Chapter 4: Writing Classes
Lab Exercises

Topics	Lab Exercises
Classes and methods	Prelab Exercises
	A Bank Account Class
	Tracking Grades
	A Band Booster Class
	Representing Names
	Drawing Squares
GUIs: Buttons and Textfields	Voting with Buttons
	Calculating Body Mass Index

Prelab Exercises

1. Constructors are special methods included in class definitions.
   a. What is a constructor used for?
   b. How do constructors differ from other methods in a class?

2. Both methods and variables in a class are declared as either *private* or *public.* Describe the difference between private and public and indicate how a programmer decides which parts of a class should be private and which public.

3. Consider a class that represents a bank account.
   a. Such a class might store information about the account balance, the name of the account holder, and an account number. What instance variables would you declare to hold this information? Give a type and name for each.
   b. There are a number of operations that would make sense for a bank account—withdraw money, deposit money, check the balance, and so on. Write a method header with return type, name, and parameter list, for each such operation described below. Don't write the whole method—just the header. They will all be public methods. The first one is done for you as an example.
      i. Withdraw a given amount from the account. This changes the account balance, but does not return a value.

```
public void withdraw(double amount)
```

      ii. Deposit a given amount into the account. This changes the account balance, but does not return a value.
      iii. Get the balance from the account. This does not change anything in the account; it simply returns the balance.
      iv. Return a string containing the account information (name, account number, balance). This does not change anything in the account.
      v. Charge a $ 10 fee. This changes the account balance but does not return a value.
      vi. Create a new account given an initial balance, the name of the owner, and the account number. Note that this will be a constructor, and that a constructor does not have a return type.

A Bank Account Class

1. File *Account.java* contains a partial definition for a class representing a bank account. Save it to your directory and study it to see what methods it contains. Then complete the Account class as described below. Note that you won't be able to test your methods until you write ManageAccounts in question #2.

   a. Fill in the code for method *toString,* which should return a string containing the name, account number, and balance for the account.
   b. Fill in the code for method *chargeFee,* which should deduct a service fee from the account.
   c. Modify *chargeFee* so that instead of returning void, it returns the new balance. Note that you will have to make changes in two places.
   d. Fill in the code for method *changeName* which takes a string as a parameter and changes the name on the account to be that string.

2. File *ManageAccounts.java* contains a shell program that uses the Account class above. Save it to your directory, and complete it as indicated by the comments.

3. Modify ManageAccounts so that it prints the balance after the calls to chargeFees. Instead of using the getBalance method like you did after the deposit and withdrawal, use the balance that is returned from the chargeFees method. You can either store it in a variable and then print the value of the variable, or embed the method call in a println statement.

```
// ******************************************************
// Account.java
//
// A bank account class with methods to deposit to, withdraw from,
// change the name on, charge a fee to, and print a summary of the account.
// ******************************************************

public class Account
{
  private double balance;
  private String name;
  private long acctNum;

  // --------------------------------------------
  //Constructor -- initializes balance, owner, and account number
  // --------------------------------------------
  public Account(double initBal, String owner, long number)
  {
    balance = initBal;
    name = owner;
    acctNum = number;
  }

  // --------------------------------------------
  // Checks to see if balance is sufficient for withdrawal.
  // If so, decrements balance by amount; if not, prints message.
  // --------------------------------------------
  public void withdraw(double amount)
  {
    if (balance >= amount)
       balance -= amount;
    else
       System.out.println("Insufficient funds");
  }

  // --------------------------------------------
  //  Adds deposit amount  to balance.
  // ----------------------------------------------
  public void deposit(double amount)
  {
    balance += amount;
  }

  // ----------------------------------------------
  // Returns balance.
  // ----------------------------------------------
  public double getBalance()
  {
    return balance;
  }

  // ----------------------------------------------
  // Returns a string containing the name, account number, and balance.
  // ----------------------------------------------
  public String toString()
  {

  }

  // ----------------------------------------------
  // Deducts $10 service fee //
  // ----------------------------------------------
  public void chargeFee()
  {

  }

  // ----------------------------------------------
  // Changes the name on the account
  // ----------------------------------------------
  public void changeName(String newName)

  {

  }
}
```

```
// ***************************************************************
//   ManageAccounts.java
//
//   Use Account class to create and manage Sally and Joe's
//   bank accounts
// ***************************************************************

public class ManageAccounts
{

    public static void main(String[] args)
    {
      Account acct1, acct2;

      //create account1 for Sally with $1000
      acct1 = new Account(1000, "Sally", 1111);

      //create account2 for Joe with $500

      //deposit $100 to Joe's account

      //print Joe's new balance (use getBalance())

      //withdraw $50 from Sally's account

      //print Sally's new balance (use getBalance())

      //charge fees to both accounts

      //change the name on Joe's account to Joseph

      //print summary for both accounts

    }
}
```

