Chapter 9: Polymorphism
Lab Exercises

Topics	Lab Exercises
Polymorphism via Inheritance	Another Type of Employee Painting Shapes
Sorting & Searching	Polymorphic Sorting Searching and Sorting An Integer List
Comparing Searches	Timing Searching and Sorting Algorithms
Color Choosers	Coloring a Moveable Circle
Sliders	Speed Control

Another Type of Employee

The files *Firm.java, Staff.java, StaffMember.java, Volunteer.java, Employee.java, Executive.java,* and *Hourly.java* are from Listings 9.1 – 9.7 in the text. The program illustrates inheritance and polymorphism. In this exercise you will add one more employee type to the class hierarchy (see Figure 9.1 in the text). The employee will be one that is an hourly employee but also earns a commission on sales. Hence the class, which we'll name *Commission,* will be derived from the *Hourly* class.

Write a class named *Commission* with the following features:

- It extends the *Hourly* class.
- It has two instance variables (in addition to those inherited): one is the total sales the employee has made (type double) and the second is the commission rate for the employee (the commission rate will be type double and will represent the percent (in decimal form) commission the employee earns on sales (so .2 would mean the employee earns 20% commission on sales)).
- The constructor takes 6 parameters: the first 5 are the same as for *Hourly* (name, address, phone number, social security number, hourly pay rate) and the 6th is the commission rate for the employee. The constructor should call the constructor of the parent class with the first 5 parameters then use the 6th to set the commission rate.
- One additional method is needed: *public void addSales (double totalSales)* that adds the parameter to the instance variable representing total sales.
- The *pay* method must call the pay method of the parent class to compute the pay for hours worked then add to that the pay from commission on sales. (See the pay method in the Executive class.) The total sales should be set back to 0 (note: you don't need to set the hoursWorked back to 0—why not?).
- The *toString* method needs to call the *toString* method of the parent class then add the total sales to that.

To test your class, update Staff.java as follows:

- Increase the size of the array to 8.
- Add two commissioned employees to the *staffList*—make up your own names, addresses, phone numbers and social security numbers. Have one of the employees earn $6.25 per hour and 20% commission and the other one earn $9.75 per hour and 15% commission.
- For the first additional employee you added, put the hours worked at 35 and the total sales $400; for the second, put the hours at 40 and the sales at $950.

