Chapter 7: Arrays
Lab Exercises

Topics	**Lab Exercises**
One-Dimensional Arrays	Tracking Sales
	Grading Quizzes
	Reversing an Array
	Adding To and Removing From an Integer List
Arrays of Objects	A Shopping Cart
Command Line Arguments	Averaging Numbers
Variable Length Parameter Lists	Exploring Variable Length Parameter Lists
Two-Dimensional Arrays	Magic Squares
ArrayList Class	A Shopping Cart Using the ArrayList Class
Polygons & Polylines	A Polygon Person
Arrays & GUIs	An Array of Radio Buttons
Mouse Events	Drawing Circles with Mouse Clicks
	Moving Circles with the Mouse
Key Events	Moving a Stick Figure

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);
    }
}
```

Grading Quizzes

Write a program that grades arithmetic quizzes as follows:

1. Ask the user how many questions are in the quiz.
2. Ask the user to enter the key (that is, the correct answers). There should be one answer for each question in the quiz, and each answer should be an integer. They can be entered on a single line, e.g., 34 7 13 100 81 3 9 10 321 12 might be the key for a 10-question quiz. You will need to store the key in an array.
3. Ask the user to enter the answers for the quiz to be graded. As for the key, these can be entered on a single line. Again there needs to be one for each question. Note that these answers do not need to be stored; each answer can simply be compared to the key as it is entered.
4. When the user has entered all of the answers to be graded, print the number correct and the percent correct.

When this works, add a loop so that the user can grade any number of quizzes with a single key. After the results have been printed for each quiz, ask "Grade another quiz? (y/n)."

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. 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)
```

```java
    {
      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)
    {
```

```java
        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.

```java
// ************************************************************
//   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
    // -------------------------------------------------
```

```java
    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.
```

```java
      // ----------------------------------------------------------
      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()
      {
      }
}
```

Averaging Numbers

As discussed in Section 7.4 of the text book, when you run a Java program called Foo, anything typed on the command line after "java Foo" is passed to the main method in the args parameter as an array of strings.

1. Write a program Average.java that just prints the strings that it is given at the command line, one per line. If nothing is given at the command line, print "No arguments".

2. Modify your program so that it assumes the arguments given at the command line are integers. If there are no arguments, print a message. If there is at least one argument, compute and print the average of the arguments. Note that you will need to use the parseInt method of the Integer class to extract integer values from the strings that are passed in. If any non-integer values are passed in, your program will produce an error, which is unavoidable at this point.

3. Test your program thoroughly using different numbers of command line arguments.

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.

```java
//*******************************************************
//  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();

          }

    }
}
```

```
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
```

A Shopping Cart Using the ArrayList Class

In this exercise you will implement a shopping cart using the ArrayList class. The file *Item.java* contains the definition of a class named *Item* that models an item one would purchase (this class was used in an earlier lab). An item has a name, price, and quantity (the quantity purchased). The file *Shop.java* is an incomplete program that models shopping.

1. Complete Shop.java as follows:
   a. Declare and instantiate a variable *cart* to be an empty ArrayList.
   b. Fill in the statements in the loop to add an item to the cart and to print the cart contents (using the default *toString* in the ArrayList class). Comments in the code indicate where these statements go.
   c. Compile your program and run it.

2. You should have observed two problems with using the default printing for the cart object: the output doesn't look very good and the total price of the goods in the cart is not computed or printed. Modify the program to correct these problems by replacing the print statement with a loop that does the following:
   a. gets each item from the cart and prints the item
   b. computes the total price of the items in the cart (you need to use the *getPrice* and *getQuantity* methods of the Item class). The total price should be printed after the loop.

3. Compile and run your program.