Tracking Grades

A teacher wants a program to keep track of grades for students and decides to create a student class for his program as follows:

- Each student will be described by three pieces of data: his/her name, his/her score on test #1, and his/her score on test#2.
- There will be one constructor, which will have one argument—the name of the student.
- There will be three methods: *getName,* which will return the student's name; *inputGrades,* which will prompt for and read in the student's test grades; and *getAverage,* which will compute and return the student's average.

1. File *Student.java* contains an incomplete definition for the Student class. Save it to your directory and complete the class definition as follows:
   a. Declare the instance data (name, score for test1, and score for test2).
   b. Create a Scanner object for reading in the scores.
   c. Add the missing method headers.
   d. Add the missing method bodies.

2. File *Grades.java* contains a shell program that declares two Student objects. Save it to your directory and use the *inputGrades* method to read in each student's test scores, then use the *getAverage* method to find their average. Print the average with the student's name, e.g., "The average for Joe is 87." You can use the *getName* method to print the student's name.

3. Add statements to your Grades program that print the values of your Student variables directly, e.g.:

```
System.out.println("Student 1: " + student1);
```

This should compile, but notice what it does when you run it—nothing very useful! When an object is printed, Java looks for a *toString* method for that object. This method must have no parameters and must return a String. If such a method has been defined for this object, it is called and the string it returns is printed. Otherwise the default *toString* method, which is inherited from the Object class, is called; it simply returns a unique hexadecimal identifier for the object such as the ones you saw above.

Add a toString method to your Student class that returns a string containing the student's name and test scores, e.g.:

```
Name:   Joe     Test1:   85     Test2:   91
```

Note that the toString method does not call System.out.println—it just returns a string.

Recompile your Student class and the Grades program (you shouldn't have to change the Grades program—you don't have to call toString explicitly). Now see what happens when you print a student object—much nicer!

```
// *******************************************************
//   Student.java
//
//   Define a student class that stores name, score on test 1, and
//   score on test 2.  Methods prompt for and read in grades,
//   compute the average, and return a string containing student's info.
// *******************************************************
import java.util.Scanner;

public class Student
{

    //declare instance data

    // --------------------------------------------
    //constructor
    // --------------------------------------------
    public Student(String studentName)
    {

      //add body of constructor
    }

    // --------------------------------------------
    //inputGrades: prompt for and read in student's grades for test1 and test2.
    //Use name in prompts, e.g., "Enter's Joe's score for test1".
    // --------------------------------------------
    public void inputGrades()
    {

      //add body of inputGrades
    }

    // --------------------------------------------
    //getAverage: compute and return the student's test average
    // --------------------------------------------

    //add header for getAverage
    {

      //add body of getAverage
    }

    // --------------------------------------------
    //getName: print the student's name
    // --------------------------------------------

    //add header for printName
    {

      //add body of printName
    }

}
```

```
// *******************************************************
//   Grades.java
//
//   Use Student class to get test grades for two students
//   and compute averages
//
// *******************************************************
public class Grades
{

    public static void main(String[] args)
    {

      Student student1 = new Student("Mary");
      //create student2, "Mike"

      //input grades for Mary
      //print average for Mary

      System.out.println();

      //input grades for Mike
      //print average for Mike

    }
}
```

Band Booster Class

In this exercise, you will write a class that models a band booster and use your class to update sales of band candy.

