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 hours Worked 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;
   }
}
```

```java
//********************************************************************
//  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;
   }
}
```

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 WeeklyS ales .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. If the target is found, the method should return its index; otherwise the method should return –1. Your algorithm will be a modification of the binary search algorithm in listing 10.12 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.
//
// ****************************************************************
```

```
public class IntegerLis0074
{
    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: ");
    }
}
```