Chapter 7: Arrays & More
Lab Exercises

Topics	Lab Exercises
One-Dimensional Arrays	**Tracking Sales**
	Working with arrays
	Cryptography
Arrays of Objects	**A Shopping Cart (Friviligt uppgift för IT –forensik)**
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);
        }
    }
```

Working with arrays

1. What will be printed by the following code segment?

```
int arr[] = { 24,0,19,21,6,-5,10,16};

for(int i=0;i<arr.lenght;i++)
  System.out.println(arr[i]);
```

2. What will be printed by the following code segment?

```
int arr[] = { 24,0,19,21,6,-5,10,16};
 for(int k =0;i<arr.lenght; k=k+2)
  System.out.println(arr[k]);
```

3. Compile and run the following class. Understand how it works and implement the c, d, and e functionalities in the text below.

```
import java.util.*;
public class Program1 {

            public static void main (String []arg)
            {
                        Random gen=new Random();

             // this create a empty array named arr
                        int [] arr = new int[10];


            //a)  this fills the array with value generated by the Random

                        for(int i=0;i <arr.length;i++)
                                    arr[i]=gen.nextInt(100);

            // b) this prints the array
                        for(int i=0;i <arr.length;i++)
                         System.out.println(arr[i]);


            // c) calculate the sum of all values in the array


            // d) calculate the average of all values in the array

            // e) search for the lucky number 7. If the array contains the
            // number 7, the text “Lucky array” should be printed
            // to the output.



            }
}
```

4. We can do it better! We should implement the functionalities in separate methods and class and call the methods in main or what ever you needed. The method fillArrayay() is ready, se my code below. Your work is to do the same for all other functionalities b, c, d, e.

```
import java.util.*;
public class MyArrayMethods{

            public static void fillArray( int[] array, int maxValue){

                        Random gen=new Random();
            for(int i=0;i <array.length;i++)
                        array[i]=gen.nextInt(maxValue);
            }




}
```

Prove that your methods work by calling these methods in the Programe2 class.

```
import java.util.*;
public class Program2 {

            public static void main (String []arg)
            {

             // this creates a empty array named arr
                        int [] arr = new int[10];

             // call the method fillArray

            MyArrayMethods.fillArray( arr, 50);


            // Here try your methods


              }
}
```

Cryptography

Cryptography has long been an important area of study within computer science.
For this program you will use a simple translation table to encode a message.
The table is two char arrays containing the alphabetic characters and the corresponding encrypted characters.

```
char [] characterArray =
{'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','r','s','t
','u','w','x','y','z',' '};

char [] encryptArray= {'c','o','m','p','u','t','e','r',' ',
's','c','i','e','n','c','e',' ', 'i','s',' ', 'f', 'u', 'n', '!', ' ', ' ',
' ' };
```

The user is expected to type a message string into the program. Then the programme will translate and print out the entered message and the correct encoded message. This translation will process as follows:

- The encoded message will contain the same number of character as the source
- Every alphabetic character will be replaced by the character from the corresponding translation table.
- Every alphabetic letter in the source string that have a corresponding empty character in the table remains unaltered.
- All other characters (nonalphabetic) remain unchanged in the decoding process.

```
import java.util.*;

    public class Chrypto
   {

       public static void main(String[] args)
      {
         Scanner scan=new Scanner ( System.in);
         String message=scan.nextLine();



         char [] characterArray =
{'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','r','s','t
','u','w','x','y','z',' '};

         char [] encryptArray= {'c','o','m','p','u','t','e','r',' ',
's','c','i','e','n','c','e',' ', 'i','s',' ', 'f', 'u', 'n', '!', ' ', ' ',
' ' };

         String encodedMessage="";


// Tip! for each character from the input message, find his position in the
characterArray.
```

```
At the same position in the encryptArray, you will find your corresponding
character. Add it to your encoded message string.


System.out.print(encodedMessage);


  }
}
```

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()
    {
    }
}
```

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) {}
   }
}
```