Compile and run the program. Make sure it is working properly.

```
//********************************************************************
//  Firm.java       Author: Lewis/Loftus
//
//  Demonstrates polymorphism via inheritance.
//********************************************************************

public class Firm
{
   //-----------------------------------------------------------------
   //  Creates a staff of employees for a firm and pays them.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Staff personnel = new Staff();

      personnel.payday();
   }
}
```

```
//********************************************************************
//  Staff.java       Author: Lewis/Loftus
//
//  Represents the personnel staff of a particular business.
//********************************************************************

public class Staff
{
   StaffMember[] staffList;

   //-----------------------------------------------------------------
   //  Sets up the list of staff members.
   //-----------------------------------------------------------------
   public Staff ()
   {
      staffList = new StaffMember[6];

      staffList[0] = new Executive ("Sam", "123 Main Line",
         "555-0469", "123-45-6789", 2423.07);

      staffList[1] = new Employee ("Carla", "456 Off Line",
         "555-0101", "987-65-4321", 1246.15);
      staffList[2] = new Employee ("Woody", "789 Off Rocker",
         "555-0000", "010-20-3040", 1169.23);

      staffList[3] = new Hourly ("Diane", "678 Fifth Ave.",
         "555-0690", "958-47-3625", 10.55);

      staffList[4] = new Volunteer ("Norm", "987 Suds Blvd.",
         "555-8374");
      staffList[5] = new Volunteer ("Cliff", "321 Duds Lane",
         "555-7282");

      ((Executive)staffList[0]).awardBonus (500.00);

      ((Hourly)staffList[3]).addHours (40);
   }

   //-----------------------------------------------------------------
   //  Pays all staff members.
   //-----------------------------------------------------------------
   public void payday ()
   {
      double amount;

      for (int count=0; count < staffList.length; count++)
      {
         System.out.println (staffList[count]);

         amount = staffList[count].pay();  // polymorphic

         if (amount == 0.0)
            System.out.println ("Thanks!");
         else
            System.out.println ("Paid: " + amount);

         System.out.println ("-----------------------------------");
      }
   }
}
```

```
//********************************************************************
//  StaffMember.java       Author: Lewis/Loftus
//
//  Represents a generic staff member.
//********************************************************************

abstract public class StaffMember
{
   protected String name;
   protected String address;
   protected String phone;

   //-----------------------------------------------------------------
   //  Sets up a staff member using the specified information.
   //-----------------------------------------------------------------
   public StaffMember (String eName, String eAddress, String ePhone)
   {
      name = eName;
      address = eAddress;
      phone = ePhone;
   }

   //-----------------------------------------------------------------
   //  Returns a string including the basic employee information.
   //-----------------------------------------------------------------
   public String toString()
   {
      String result = "Name: " + name + "\n";

      result += "Address: " + address + "\n";
      result += "Phone: " + phone;

      return result;
   }

   //-----------------------------------------------------------------
   //  Derived classes must define the pay method for each type of
   //  employee.
   //-----------------------------------------------------------------
   public abstract double pay();
}
```

```
//********************************************************************
//  Volunteer.java       Author: Lewis/Loftus
//
//  Represents a staff member that works as a volunteer.
//********************************************************************

public class Volunteer extends StaffMember
{
   //-----------------------------------------------------------------
   //  Sets up a volunteer using the specified information.
   //-----------------------------------------------------------------
   public Volunteer (String eName, String eAddress, String ePhone)
   {
      super (eName, eAddress, ePhone);
   }

   //-----------------------------------------------------------------
   //  Returns a zero pay value for this volunteer.
   //-----------------------------------------------------------------
   public double pay()
   {
      return 0.0;
   }
}
```

```
//********************************************************************
//  Employee.java       Author: Lewis/Loftus
//
//  Represents a general paid employee.
//********************************************************************

public class Employee extends StaffMember
{
   protected String socialSecurityNumber;
   protected double payRate;

   //-----------------------------------------------------------------
   //  Sets up an employee with the specified information.
   //-----------------------------------------------------------------
   public Employee (String eName, String eAddress, String ePhone,
                    String socSecNumber, double rate)
   {
      super (eName, eAddress, ePhone);

      socialSecurityNumber = socSecNumber;
      payRate = rate;
   }

   //-----------------------------------------------------------------
   //  Returns information about an employee as a string.
   //-----------------------------------------------------------------
   public String toString()
   {
      String result = super.toString();

      result += "\nSocial Security Number: " + socialSecurityNumber;

      return result;
   }

   //-----------------------------------------------------------------
   //  Returns the pay rate for this employee.
   //-----------------------------------------------------------------
   public double pay()
   {
      return payRate;
   }
}
```

```
//********************************************************************
//  Executive.java       Author: Lewis/Loftus
//
//  Represents an executive staff member, who can earn a bonus.
//********************************************************************

public class Executive extends Employee
{
   private double bonus;

   //-----------------------------------------------------------------
   //  Sets up an executive with the specified information.
   //-----------------------------------------------------------------
   public Executive (String eName, String eAddress, String ePhone,
                     String socSecNumber, double rate)
   {
      super (eName, eAddress, ePhone, socSecNumber, rate);

      bonus = 0;  // bonus has yet to be awarded
   }

   //-----------------------------------------------------------------
   //  Awards the specified bonus to this executive.
   //-----------------------------------------------------------------
   public void awardBonus (double execBonus)
   {
      bonus = execBonus;
   }

   //-----------------------------------------------------------------
   //  Computes and returns the pay for an executive, which is the
   //  regular employee payment plus a one-time bonus.
   //-----------------------------------------------------------------
   public double pay()
   {
      double payment = super.pay() + bonus;

      bonus = 0;

      return payment;
   }
}
```

```
//********************************************************************
//  Hourly.java       Author: Lewis/Loftus
//
//  Represents an employee that gets paid by the hour.
//********************************************************************

public class Hourly extends Employee
{
   private int hoursWorked;

   //-----------------------------------------------------------------
   //  Sets up this hourly employee using the specified information.
   //-----------------------------------------------------------------
   public Hourly (String eName, String eAddress, String ePhone,
                  String socSecNumber, double rate)
   {
      super (eName, eAddress, ePhone, socSecNumber, rate);

      hoursWorked = 0;
   }

   //-----------------------------------------------------------------
   //  Adds the specified number of hours to this employee's
   //  accumulated hours.
   //-----------------------------------------------------------------
   public void addHours (int moreHours)
   {
      hoursWorked += moreHours;
   }

   //-----------------------------------------------------------------
   //  Computes and returns the pay for this hourly employee.
   //-----------------------------------------------------------------
   public double pay()
   {
      double payment = payRate * hoursWorked;

      hoursWorked = 0;

      return payment;
   }

   //-----------------------------------------------------------------
   //  Returns information about this hourly employee as a string.
   //-----------------------------------------------------------------
   public String toString()
   {
      String result = super.toString();

      result += "\nCurrent hours: " + hoursWorked;

      return result;
   }
}
```

Painting Shapes

In this lab exercise you will develop a class hierarchy of shapes and write a program that computes the amount of paint needed to paint different objects. The hierarchy will consist of a parent class Shape with three derived classes - Sphere, Rectangle, and Cylinder. For the purposes of this exercise, the only attribute a shape will have is a name and the method of interest will be one that computes the area of the shape (surface area in the case of three-dimensional shapes). Do the following.

1. Write an abstract class Shape with the following properties:
   - An instance variable shapeName of type String
   - An abstract method area()
   - A toString method that returns the name of the shape

2. The file *Sphere.java* contains a class for a sphere which is a descendant of Shape. A sphere has a radius and its area (surface area) is given by the formula 4*PI*radius^2. Define similar classes for a rectangle and a cylinder. Both the Rectangle class and the Cylinder class are descendants of the Shape class. A rectangle is defined by its length and width and its area is length times width. A cylinder is defined by a radius and height and its area (surface area) is PI*radius^2*height. Define the toString method in a way similar to that for the Sphere class.

