object as a parameter.

File *SquareTest.java* contains the shell for a program that reads input for squares from a file named *magicData* and tells whether each is a magic square. Following the comments, fill in the remaining code. Note that the main method reads the size of a square, then after constructing the square of that size, it calls the *readSquare* method to read the square in. The readSquare method must be sent the Scanner object as a parameter.

You should find that the first, second, and third squares in the input are magic, and that the rest (fourth through seventh) are not. Note that the -1 at the bottom tells the test program to stop reading.

```
// ****************************************************************
// Square.java
//
// Define a Square class with methods to create and read in
// info for a square matrix and to compute the sum of a row,
// a col, either diagonal, and whether it is magic.
//
// ****************************************************************

import java.util.Scanner;

public class Square
{
    int[][] square;

    //---------------------------------------
    //create new square of given size
    //---------------------------------------
    public Square(int size)
    {

    }

    //---------------------------------------
    //return the sum of the values in the given row
    //---------------------------------------
    public int sumRow(int row)
    {

    }

    //---------------------------------------
    //return the sum of the values in the given column
    //---------------------------------------
    public int sumCol(int col)
    {

    }
```

```
    //---------------------------------------
    //return the sum of the values in the main diagonal
    //---------------------------------------
    public int sumMainDiag()
    {

    }

    //---------------------------------------
    //return the sum of the values in the other ("reverse") diagonal
    //---------------------------------------
    public int sumOtherDiag()
    {

    }

    //---------------------------------------
    //return true if the square is magic (all rows, cols, and diags have
    //same sum), false otherwise
    //---------------------------------------
    public boolean magic()
    {

    }

    //---------------------------------------
    //read info into the square from the standard input.
    //---------------------------------------
    public void readSquare(Scanner scan)
    {
      for (int row = 0; row < square.length; row++)
             for (int col = 0; col < square.length; col ++)
               square[row][col] = scan.nextInt();
    }

    //---------------------------------------
    //print the contents of the square, neatly formatted
    //---------------------------------------
    public void printSquare()
    {

    }

}
```

```
// ********************************************************************
// SquareTest.java
//
// Uses the Square class to read in square data and tell if
// each square is magic.
//
// ********************************************************************

import java.util.Scanner;

public class SquareTest
{
    public static void main(String[] args) throws IOException
    {
      Scanner scan = new Scanner(new File("magicData"));

           int count = 1;                 //count which square we're on
           int size = scan.nextInt();     //size of next square

           //Expecting -1 at bottom of input file
           while (size != -1)
               {

                      //create a new Square of the given size

                      //call its read method to read the values of the
square

System.out.println("\n******** Square " + count + " ********");

//print the square

                      //print the sums of its rows

                      //print the sums of its columns

                      //print the sum of the main diagonal

                      //print the sum of the other diagonal

                      //determine and print whether it is a magic square

                      //get size of next square
                      size = scan.nextInt();

               }

   }
}
```

magicData

```
3
8  1  6
3  5  7
4  9  2

7
30  39  48   1  10  19  28
38  47   7   9  18  27  29
46   6   8  17  26  35  37
5   14  16  25  34  36  45
13  15  24  33  42  44   4
21  23  32  41  43   3  12
22  31  40  49   2  11  20

4
48   9   6   39
27  18  21   36
15  30  33   24
12  45  42    3

3
6  2  7
1  5  3
2  9  4

4
3  16   2  13
6   9   7  12
10  5  11   8
15  4  14   1

5
17  24  15   8   1
23   5  16  14   7
 4   6  22  13  20
10  12   3  21  19
11  18   9   2  25

7
30  39  48   1  10  28  19
38  47   7   9  18  29  27
46   6   8  17  26  37  35
5   14  16  25  34  45  36
13  15  24  33  42   4  44
21  23  32  41  43  12   3
22  31  40  49   2  20  11
-1
```

Drawing Circles with Mouse Clicks

File *Circles.java* sets up a panel that creates and draws a circle as defined in *Circle.java* of random size and color at each mouse click. Each circle replaces the one before it. The code to handle the mouse clicks and do the drawing is in *CirclePanel.java*. Save these files to your directory, compile them and run them and experiment with the GUI. Then modify these files as described below.

1. This program creates a new circle each time—you can tell because each circle is a different color and size. Write a method *void move(Point p)* for your Circle class that takes a Point and moves the circle so its center is at that point. Now modify your CirclesListener class (defined inside CirclePanel) so that instead of creating a new circle every time the user clicks, it moves the existing circle to the clickpoint if a circle already exists. If no circle exists, a new one should be created at the clickpoint. So now a circle of the same color and size should move around the screen.