1. Write the BandBooster class assuming a band booster object is described by two pieces of instance data: *name* (a String) and *boxesSold* (an integer that represents the number of boxes of band candy the booster has sold in the band fundraiser). The class should have the following methods:
   - A constructor that has one parameter—a String containing the name of the band booster. The constructor should set boxesSold to 0.
   - A method *getName* that returns the name of the band booster (it has no parameters).
   - A method *updateSales* that takes a single integer parameter representing the number of additional boxes of candy sold. The method should add this number to *boxesSold.*
   - A *toString* method that returns a string containing the name of the band booster and the number of boxes of candy sold in a format similar to the following:

     ```
     Joe:  16 boxes
     ```

2. Write a program that uses BandBooster objects to track the sales of 2 band boosters over 3 weeks. Your program should do the following:
   - Read in the names of the two band boosters and construct an object for each.
   - Prompt for and read in the number of boxes sold by each booster for each of the three weeks. Your prompts should include the booster's name as stored in the BandBooster object. For example,

     ```
     Enter the number of boxes sold by Joe this week:
     ```

     For each member, after reading in the weekly sales, invoke the *updateSales* method to update the total sales by that member.
   - After reading the data, print the name and total sales for each member (you will implicitly use the toString method here).

Representing Names

1. Write a class *Name* that stores a person’s first, middle, and last names and provides the following methods:
   - public Name(String first, String middle, String last)—constructor. The name should be stored in the case given; don’t convert to all upper or lower case.
   - public String getFirst()—returns the first name
   - public String getMiddle()—returns the middle name
   - public String getLast()—returns the last name
   - public String firstMiddleLast()—returns a string containing the person’s full name in order, e.g., “Mary Jane Smith”.
   - public String lastFirstMiddle()—returns a string containing the person’s full name with the last name first followed by a comma, e.g., “Smith, Mary Jane”.
   - public boolean equals(Name otherName)—returns true if this name is the same as otherName. Comparisons should not be case sensitive. (Hint: There is a String method *equalsIgnoreCase* that is just like the String method *equals* except it does not consider case in doing its comparison.)
   - public String initials()—returns the person’s initials (a 3-character string). The initials should be all in upper case, regardless of what case the name was entered in. (Hint: Instead of using *charAt,* use the *substring* method of String to get a string containing only the first letter—then you can upcase this one-letter string. See Figure 3.1 in the text for a description of the *substring* method.)
   - public int length()—returns the total number of characters in the full name, not including spaces.

2. Now write a program TestNames.java that prompts for and reads in two names from the user (you’ll need first, middle, and last for each), creates a Name object for each, and uses the methods of the Name class to do the following:
   a. For each name, print
      - first-middle-last version
      - last-first-middle version
      - initials
      - length

   b. Tell whether or not the names are the same.

Drawing Squares

1. Write a class Square that represents a square to be drawn. Store the following information in instance variables:
   - size (length of any side)
   - x coord of upper left-hand corner
   - y coord of upper left-hand corner
   - color

   Provide the following methods:
   - A parameterless constructor that generates random values for the size, color, x, and y. Make the size between 100 and 200, x between 0 and 600, y between 0 and 400. The squares can be any color—note that you can pass a single int parameter to the Color constructor, but it will only consider the first 24 bits (8 bits for each of R, G, B component).

   - IMPORTANT: Your random number generator must be declared at the class level (not inside the constructor), and must be declared *static.* So its declaration and initialization should appear with the declarations of size, color, x, and y, and should look like this:

     ```
     private static Random generator = new Random();
     ```

   - A *draw* method that draws the square at its x,y coordinate in its color. Note that you need a Graphics object, like the *page* parameter of the paint method, to draw. Your *draw* method should take a Graphics object as a parameter.

2. Now write an applet DrawSquares that uses your Square class to create and draw 5 squares. This code should be very simple; the *paint* method will simply create a Square and then draw it, repeated 5 times. Don't forget to pass the Graphics object to *draw.*

Voting with Buttons

Files *VoteCounter.java* and *VoteCounterPanel.java* contain slightly modified versions of *PushCounter.java* and *PushCounterPanel.java* in listings 4.10 and 4.11 of the text. As in the text the program counts the number of times the button is pushed; however, it assumes ("pretends") each push is a vote for Joe so the button and variables have been renamed appropriately.

1. Compile the program, then run it to see how it works.

2. Modify the program so that there are two candidates to vote for—Joe and Sam. To do this you need to do the following:
   a. Add variables for Sam—a vote counter, a button, and a label.
   b. Add a new inner class named *SamButtonListener* to listen for clicks on the button for Sam. Instantiate an instance of the class when adding the ActionListener to the button for Sam.
   c. Add the button and label for Sam to the panel.