3. The file *Paint.java* contains a class for a type of paint (which has a "coverage" and a method to compute the amount of paint needed to paint a shape). Correct the return statement in the amount method so the correct amount will be returned. Use the fact that the amount of paint needed is the area of the shape divided by the coverage for the paint. (NOTE: Leave the print statement - it is there for illustration purposes, so you can see the method operating on different types of Shape objects.)

4. The file *PaintThings.java* contains a program that computes the amount of paint needed to paint various shapes. A paint object has been instantiated. Add the following to complete the program:
   - Instantiate the three shape objects: deck to be a 20 by 35 foot rectangle, bigBall to be a sphere of radius 15, and tank to be a cylinder of radius 10 and height 30.
   - Make the appropriate method calls to assign the correct values to the three amount variables.
   - Run the program and test it. You should see polymorphism in action as the amount method computes the amount of paint for various shapes.

```
//*****************************************
//  Sphere.java
//
//  Represents a sphere.
//*****************************************
public class Sphere extends Shape
{
    private double radius;  //radius in feet

   //---------------------------------
   //  Constructor: Sets up the sphere.
   //---------------------------------
   public Sphere(double r)
   {
      super("Sphere");
      radius = r;
   }

   //---------------------------------------
   //  Returns the surface area of the sphere.
   //---------------------------------------
   public double area()
   {
       return 4*Math.PI*radius*radius;
   }
```

```
   //----------------------------------
   //  Returns the sphere as a String.
   //----------------------------------
   public String toString()
   {
       return super.toString() + " of radius " + radius;
   }

}



//*******************************************************
//   Paint.java
//
//   Represents a type of paint that has a fixed area
//   covered by a gallon. All measurements are in feet.
//*******************************************************

public class Paint
{
    private double coverage;  //number of square feet per gallon

    //-----------------------------------------
    //  Constructor:  Sets up the paint object.
    //-----------------------------------------
    public Paint(double c)
    {
        coverage = c;
    }

    //-------------------------------------------------
    //  Returns the amount of paint (number of gallons)
    //  needed to paint the shape given as the parameter.
    //-------------------------------------------------
    public double amount(Shape s)
    {
        System.out.println ("Computing amount for " + s);
        return 0;
    }

}
```

```
//********************************************************
//  PaintThings.java
//
//  Computes the amount of paint needed to paint various
//  things. Uses the amount method of the paint class which
//  takes any Shape as a parameter.
//********************************************************

import java.text.DecimalFormat;

public class PaintThings
{
    //-----------------------------------------
    // Creates some shapes and a Paint object
    // and prints the amount of paint needed
    // to paint each shape.
    //-----------------------------------------
   public static void main (String[] args)
   {
      final double COVERAGE = 350;
      Paint paint = new Paint(COVERAGE);

      Rectangle deck;
      Sphere bigBall;
      Cylinder tank;

      double deckAmt, ballAmt, tankAmt;

      // Instantiate the three shapes to paint

      // Compute the amount of paint needed for each shape


      // Print the amount of paint for each.
      DecimalFormat fmt = new DecimalFormat("0.#");
      System.out.println ("\nNumber of gallons of paint needed...");
      System.out.println ("Deck " + fmt.format(deckAmt));
      System.out.println ("Big Ball " + fmt.format(ballAmt));
      System.out.println ("Tank " + fmt.format(tankAmt));
   }
}
```

Polymorphic Sorting

The file *Sorting.java* contains the Sorting class from Listing 9.9 in the text. This class implements both the selection sort and the insertion sort algorithms for sorting any array of Comparable objects in ascending order. In this exercise, you will use the Sorting class to sort several different types of objects.

1. The file Numbers.java reads in an array of integers, invokes the selection sort algorithm to sort them, and then prints the sorted array. Save Sorting.java and Numbers.java to your directory. Numbers.java won't compile in its current form. Study it to see if you can figure out why.

2. Try to compile Numbers.java and see what the error message is. The problem involves the difference between primitive data and objects. Change the program so it will work correctly (note: you don't need to make many changes - the autoboxing feature of Java 1.5 will take care of most conversions from int to Integer).

3. Write a program Strings.java, similar to Numbers.java, that reads in an array of String objects and sorts them. You may just copy and edit Numbers.java.

4. Modify the insertionSort algorithm so that it sorts in descending order rather than ascending order. Change Numbers.java and Strings.java to call insertionSort rather than selectionSort. Run both to make sure the sorting is correct.

5. The file Salesperson.java partially defines a class that represents a sales person. This is very similar to the Contact class in Listing 9.10. However, a sales person has a first name, last name, and a total number of sales (an int) rather than a first name, last name, and phone number. Complete the compareTo method in the Salesperson class. The comparison should be based on total sales; that is, return a negative number if the executing object has total sales less than the other object and return a positive number if the sales are greater. Use the name of the sales person to break a tie (alphabetical order).

6. The file WeeklySales.java contains a driver for testing the compareTo method and the sorting (this is similar to Listing 9.8 in the text). Compile and run it. Make sure your compareTo method is correct. The sales staff should be listed in order of sales from most to least with the four people having the same number of sales in reverse alphabetical order.