2. Write a method *boolean isInside(Point p)* for your Circle class that takes a Point and tells whether it is inside the circle. A point is inside the circle if its distance from the center is less than the radius. (Recall that the distance between two points (x1,y1) and (x2,y2) is sqrt($(x2-x1)^2+(y2-y1)^2$.)

3. Now modify the *mousePressed* method of CirclesListener so that the GUI behaves as follows:
   - If there is no circle (i.e., it is null) and the user clicks anywhere, a new (random) circle should be drawn at the click point.
   - If there is a circle on the screen and the user clicks inside that circle, the circle should go away. (Hint: To make the circle go away, set it to null and repaint.)
   - If there is a circle on the screen and the user clicks somewhere else, the circle should move to that point (no change from before).

   So the logic for *mousePressed* should look like this:

```
if there is currently no circle
   create a new circle at the click point
else if the click is inside the circle
   make the circle go away
else
   move the circle to the click point

repaint
```

4. Add bodies for the *mouseEntered* and *mouseExited* methods so that when the mouse enters the panel the background turns white, and when it exits the background turns blue. Remember that you can set the background color with the *setBackground* method.

```
//********************************************************************
//  Circles.java
//
//  Demonstrates mouse events and drawing on a panel.
//  Derived from Dots.java in Lewis and Loftus
//********************************************************************

import javax.swing.JFrame;

public class Circles
{
   //-----------------------------------------------------------------
   //  Creates and displays the application frame.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      JFrame circlesFrame = new JFrame ("Circles");
      circlesFrame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      circlesFrame.getContentPane().add (new CirclePanel());

      circlesFrame.pack();
      circlesFrame.setVisible(true);
   }
}
```

```
// ****************************************************************
// Circle.java
//
// Define a Circle class with methods to create and draw
// a circle of random size, color, and location.
//
// ****************************************************************


import java.awt.*;
import java.util.Random;

public class Circle
{
    private int centerX, centerY;
    private int radius;
    private Color color;

    static Random generator = new Random();

    //-------------------------------------------------------
    // Creates a circle with center at point given, random radius and color
    //   -- radius 25..74
    //   -- color RGB value 0..16777215 (24-bit)
    //-------------------------------------------------------
    public Circle(Point point)
    {
           radius = Math.abs(generator.nextInt())%50 + 25;
           color = new Color(Math.abs(generator.nextInt())% 16777216);
           centerX = point.x;
           centerY = point.y;
    }

    //-------------------------------------------------------
    // Draws circle on the graphics object given
    //-------------------------------------------------------
    public void draw(Graphics page)
    {
           page.setColor(color);
           page.fillOval(centerX-radius,centerY-radius,radius*2,radius*2);
    }
}
```

```
//********************************************************************
//  CirclePanel.java
//
//  Represents the primary panel for the Circles program on which the
//  circles are drawn.  Derived from the Lewis and Loftus DotsPanel class.
//********************************************************************

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class CirclePanel extends JPanel
{
   private final int WIDTH = 600, HEIGHT = 400;
   private Circle circle;

   //-----------------------------------------------------------------
   //  Sets up this panel to listen for mouse events.
   //-----------------------------------------------------------------
   public CirclePanel()
   {
      addMouseListener (new CirclesListener());
      setPreferredSize (new Dimension(WIDTH, HEIGHT));
   }

   //-----------------------------------------------------------------
   //  Draws the current circle, if any.
   //-----------------------------------------------------------------
   public void paintComponent (Graphics page)
   {
      super.paintComponent(page);
      if (circle != null)
            circle.draw(page);
   }

   //*****************************************************************
   //  Represents the listener for mouse events.
   //*****************************************************************
   private class CirclesListener implements MouseListener
   {
      //--------------------------------------------------------------
      // Creates a new circle at the current location whenever the
      // mouse button is pressed and repaints.
      //--------------------------------------------------------------
      public void mousePressed (MouseEvent event)
      {
         circle = new Circle(event.getPoint());
         repaint();
      }

      //--------------------------------------------------------------
      //  Provide empty definitions for unused event methods.
      //--------------------------------------------------------------
      public void mouseClicked (MouseEvent event) {}
      public void mouseReleased (MouseEvent event) {}
      public void mouseEntered (MouseEvent event) {}
      public void mouseExited (MouseEvent event) {}
   }
}
```