3. Compile and run the program.

```
// *******************************************************
// VoteCounter.java
//
// Demonstrates a graphical user interface and event listeners to
// tally votes for two candidates, Joe and Sam.
// *******************************************************
import javax.swing.JFrame;

public class VoteCounter
{
    // ----------------------------------------------
    // Creates the main program frame.
    // ----------------------------------------------
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Vote Counter");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

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

```
// ******************************************************
// VoteCounterPanel.java
//
// Demonstrates a graphical user interface and event listeners to
// tally votes for two candidates, Joe and Sam.
// ******************************************************

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

public class VoteCounterPanel extends JPanel
{
    private int votesForJoe;
    private JButton joe;
    private JLabel labelJoe;

    // ----------------------------------------------
    // Constructor: Sets up the GUI.
    // ----------------------------------------------
    public VoteCounterPanel()
    {
       votesForJoe = 0;

       joe = new JButton("Vote for Joe");
       joe.addActionListener(new JoeButtonListener());

       labelJoe = new JLabel("Votes for Joe: " + votesForJoe);

       add(joe);
       add(labelJoe);

       setPreferredSize(new Dimension(300, 40));
       setBackground(Color.cyan);
    }

    // ***********************************************
    // Represents a listener for button push (action) events
    // ***********************************************
    private class JoeButtonListener implements ActionListener
    {
        //--------------------------------------------
        // Updates the counter and label when Vote for Joe
        // button is pushed
        //--------------------------------------------
        public void actionPerformed(ActionEvent event)
        {
            votesForJoe++;
            labelJoe.setText("Votes for Joe: " + votesForJoe);
        }
    }
}
```

Calculating Body Mass Index

Body Mass Index (BMI) is measure of weight that takes height into account. Generally, a BMI above 25 is considered high, that is, likely to indicate that an individual is overweight. BMI is calculated as follows for both men and women:

```
       (703 * height in inches) / (weight in pounds)2
```

Files *BMI.java* and *BMIPanel.java* contain skeletons for a program that uses a GUI to let the user compute their BMI. This is similar to the Fahrenheit program in listings 4.12 and 4.13 of the text. Fill in the code as indicated by the comments and compile and run this program; you should see the BMI calculator displayed.

```
// *********************************************************
//  BMI.java
//
//  Sets up a GUI to calculate body mass index.
// *********************************************************

import javax.swing.JFrame;
public class BMI
{
   // ------------------------------------------
   //  Creates and displays the BMI GUI.
   // ------------------------------------------
   public static void main (String[] args)
   {

       JFrame frame = new JFrame("BMI");
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


       BMIPanel panel = new BMIPanel();


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

```
//*********************************************************
//  BMIPanel.java
//
//  Computes body mass index in a GUI.
//*********************************************************

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

public class BMIPanel extends JPanel
{
   private int WIDTH = 300;
   private int HEIGHT = 120;
   private JLabel heightLabel, weightLabel, BMILabel, resultLabel;
   private JTextField height, weight;
   private JButton calculate;

   // ------------------------------------------------------------------
   //  Sets up the GUI.
   // ------------------------------------------------------------------
   public BMIPanel()
   {
      //create labels for the height and weight textfields
      heightLabel = new JLabel ("Your height in inches: ");
      weightLabel = new JLabel ("Your weight in pounds: ");


      //create a "this is your BMI" label
      //create a result label to hold the BMI value


      //create a JTextField to hold the person's height in inches
      //create a JTextField to hold the person's weight in pounds


      //create a button to press to calculate BMI
      //create a BMIListener and make it listen for the button to be pressed


      //add the height label and height textfield to the panel
      //add the weight label and weight textfield to the panel
      //add the button to the panel
      //add the BMI label to the panel
      //add the label that holds the result to the panel

      //set the size of the panel to the WIDTH and HEIGHT constants
      //set the color of the panel to whatever you choose
   }


   // *************************************************************
   //  Represents an action listener for the calculate button.
   // *************************************************************
   private class BMIListener implements ActionListener
   {
      // ----------------------------------------------
      //  Compute the BMI when the button is pressed
      // ----------------------------------------------
      public void actionPerformed (ActionEvent event)
      {
         String heightText, weightText;
         int heightVal, weightVal;
         double bmi;

         //get the text from the height and weight textfields

         //Use Integer.parseInt to convert the text to integer values

         //Calculate the bmi = 703 * weight in pounds / (height in inches)^2

         //Put result in result label.  Use Double.toString to convert double to
string.

      }
   }
```

```
}
```