7. OPTIONAL: Modify WeeklySales.java so the salespeople are read in rather than hardcoded in the program.

```
//********************************************************************
//  Sorting.java       Author: Lewis/Loftus
//
//  Demonstrates the selection sort and insertion sort algorithms.
//********************************************************************

public class Sorting
{
   //-----------------------------------------------------------------
   //  Sorts the specified array of objects using the selection
   //  sort algorithm.
   //-----------------------------------------------------------------
   public static void selectionSort (Comparable[] list)
   {
      int min;
      Comparable temp;

      for (int index = 0; index < list.length-1; index++)
      {
         min = index;
         for (int scan = index+1; scan < list.length; scan++)
            if (list[scan].compareTo(list[min]) < 0)
```

```
                min = scan;

            // Swap the values
            temp = list[min];
            list[min] = list[index];
            list[index] = temp;
        }
    }

    //-----------------------------------------------------------------
    //  Sorts the specified array of objects using the insertion
    //  sort algorithm.
    //-----------------------------------------------------------------
    public static void insertionSort (Comparable[] list)
    {
       for (int index = 1; index < list.length; index++)
       {
          Comparable key = list[index];
          int position = index;

          //  Shift larger values to the right
          while (position > 0 && key.compareTo(list[position-1]) < 0)
          {
             list[position] = list[position-1];
             position--;
          }

          list[position] = key;
       }
    }
}



// ********************************************************
//   Numbers.java
//
//   Demonstrates selectionSort on an array of integers.
// ********************************************************

import java.util.Scanner;

public class Numbers
{
    // -----------------------------------------
    //  Reads in an array of integers, sorts them,
    //  then prints them in sorted order.
    // -----------------------------------------
    public static void main (String[] args)
    {
        int[] intList;
        int size;

        Scanner scan = new Scanner(System.in);

        System.out.print ("\nHow many integers do you want to sort? ");
        size = scan.nextInt();
        intList = new int[size];

        System.out.println ("\nEnter the numbers...");
        for (int i = 0; i < size; i++)
            intList[i] = scan.nextInt();

        Sorting.selectionSort(intList);
```

```
        System.out.println ("\nYour numbers in sorted order...");
        for (int i = 0; i < size; i++)
            System.out.print(intList[i] + "  ");
        System.out.println ();
    }
}


// ********************************************************
//   Salesperson.java
//
//   Represents a sales person who has a first name, last
//   name, and total number of sales.
// ********************************************************

public class Salesperson implements Comparable
{
    private String firstName, lastName;
    private int totalSales;

    //-------------------------------------------------------
    //  Constructor:  Sets up the sales person object with
    //                the given data.
    //-------------------------------------------------------
    public Salesperson (String first, String last, int sales)
    {
        firstName = first;
        lastName = last;
        totalSales = sales;
    }

    //-----------------------------------------
    //  Returns the sales person as a string.
    //-----------------------------------------
    public String toString()
    {
        return lastName + ", " + firstName + ": \t" + totalSales;
    }


    //-----------------------------------------
    //  Returns true if the sales people have
    //  the same name.
    //-----------------------------------------
    public boolean equals (Object other)
    {
        return (lastName.equals(((Salesperson)other).getLastName()) &&
                firstName.equals(((Salesperson)other).getFirstName()));
    }

    //-------------------------------------------------
    //  Order is based on total sales with the name
    //  (last, then first) breaking a tie.
    //-------------------------------------------------
    public int compareTo(Object other)
    {
        int result;

        return result;
    }


    //-------------------------
    //  First name accessor.
    //-------------------------
    public String getFirstName()
    {
```

```
        return firstName;
    }

    //-------------------------
    //  Last name accessor.
    //-------------------------
    public String getLastName()
    {
        return lastName;
    }

    //-------------------------
    //  Total sales accessor.
    //-------------------------
    public int getSales()
    {
        return totalSales;
    }
}



// ************************************************************
//    WeeklySales.java
//
//    Sorts the sales staff in descending order by sales.
// ************************************************************

public class WeeklySales
{
    public static void main(String[] args)
    {
        Salesperson[] salesStaff = new Salesperson[10];

        salesStaff[0] = new Salesperson("Jane", "Jones", 3000);
        salesStaff[1] = new Salesperson("Daffy", "Duck", 4935);
        salesStaff[2] = new Salesperson("James", "Jones", 3000);
        salesStaff[3] = new Salesperson("Dick", "Walter", 2800);
        salesStaff[4] = new Salesperson("Don", "Trump", 1570);
        salesStaff[5] = new Salesperson("Jane", "Black", 3000);
        salesStaff[6] = new Salesperson("Harry", "Taylor", 7300);
        salesStaff[7] = new Salesperson("Andy", "Adams", 5000);
        salesStaff[8] = new Salesperson("Jim", "Doe", 2850);
        salesStaff[9] = new Salesperson("Walt", "Smith", 3000);

        Sorting.insertionSort(salesStaff);

        System.out.println ("\nRanking of Sales for the Week\n");

        for (Salesperson s : salesStaff)
            System.out.println (s);
    }
}
```

Searching and Sorting In 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
- int search(int target)—looks for value *target* in the list using a linear (also called sequential) search algorithm. Returns the index where it first appears if it is found, -1 otherwise.
- void selectionSort()—sorts the lists into ascending order using the selection sort algorithm.

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. For example, create a list, print it, and search for an element in the list. Does it return the correct index? Now look for an element that is not in the list. Now sort the list and print it to verify that it is in sorted order.