```
// ****************************************************************
//   Shop.java
//
//   Uses the Item class to create items and add them to a shopping
//   cart stored in an ArrayList.
// ****************************************************************

import java.util.ArrayList;
import java.util.Scanner;

public class Shop
{
    public static void main (String[] args)
    {
      ArrayList<Item> cart = new ArrayList();

      Item item;
      String itemName;
      double itemPrice;
      int quantity;

      Scanner scan = new Scanner(System.in);

      String keepShopping = "y";

      do
          {
            System.out.print ("Enter the name of the item: ");
            itemName = scan.nextLine();

            System.out.print ("Enter the unit price: ");
            itemPrice = scan.nextDouble();

            System.out.print ("Enter the quantity: ");
            quantity = scan.nextInt();

            // *** create a new item and add it to the cart
```

```java
            // *** print the contents of the cart object using println


            System.out.print ("Continue shopping (y/n)? ");
            keepShopping = scan.nextLine();
        }
    while (keepShopping.equals("y"));

    }
}
```

A Polygon Person

A polygon is a multisided closed figure; a polyline is a line with an arbitrary number of segments. Both polygons and polylines are defined by a set of points, and Java provides graphics methods for both that are based on arrays. Read section 7.8 in the text and study the Rocket example in Listing 7.16 & 7.17.

Files *DrawPerson.java* and *DrawPersonPanel.java* contain a program that draws a blue shirt. Copy the programs to your directory, compile DrawPerson.java, and run it to see what it does. Now modify it as follows:

1. Draw pants to go with the shirt (they should be a different color). You will need to declare pantsX and pantsY arrays like the shirtX and shirtY arrays and figure out what should go in them. Then make the paint method draw the pants as well as the shirt.
2. Draw a head. This can just be a circle (or oval), so you won't need to use the Polygon methods. Declare variables headX and headY to hold the position of the head (its upper lefthand corner), and use them when you draw the circle.
3. Draw hair on the head. This is probably best done with a polygon, so again you'll need two arrays to hold the points.
4. Draw a zigzag across the front of the shirt. Use a polyline.
5. Write a method *movePerson(int x, int y)* that moves the person by the given number of pixels in the x and y direction. This method should just go through the shirt, pants, hair and zigzag arrays and the head x and y coords and increment all of the coordinates by the x or y value as appropriate. (This isn't necessarily the cleanest way to do this, but it's very straightforward).
6. Now put a loop in your paintComponent method that draws the person three times, moving him (her?) 150 or so pixels each time (you decide how far).

```
// ***************************************************************
//    DrawPerson.java
//
//    An program that uses the Graphics draw methods to draw a person.
// ***************************************************************

import javax.swing.JPanel;

public class DrawPerson
{
    //-----------------------------------------------
    //  Creates the main frame for the draw program
    //-----------------------------------------------
    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Draw Person");
        frame.setDefaultCloseOperaton (JFrame.EXIT_ON_CLOSE);

        DrawPersonPanel panel = new DrawPersonPanel ();

        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}

// ***************************************************************
//    DrawPersonPanel.java
//
//    An program that uses the Graphics draw methods to draw a person.
// ***************************************************************

import javax.swing.JPanel;
import java.awt.*;
```

```java
public class DrawPersonPanel extends JPanel
{
    private final int WIDTH = 600;
    private final int HEIGHT = 400;

    private int[] shirtX = {60,0,20,60,50,130,120,160,180,120};
    private int[] shirtY = {100,150,180,160,250,250,160,180,150,100};

    //-------------------------------------
    //  Constructor: Set up the panel.
    //-------------------------------------
    public DrawPersonPanel()
    {
      setPreferredSize(new Dimension(WIDTH, HEIGHT));
    }

    //-------------------------------------
    //   Draw person
    //-------------------------------------
    public void paintComponent (Graphics page)
    {
      page.setColor(Color.blue);
      page.fillPolygon(shirtX, shirtY, shirtX.length);