Modify the code in these files as follows:

1. Add a method *void replaceFirst(int oldVal, int newVal)* to the IntegerList class that replaces the first occurrence of oldVal in the list with newVal. If oldVal does not appear in the list, it should do nothing (but it's not an error). If oldVal appears multiple times, only the first occurrence should be replaced. Note that you already have a method to find oldVal in the list; use it!

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

2. Add a method *void replaceAll(int oldVal, int newVal)* to the IntegerList class that replaces all occurrences of oldVal in the list with newVal. If oldVal does not appear in the list, it should do nothing (but it's not an error). Does it still make sense to use the search method like you did for *replaceFirst*, or should you do your own searching here? Think about this.

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

3. Add a method *void sortDecreasing()* to the IntegerList class that sorts the list into decreasing (instead of increasing) order. Use the selection sort algorithm, but modify it to sort the other way. Be sure you change the variable names so they make sense!

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

4. Add a method *int binarySearchD (int target)* to the IntegerList class that uses a binary search to find the target assuming the list is sorted in decreasing order. Your algorithm will be a modification of the binary search algorithm in listing 9.12 of the text.

   Add an option to the menu in IntegerListTest to test your new method. In testing, make sure your method works on a list sorted in descending order then see what the method does if the list is not sorted (it shouldn't be able to find some things that are in the list).

```
// ********************************************************************
// IntegerList.java
//
// Define an IntegerList class with methods to create, fill,
// sort, and search in a list of integers.
//
// ********************************************************************

import java.util.Scanner;
```

```
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]);
    }

    //-------------------------------------------------------
    //return the index of the first occurrence of target in the list.
    //return -1 if target does not appear in the list
    //-------------------------------------------------------
    public int search(int target)
    {
      int location = -1;
      for (int i=0; i<list.length && location == -1; i++)
          if (list[i] == target)
            location = i;
      return location;
    }

    //-------------------------------------------------------
    //sort the list into ascending order using the selection sort algorithm
    //-------------------------------------------------------
    public void selectionSort()
    {
      int minIndex;
      for (int i=0; i < list.length-1; i++)
          {
            //find smallest element in list starting at location i
            minIndex = i;
            for (int j = i+1; j < list.length; j++)
                if (list[j] < list[minIndex])
                      minIndex = j;

            //swap list[i] with smallest element
            int temp = list[i];
```

```
            list[i] = list[minIndex];
            list[minIndex] = temp;
          }
    }
}



// ****************************************************************
// 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.selectionSort();
            break;
          case 3:
            System.out.print("Enter the value to look for: ");
            loc = list.search(scan.nextInt());
```

```
            if (loc != -1)
                System.out.println("Found at location " + loc);
            else
                System.out.println("Not in list");
            break;
          case 4:
            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: Sort the list using selection sort");
      System.out.println("3: Find an element in the list using linear search");
      System.out.println("4: Print the list");
      System.out.print("\nEnter your choice: ");
    }
}
```

Timing Searching and Sorting Algorithms

Chapter 9 has a brief discussion comparing sorting alorithms (page 506) and searching algorithms (page 513). In this exercise you will use an IntegerList class (in the file *IntegerList.java*) and a driver (in the file *IntegerListTest.java*) to examine the runtimes of the searching and sorting algorithms. The IntegerListTest class has several options for creating a list of a given size, filling the list with random integers or with already sorted integers, and searching or sorting the list. (NOTE: You may have used a version of these classes in the last lab.) Save these files to your directory and run IntegerListTest a few times to explore the options.

The runtimes of the sorting and searching algorithms can be examined using the Java method *System.currentTimeMillis()*, which returns the current system time in milliseconds. (Note that it returns a long, not an int.) You will have to import java.util.* to have access to this method. In IntegerListTest, just get the system time immediately before and immediately after you perform any of the searches or sorts. Then subtract the first from the second, and you have the time required for the operation in milliseconds. WARNING: Be sure you are not including any input or output in your timed operations; these are very expensive and will swamp your algorithm times!

Add appropriate calls to System.currentTimeMillis() to your program, run it and fill out the tables below. Note that you will use much larger arrays for the search algorithms than for the sort algorithms; do you see why? Also note that the first couple of times you run a method you might get longer runtimes as it loads the code for that method. Ignore these times and use the "steady-state" times you get on subsequent runs. On a separate sheet, explain the times you see in terms of the known complexities of the algorithms. Remember that the most interesting thing is not the absolute time required by the algorithms, but how the time changes as the size of the input increases (doubles here).

Array Size	Selection Sort (random array)	Selection Sort (sorted array)	Insertion Sort (random array)	Selection Sort (sorted array)
10,000				
20,000				
40,000				
80,000				