    }
}
```

An Array of Radio Buttons

File *ColorOptions.java* contains a program that will display a set of radio buttons that let the user change the background color of the GUI. The file *ColorOptionsPanel.java* contains the skeleton of the panel for this program . Open the files and study the code that is already there. You will note that in ColorOptionsPanel.java there is an array *color* containing 5 colors already defined. Your task is to add an array of radio buttons so that a click of a radio button will cause the background of the panel to change to the corresponding color in the *color* array.

1. Define *colorButton* to be an array of NUM_COLORS objects of type JRadioButton.

2. Instantiate each *colorButton* with the appropriate color as the label (for example, the first button should be labeled "Yellow"). The first button (corresponding to yellow) should be on (*true*) initially.

3. Recall that radio buttons must be grouped and that the selection of a radio button produces an action event. Hence you must have a ButtonGroup object and an ActionListener. Note that the skeleton of an ActionListener named *ColorListener* is already provided. So, you need to:
    a. Instantiate a ButtonGroup object and a ColorListener object. Comments in the code indicate where to do this.
    b. Each radio button needs to be added to your ButtonGroup object, the background color needs to be set (use white), your ColorListener needs to be added, and the button needs to be added to the panel. All of these can be done using a single for loop. So, add a for loop that goes through the radio buttons adding each to your ButtonGroup object, setting the background of each to white, adding your ColorListener to each, and adding each to the panel.

4. Fill in the body of the actionPerformed method. This method needs to go through the buttons to determine which is selected and then set the background color accordingly. A simple for loop can do this. Use the *isSelected* method to determine if a button is selected (for example, if (colorButton[i].isSelected())....). Use the *color* array to set the background color.

5. Test your program!

```
// ********************************************************************
//   ColorOptions.java
//
//   Uses an array of radio buttons to change the background color.
// ********************************************************************

import javax.swing.*;

public class ColorOptions
{
    // -----------------------------------------------------------
    //  Creates and presents the frame for the color change panel.
    // -----------------------------------------------------------
    public static void main (String[] args)
      {
          JFrame colorFrame = new JFrame ("Color Options");
          colorFrame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

          ColorOptionsPanel panel = new ColorOptionsPanel();
          colorFrame.getContentPane().add (panel);

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

```java
// **************************************************************************
//   ColorOptionsPanel.java
//
//   Represents the user interface for the ColorOptions program that lets
//   the user change background color by selecting a radio button.
// **************************************************************************

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

public class ColorOptionsPanel
{
    private final int WIDTH = 350, HEIGHT = 100, FONT_SIZE = 20;
    private final int NUM_COLORS = 5;
    private Color [] color = new Color[NUM_COLORS];
    private JLabel heading;

    // --------------------------------------------------------------------
    //   Sets up a panel with a label at the top and a set of radio buttons
    //   that control the background color of the panel.
    // --------------------------------------------------------------------
    public ColorOptionsPanel ()
    {
      // Set up heading and colors
      heading = new JLabel ("Choose the background color!");
      heading.setFont (new Font ("Helvetica", Font.BOLD, FONT_SIZE));

      color[0] = Color.yellow;
      color[1] = Color.cyan;
      color[2] = Color.red;
      color[3] = Color.green;
      color[4] = Color.magenta;

      // Instantiate a ButtonGroup object and a ColorListener object

      // Set up the panel
      add (heading);
      setBackground (Color.yellow);
      setPreferredSize (new Dimension (WIDTH, HEIGHT));

      // Group the radio buttons, add a ColorListener to each,
      // set the background color of each and add each to the panel.
    }

    // **********************************************************
    //     Represents the listener for the radio buttons.
    // **********************************************************
    private class ColorListener implements ActionListener
    {
      // ----------------------------------------------------------
      //   Updates the background color of the panel based on
      //   which radio button is selected.
      // ----------------------------------------------------------
      public void actionPerformed (ActionEvent event)
      {

      }
    }
}
```