Array Size	Linear Search (unsuccessful)	Binary Search (unsuccessful)
100,000		
200,000		
400,000		
800,000		
1,600,000		

```
// ************************************************************
// FILE:  IntegerList.java
//
// Purpose: Define an IntegerList class with methods to create, fill,
//          sort, and search in a list of integers.
//
// ************************************************************

import java.util.Scanner;

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

    //-------------------------------------------------------------
    // Constructor -- takes an integer and creates a list of that
    // size.  All elements default to value 0.
    //-------------------------------------------------------------
    public IntegerList(int size)
    {
        list = new int[size];
    }


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


    //-------------------------------------------------------------
    // fillSorted -- fills the array with sorted values
    //-------------------------------------------------------------
    public void fillSorted()
    {
        for (int i=0; i<list.length; i++)
            list[i] = i + 2;
    }


    //-------------------------------------------------------------
    // print -- prints array elements with indices, one per line
    //-------------------------------------------------------------
    public String toString()
    {
        String s = "";
        for (int i=0; i<list.length; i++)
           s += i + ":\t" + list[i] + "\n";
        return s;
    }


    //-------------------------------------------------------------
    // linearSearch -- takes a target value and returns the index
    // of the first occurrence of target in the list.  Returns -1
    // if target does not appear in the list
    //-------------------------------------------------------------
    public int linearSearch(int target)
    {
        int location = -1;
        for (int i=0; i<list.length && location == -1; i++)
            if (list[i] == target)
```

```
                location = i;
        return location;
    }


    //-----------------------------------------------------------
    // sortIncreasing  -- uses selection sort
    //-----------------------------------------------------------
    public void sortIncreasing()
    {
        for (int i=0; i<list.length-1; i++)
            {
                int minIndex = minIndex(list, i);
                swap(list, i, minIndex);
            }
    }



// ************************************************************
// FILE:  IntegerListTest.java
//
// Purpose: 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);

    //-------------------------------------------------------
    // main -- creates an initial list, then repeatedly prints
    // the menu and does 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();
            }
    }

    //-------------------------------------------------------
    // dispatch -- takes a choice and does what needs doing
    //-------------------------------------------------------
    public static void dispatch(int choice)
    {
        int loc;
        int val;
        long time1, time2;
        switch(choice)
            {
            case 0:
                System.out.println("Bye!");
                break;
            case 1:
                System.out.println(list);
                break;
            case 2:
                System.out.println("How big should the list be?");
                list = new IntegerList(scan.nextInt());
                System.out.println("List is created.");
```

```
            break;
        case 3:
            list.randomize();
            System.out.println("List is filled with random elements.");
            break;
        case 4:
            list.fillSorted();
            System.out.println("List is filled with sorted elements.");
            break;
        case 5:
            System.out.print("Enter the value to look for: ");
            val = scan.nextInt();
            loc = list.linearSearch(val);
            if (loc != -1)
                System.out.println("Found at location " + loc);
            else
                System.out.println("Not in list");
            break;
        case 6:
            System.out.print("Enter the value to look for: ");
            val = scan.nextInt();
            loc = list.binarySearch(val);
            if (loc != -1)
                System.out.println("Found at location " + loc);
            else
                System.out.println("Not in list");
            break;
        case 7:
            list.sortIncreasing();
            System.out.println("List has been sorted.");
            break;
        case 8:
            list.sortDecreasing();
            System.out.println("List has been sorted.");
            break;
        default:
            System.out.println("Sorry, invalid choice");
        }
}


//-----------------------------------------------------
// printMenu -- prints 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: Print the list");
    System.out.println("2: Create a new list of a given size");
    System.out.println("3: Fill the list with random ints in range 1-length");
    System.out.println("4: Fill the list with already sorted elements");
    System.out.println("5: Use linear search to find an element");
    System.out.println("6: Use binary search to find an element " +
          "(list must be sorted in increasing order)");
    System.out.println("7: Use selection sort to sort the list into " +
          " increasing order");
    System.out.println("8: Use insertion sort to sort the list into " +
          " decreasing order");
    System.out.print("\nEnter your choice: ");
}
```

Coloring a Movable Circle

File *MoveCircle.java* contains a program that uses *CirclePanel.java* to draw a circle and let the user move it by pressing buttons. Save these files to your directory and compile and run MoveCircle to see how it works. Then study the code, noting the following:

- CirclePanel uses a BorderLayout so that the buttons can go on the bottom. But the buttons are not added directly to the south of the main panel—if they were they would all be on top of each other, and only the last one would show. Instead, a new panel *buttonPanel* is created and the buttons are added to it. *buttonPanel* uses a flow layout (the default panel layout), so the buttons will appear next to each other and centered. This panel is added to the south of the main panel.
- The listeners for the buttons are all instances of the MoveListener class, which is also defined here. The parameters to the constructor tell how how many pixels in the x and y directions the circle should move when the button is pressed.
- The circle is not drawn in the constructor, as it is not a component. It is drawn in paintComponent, which provides a graphics context for drawing on CirclePanel.
- In MoveCircle the frame size is explicitly set so there will be room to move the circle around.

Modify the program as follows.

1. Modify CirclePanel so that in addition to moving the circle, the user can press a button to change its color. The color buttons should be on the top of the panel; have four color choices. You will need to do the following:
   - Create a button for each color you want to provide, and label them appropriately.
   - Write a new listener class ColorListener whose constructor takes the color the circle should change to. When the button is pressed, just change the circle's color and repaint.
   - Create a new ColorListener for each color button, and add the listeners to the buttons.
   - Create a panel for the color buttons to go on, and add them to it.
   - Add the color panel to the north part of the main panel.

   You do not need to make any changes to MoveCircle.

2. Set the background or text (you choose) of each button to be the color that it represents.

3. Add another button to the top that says "Choose Color." Place the button in the middle of your other color buttons. When pressed, this button should bring up a JColorChooser, and the circle color should become the color that the user chooses. You can use the same ColorListener class that you used for the other buttons; just pass *null* for the color when the user wants to choose their own, and in the *actionPerformed* method bring up a JColorChooser if the color is *null*. Remember that the easiest way to use a JColorChooser is to call its static *showDialog* method, passing three parameters: the component to add it to (the "Choose Color" button), a string to title the chooser window, and a default color (the current circle color). Note that the "Choose Color" button will have to be an instance variable (instead of being local to the CirclePanel constructor like the other buttons) to be visible in the listener.

```
// ********************************************************************
//   MoveCircle.java
//
//   Uses CirclePanel to display a GUI that lets the user move
//   a circle by pressing buttons.
// ********************************************************************

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

public class MoveCircle
{
    //-----------------------------------
    //  Set up a frame for the GUI.
    //-----------------------------------
    public static void main(String[] args)
    {
      JFrame frame = new JFrame ("MoveCircle");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setPreferredSize(new Dimension(400,300));

      frame.getContentPane().add(new CirclePanel(400,300));

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

```
// ********************************************************************
//   CirclePanel.java
//
//   A panel with a circle drawn in the center and buttons on the
//   bottom that move the circle.
// ********************************************************************
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class CirclePanel extends JPanel
{
    private final int CIRCLE_SIZE = 50;
    private int x,y;
    private Color c;

    //-----------------------------------------------------------------
    // Set up circle and buttons to move it.
    //-----------------------------------------------------------------
    public CirclePanel(int width, int height)
    {
      // Set coordinates so circle starts in middle
      x = (width/2)-(CIRCLE_SIZE/2);
      y = (height/2)-(CIRCLE_SIZE/2);

      c = Color.green;

      // Need a border layout to get the buttons on the bottom
      this.setLayout(new BorderLayout());

      // Create buttons to move the circle
      JButton left = new JButton("Left");
      JButton right = new JButton("Right");
      JButton up = new JButton("Up");
      JButton down = new JButton("Down");

      // Add listeners to the buttons
      left.addActionListener(new MoveListener(-20,0));
      right.addActionListener(new MoveListener(20,0));
      up.addActionListener(new MoveListener(0,-20));
      down.addActionListener(new MoveListener(0,20));

      // Need a panel to put the buttons on or they'll be on
      // top of each other.
      JPanel buttonPanel = new JPanel();
      buttonPanel.add(left);
      buttonPanel.add(right);
      buttonPanel.add(up);
      buttonPanel.add(down);

      // Add the button panel to the bottom of the main panel
      this.add(buttonPanel, "South");

    }

    //-----------------------------------------------------------------
    // Draw circle on CirclePanel
    //-----------------------------------------------------------------
    public void paintComponent(Graphics page)
    {
```

```
      super.paintComponent(page);

      page.setColor(c);
      page.fillOval(x,y,CIRCLE_SIZE,CIRCLE_SIZE);
    }

    //-----------------------------------------------------------------
    // Class to listen for button clicks that move circle.
    //-----------------------------------------------------------------
    private class MoveListener implements ActionListener
    {
      private int dx;
      private int dy;

      //-----------------------------------------------------------------
      // Parameters tell how to move circle at click.
      //-----------------------------------------------------------------
      public MoveListener(int dx, int dy)
      {
          this.dx = dx;
          this.dy = dy;
      }

      //-----------------------------------------------------------------
      // Change x and y coordinates and repaint.
      //-----------------------------------------------------------------
      public void actionPerformed(ActionEvent e)
      {
          x += dx;
          y += dy;
          repaint();
      }
    }
}
```

Speed Control

The files *SpeedControl.java* and *SpeedControlPanel.java* contain a program (and its associated panel) with a circle that moves on the panel and rebounds from the edges. (NOTE: the program is derived from Listing 8.15 and 8.16 in the text. That program uses an image rather than a circle. You may have used it in an earlier lab on animation.) The Circle class is in the file *Circle.java*. Save the program to your directory and run it to see how it works.

In this lab exercise you will add to the panel a slider that controls the speed of the animation. Study the code in SlideColorPanel.java (Listing 9.16 in the text) to help understand the steps below.

1. Set up a JSlider object. You need to
   - Declare it.
   - Instantiate it to be a JSlider that is horizontal with values ranging from 0 to 200, initially set to 30.
   - Set the major tick spacing to 40 and the minor tick spacing to 10.
   - Set paint ticks and paint labels to true and the X alignment to left.

2. Set up the change listener for the slider. A skeleton of a class named *SlideListener* is already in SpeedControlPanel.java. You need to
   - Complete the body of the statedChanged function. This function must determine the value of the slider, then set the timer delay to that value. The timer delay can be set with the method *setDelay (int delay)* in the Timer class.
   - Add the change listener to the JSlider object.

3. Create a label ("Timer Delay") for the slider and align it to the left.

4. Create a JPanel object and add the label and slider to it then add your panel to the SOUTH of the main panel (note that it has already been set up to use a border layout).

5. Compile and run the program. Make sure the speed is changing when the slider is moved. (NOTE: Larger delay means slower!)

6. You should have noticed one problem with the program. The ball (circle) goes down behind the panel the slider is on. To fix this problem do the following:
   - In actionPerformed, declare a variable *slidePanelHt* (type int). Use the getSize() method to get the size (which is a Dimension object) of the panel you put the slider on. Assign *slidePanelHt* to be the height of the Dimension object. For example, if your panel is named *slidePanel* the following assignment statement is what you need:

     ```
     slidePanelHt = slidePanel.getSize().height;
     ```

   - Now use this height to adjust the condition that tests to see if the ball hits the bottom of the panel.
   - Test your program to make sure it is correct.