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);
   }
}
```

```java
// *****************************************************************
// 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) {}
   }
}
```

Moving Circles with the Mouse

File *Circles.java* sets up a GUI 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*. (The files are from the previous exercise, Drawing Circles.) Save these files to your directory, compile them and run them and experiment with the GUI. Then modify the code in CirclePanel.java so that a circle is drawn when the user presses a mouse, but the user can drag it around as long as the mouse button is depressed. If the mouse button is released and then pressed again, a new circle is created, which can then be dragged around. You will need to make the following changes:

1. 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. (You may have already done this in a previous exercise.)

2. In the CirclePanel constructor, create a CirclesListener object and make it listen for both mouse events and mouse motion events.

3. Make the CirclesListener class implement the MouseMotionListener interface in addition to the MouseListener interface. This requires two steps:
   Note in the header that CirclesListener implements MouseMotionListener.
   Add bodies for the two MouseMotionListener methods, *mouseDragged* and *mouseMoved*. In *mouseDragged*, simply move the circle to the point returned by the *getPoint* method of the MouseEvent and repaint. Provide an empty body for *mouseMoved*.

Moving a Stick Figure

The file *StickFigure.java* contains a class that represents a stick figure. Study the file and note that in addition to a method to draw the stick figure there are methods to manipulate it:

> The method *move* repositions ("moves") the figure up or down and over (left or right) on the panel by changing the instance variables *baseX* and *baseY*.
> The method *grow* changes the size of the figure by a given factor by modifying the instance variable *height* and others.
> The method *setLegPosition* sets the position of the legs (number of pixels from vertical).
> The method *setArmPosition* sets the position of the arms (number of pixels from horizontal).

The file *MoveStickMan.java* contains a program that draws a single stick figure that can be moved around and modified using the keyboard. The file *MovePanel.java* represents the panel on which the stick figure is displayed. It is similar to DirectionPanel.java in Listing 7.23 of the text. Note that the constructor adds a KeyListener and contains a call to the setFocusable method. There is a partially defined inner class named *MoveListener* that implements the KeyListener interface. Currently the MoveListener listens only for two arrow keys (left and right) and the g key. The arrow keys move the stick figure left and right on the panel (the number of pixels moved is in the constant *JUMP*) and when the user presses the letter g, the figure "grows" (increases in height by 50%). Compile, and run the program. Test out the arrow keys and the g key.

Now add code to MovePanel.java to have the program respond to the following additional key events:

> When the up and down arrow keys are pressed the figure should move JUMP pixels up or down, respectively, on the panel.
> When the s key is pressed the figure should shrink by 50%.
> When the letter u (for up) is pressed, the stick figure should move its arms up and legs out; to make this happen add a call to *setArmPosition* to set the arm position and a call to *setLegPosition* to set the leg position. For example, when the u is pressed set the arm position to 60 and the leg position to 40 as follows:

```
        stickMan.setArmPosition(60);

        stickMan.setLegPosition(40);
```

> When the letter m (for middle) is pressed, the stick figure should place its arms horizontally with legs not quite as far out. To do this the arm position needs to be 0 (0 pixels above horizontal); a value of 20 for the leg position is good.
> When the letter d (down) is pressed, the figure should move its arms down and its legs closer together. (Use -60 and 10 for the arm and leg positions, respectively).

Compile and run the program. Try out all the keys it implements.

```
// ******************************************************
//    StickFigure.java
//
//    Represents a graphical stick figure
// ******************************************************

import java.awt.*;

public class StickFigure
{
    private int baseX;        // center of the figure
    private int baseY;        // bottom of the feet
    private Color color;      // color of the figure
    private int height;       // height of the figure
    private int headW;        // width of the head
    private int legLength;    // length of the legs
    private int legPosition;  // # pixels the legs are up from vertical
    private int armLength;    // horizontal length of the arms
    private int armToFloor;   // distance from base to arms
```

```java
    private int armPosition;// # pixels arm is above/below horizontal

    // --------------------------------------------------------------
    // Construct a stick figure given its four attributes
    // --------------------------------------------------------------
    public StickFigure (int center, int bottom, Color shade, int size)
    {
      baseX = center;
      baseY = bottom;
      color = shade;
      height = size;

      // define body positions proportional to height
      headW = height / 5;
      legLength = height / 2;
      armToFloor = 2 * height / 3;
      armLength =  height / 3;

      // set initial position of arms and legs
      armPosition = -20;
      legPosition = 15;
    }

    // ---------------------------------------------
    // Draw the figure
    // ---------------------------------------------
    public void draw (Graphics page)
    {
      // compute y-coordinate of top of head
      int top = baseY - height;

      page.setColor (color);

      // draw the head
      page.drawOval(baseX-headW/2, top, headW, headW);

      // draw the trunk
      page.drawLine (baseX, top+headW, baseX, baseY - legLength);

      // draw the legs
      page.drawLine(baseX, baseY-legLength, baseX-legPosition, baseY);
      page.drawLine(baseX, baseY-legLength, baseX+legPosition, baseY);

      // draw the arms
      int startY = baseY - armToFloor;
      page.drawLine(baseX, startY, baseX-armLength, startY-armPosition);
      page.drawLine(baseX, startY, baseX+armLength, startY-armPosition);
    }

    // -----------------------------------------------------
    // Move the figure -- first parameter gives the
    // number of pixels over (to right if over is positive,
    // to the left if over is negative) and up or down
    // (down if the parameter down is positive, up if it is
    // negative)
    // -----------------------------------------------------
    public void move (int over, int down)
    {
      baseX += over;
      baseY += down;
```

```java
    }

    // -------------------------------------------------------
    // Increase the height by the given factor (if the
    // factor is > 1 the figure will "grow" else it will
    // shrink)
    // -------------------------------------------------------
    public void grow (double factor)
    {
      height = (int) (factor * height);

      // reset body parts proportional to new height
      headW = height / 5;
      legLength = height / 2;
      armToFloor = 2 * height / 3;
      armLength = height / 3;
    }

    // -------------------------------------------------
    // set the legPosition (dist. from vertical) to
    // new value
    // -------------------------------------------------
    public void setLegPosition (int newPosition)
    {
      legPosition = newPosition;
    }

    // -----------------------------------------
    // set the arm position to the new value
    // -----------------------------------------
    public void setArmPosition (int newPos)
    {
      armPosition = newPos;
    }
}


// ****************************************************************
//   MoveStickMan.java
//
//   Uses key events to move a stick figure around.
// ****************************************************************
import javax.swing.*;

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

      frame.getContentPane().add (new MovePanel());

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

```
// *********************************************************************
// FILE:  MovePanel.java
//
// The display panel for a key events program  -- arrow keys are used
// to move a stick figure around, the g key is used to make the figure
// grow by 50% (increase in height by 50%), the s key causes the
// figure to shrink (to half its size)
// *********************************************************************

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

public class MovePanel extends JPanel
{
    private final int WIDTH = 600;
    private final int HEIGHT = 400;

    private final int JUMP = 5;    // number of pixels moved each step

    // the following give the initial parameters for the figure
    private final int START_CENTER = WIDTH/2;
    private final int START_BOTTOM = HEIGHT - 40;
    private final int SIZE = HEIGHT / 2;

    private StickFigure stickMan;

    // ---------------------------------------
    //   Constructor:  Sets up the panel
    // ---------------------------------------
    public MovePanel ()
    {
      addKeyListener(new MoveListener());

      stickMan = new StickFigure (START_CENTER, START_BOTTOM,
                        Color.yellow, SIZE);

      setBackground (Color.black);
      setPreferredSize (new Dimension (WIDTH, HEIGHT));
      setFocusable(true);
    }

    // ---------------------------------------
    //    Draws the figure
    // ---------------------------------------
    public void paintComponent (Graphics page)
    {
      super.paintComponent (page);
      stickMan.draw (page);
    }


    // ********************************************************
    //  Represents a listener for keyboard activity.
    // ********************************************************
    private class MoveListener implements KeyListener
    {
      // ---------------------------------------------------
      // Handle a key-pressed event: arrow keys cause the
      // figure to move horizontally or vertically; the g
```

```
        // key causes the figure to "grow", the s key causes
        // the figure to shrink, the u key causes arms and
        // legs to go up, m puts them in the middle, and d
        // down.
        // ---------------------------------------------------
        public void keyPressed (KeyEvent event)
        {
            switch (event.getKeyCode())
              {
              case KeyEvent.VK_LEFT:
                  stickMan.move(-1*JUMP, 0);
                  break;
              case KeyEvent.VK_RIGHT:
                  stickMan.move(JUMP, 0);
                  break;
              case KeyEvent.VK_G:
                  stickMan.grow (1.5);
                  break;
              default:
              }

            repaint();
        }

    // ----------------------------------------------
    // Define empty bodies for key event methods
    // not used
    // ----------------------------------------------
    public void keyTyped (KeyEvent event) {}
    public void keyReleased (KeyEvent event) {}
  }
}
```