```
// ********************************************************************
//   SpeedControl.java
//
//   Demonstrates animation -- balls bouncing off the sides of a panel -
//   with speed controlled by a slider.
// ********************************************************************

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

public class SpeedControl
{
    // -----------------------------------
    //  Sets up the frame for the animation.
    // -----------------------------------
    public void static main (String[] args)
    {
      JFrame frame = new JFrame ("Bouncing Balls");
      frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

      frame.getContentPane.add(new SpeedControlPanel ());
      frame.pack();
      frame.setVisible(true);
    }
}
```

```
// ********************************************************************
//   SpeedControlPanel.java
//
//   The panel for the bouncing ball.  Similar to
//   ReboundPanel.java in Listing 8.16 in the text, except a circle
//   rather than a happy face is rebounding off the edges of the
//   window.
// ********************************************************************

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

public class SpeedControlPanel extends JPanel
{
    private final int WIDTH = 600;
    private final int HEIGHT = 400;
    private final int BALL_SIZE = 50;

    private Circle bouncingBall;   // the object that moves
    private Timer timer;
    private int moveX, moveY;      // increment to move each time

    // -----------------------------------------
    //  Sets up the panel, including the timer
    //  for the animation
    // -----------------------------------------
    public SpeedControlPanel ()
    {
      timer = new Timer(30, new ReboundListener());

      this.setLayout (new BorderLayout());

      bouncingBall = new Circle(BALL_SIZE);

      moveX = moveY = 5;

      // Set up a slider object here

      setPreferredSize (new Dimension (WIDTH, HEIGHT));
      setBackground(Color.black);
      timer.start();
    }

    // --------------------
    // Draw the ball
    // --------------------
    public void paintComponent (Graphics page)
    {
        super.paintComponent (page);
        bouncingBall.draw(page);
    }

    // *************************************************
    //   An action listener for the timer
    // *************************************************
    public class ReboundListener implements ActionListener
    {
      // -------------------------------------------------
```

```
      // actionPerformed is called by the timer -- it updates
      // the position of the bouncing ball
      // -----------------------------------------------------
      public void actionPerformed(ActionEvent action)
      {
          bouncingBall.move(moveX, moveY);

          // change direction if ball hits a side
          int x = bouncingBall.getX();
          int y = bouncingBall.getY();
          if (x < 0 || x >= WIDTH - BALL_SIZE)
            moveX = moveX * -1;

          if (y <= 0 || y >= HEIGHT - BALL_SIZE)
            moveY = moveY * -1;
          repaint();
      }
    }

    // *****************************************************
    //   A change listener for the slider.
    // *****************************************************
    private class SlideListener implements ChangeListener
    {
      // -------------------------------------------------
      //  Called when the state of the slider has changed;
      //  resets the delay on the timer.
      // -------------------------------------------------
      public void stateChanged (ChangeEvent event)
      {


      }
    }
}
```

```
// ************************************************************
// FILE:  Circle.java
//
// Purpose: 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 x, y;       // coordinates of the corner
    private int radius;     // radius of the circle
    private Color color;    // color of the circle

    static Random generator = new Random();

    //-----------------------------------------------------------
    // Creates a random circle with properties in ranges given:
    //   -- radius 25..74
    //   -- color RGB value 0..16777215 (24-bit)
    //   -- x-coord of upper left-hand corner 0..599
    //   -- y-coord of upper left-hand corner 0..399
    //-----------------------------------------------------------
    public Circle()
    {
        radius = Math.abs(generator.nextInt())%50 + 25;
        color = new Color(Math.abs(generator.nextInt())% 16777216);
        x = Math.abs(generator.nextInt())%600;
        y = Math.abs(generator.nextInt())%400;
    }

    //-----------------------------------------------------------
    // Creates a circle of a given size (diameter).  Other
    // attributes are random (as described above)
    //-----------------------------------------------------------
    public Circle(int size)
    {
        radius = Math.abs(size/2);
        color = new Color(Math.abs(generator.nextInt())% 16777216);
        x = Math.abs(generator.nextInt())%600;
        y = Math.abs(generator.nextInt())%400;
    }


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


    //-----------------------------------------------------------
    // Shifts the circle's position -- "over" is the number of
    // pixels to move horizontally (positive is to the right;
```

```
    // negative to the left); "down" is the number of pixels
    // to move vertically (positive is down; negative is up)
    //-----------------------------------------------------------
    public void move (int over, int down)
    {
        x = x + over;
        y = y + down;
    }

    // -------------------------------------------
    // Return the x coordinate of the circle corner
    // -------------------------------------------
    public int getX()
    {
        return x;
    }

    // -------------------------------------------
    // Return the y coordinate of the circle corner
    // -------------------------------------------
    public int getY()
    {
        return y;
    }
}
```