Chapter 4: Writing Classes
Lab Exercises

Using the Coin Class

In folder `CoinProject` are files `Coin.java` and `CoinTest.java`. You will be writing a program to find the length of the longest run of heads in 100 flips of the coin. A skeleton of the program is in the file `CoinTest.java`. To use the `Coin` class you need to do the following in the `CoinTest` program:

1. Create a `Coin` object.
2. Inside the loop, you should use the `flip` method to flip the coin, the `toString` method (used implicitly) to print the results of the flip, and the `getFace` method to see if the result was HEADS. Keeping track of the current run length (the number of times in a row that the coin was HEADS) and the maximum run length is an exercise in loop techniques!
3. Print the maxiumum run length after the loop.

```
// ********************************************************************
//   Coin.java            Author: Lewis and Loftus
//
//   Represents a coin with two sides that can be flipped.
// ********************************************************************

public class Coin
{
    public final int HEADS = 0;
    public final int TAILS = 1;

    private int face;

    // -----------------------------------------------
    //   Sets up the coin by flipping it initially.
    // -----------------------------------------------
    public Coin ()
    {
      flip();
    }

    // -----------------------------------------------
    //   Flips the coin by randomly choosing a face.
    // -----------------------------------------------
    public void flip()
    {
      face = (int) (Math.random() * 2);
    }

    // -----------------------------------------------------------
    //   Returns true if the current face of the coin is heads.
    // -----------------------------------------------------------
    public boolean isHeads()
    {
      return (face == HEADS);
    }

    // -----------------------------------------------------
    //   Returns the current face of the coin as a string.
    // -----------------------------------------------------
    public String toString()
    {
      String faceName;

      if (face == HEADS)
```

```
            faceName = "Heads";
        else
            faceName = "Tails";

        return faceName;
    }
}



// ********************************************************************
// CoinTest.java
//
// Finds the length of the longest run of heads in 100 flips of a coin.
// ********************************************************************

public class CoinTest
{
    public static void main (String[] args)
    {
      final int FLIPS = 100; // number of coin flips

      int currentRun = 0; // length of the current run of HEADS
      int maxRun = 0;     // length of the maximum run so far

      // Create a coin object

      // Flip the coin FLIPS times
      for (int i = 0; i < FLIPS; i++)
          {
            // Flip the coin & print the result

            // Update the run information

          }

      // Print the results

    }
}
```

Example Output: (the number of output lines has been shorted from 100 to save space)

```
Tails
Heads
Heads
Heads
Heads
Heads
Heads
Tails
Tails
Tails
Heads
Heads
Tails
Heads
Heads
The maxiumum run of HEADS was 6
```

A Bank Account Class

1. File `Account.java` contains a partial definition for a class representing a bank account. Open the file in the `AccountsProject` folder 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.java` 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. Note that the `toString` method does not call `System.out.println`—it just returns a string.
   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. 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 $1000 from Joe’s account

      //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

    }
}
```

Sample Output:

```
Joe's balance: $600.00

Insufficient funds
Sally's balance: $950.00

Service fee charged. Sally's balance $940.00

Service fee charged. Joe's balance $590.00

Account Summary for Sally
Account Number: 1111
Balance: $940.00

Account Summary for Joeseph
Account Number: 2222
Balance: $590.00
```

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` in folder `GradesProject` contains an incomplete definition for the `Student` class. Open the file 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. 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. Create a folder called `BandProject` which will contain `BandBooster.java` and `BandTest.java` files.

1. Write the `BandBooster` class assuming a band booster object is described by two pieces of instance data: `name` (a String that represents the name of the band booster) 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 called `BandTest` 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 for week 1:
     ```

     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).

Sample Output:

```
Please enter the name of the band booster: Sally
Please enter the name of the band booster: Johnny

Enter the number of boxes sold by Sally for week 1: 20
Enter the number of boxes sold by Sally for week 2: 24
Enter the number of boxes sold by Sally for week 3: 13

Enter the number of boxes sold by Johnny for week 1: 22
Enter the number of boxes sold by Johnny for week 2: 8
Enter the number of boxes sold by Johnny for week 3: 19

Sally sold 57 boxes.

Johnny sold 49 boxes.
```

Representing Names

1. Create a folder called `NameProject`. Write a class `Name.java` 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: Use the `substring` method of `String` to get a string containing only the first letter—then you can upcase this one-letter string.)
   - `public int length()`—returns the total number of characters in the full name, **not** including spaces.

2. Now write a program `NameTest.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.

Sample Output:

```
Please enter the name of person #1 (first middle last): Sally Jo Brown
Please enter the name of person #2(first middle last): Tom Michael Green

Sally Jo Brown
Brown, Sally Jo
SJB
The name length is 12

Tom Michael Green
Green, Tom Michael
TMG
The name length is 15

Both names are different
```

Stick Figure

Often an object has a graphical representation. The `LineUp` applet in the textbook creates several `StickFigure` objects, of different colors and heights. The `StickFigure` objects are instantiated in the `init` method of the applet, so they are created only once, when the applet is first loaded.

The `paint` method asks that the stick figures redraw themselves whenver the mehod is called. The `paint` method is called whenever something happens that might change the graphic representation of the applet itselft. For instance, when the window that the applet is displayed in is moved, `paint` redraws the applet contents.

A `StickFigure` object contains data that defines its state, such as the position, color and height of the figure. The `draw` method contains the individual commands that draw the figure itself.

Pages 240 – 242 in the text list the code for two class files, `LineUp.java` and `StickFigure.java`. Create a folder in Chapter 4, called `StickProject`. Create the two `.java` files and copy the code from the textbook. You have a pdf copy of chapter 4 in your Chapter 4 lab folder. Create an HTML file, `LineUp.html`. (Copy an html file you have used in a previous lab and change the applet code to `LineUp.class`).

```
<html>
<applet code="LineUp.class" width=600 height=400>
</applet>
</html>
```

1. Modify the code in the book to use the `Math.random()` method rather than the `Random` class. Remove the `java.util.Random` import statement at the top.

2. Use the `Math.random()` method to generate a random integer between 1 and VARIANCE.

3. Modify the program to draw 5 stick figures, each one a different color. You will need to adjust the size of the applet width so that the 5 figures are centered in the window.

Meeting Finch

Write a program, `FinchCommands.java,` which contains an algorithm that has the Finch perform a specific action. You must use all the commands below at least once. Remember to document all of your code.

Output to the screen: `System.out.print("This will output on the screen.");`
Finch Pause: `myFinch.sleep(1000); // in milliseconds (1000 = 1 second)`
Finch Beak Color: `myFinch.setLED(red,green,blue); // 0 to 255`
Finch Speak: `myFinch.saySomething("The Finch will say this.");`
Finch Moves: `myFinch.setWheelVelocities(left,right,time) // -255 to 255 & time is // in milliseconds`

- Look at the Javadocs for the Finch class for more information on the methods listed above at http://www.finchrobot.com/javadoc/index.html

- Use the `FinchTemplateFile.java` program as a template for your program.

```
// Needs a package declaration to move to another folder

import edu.cmu.ri.createlab.terk.robot.finch.Finch;

/**
 * Created by:
 * Date:
 * A starter file to use the Finch
 */

public class FinchTemplateFile
   {
   public static void main(final String[] args)
      {
      // Instantiating the Finch object
      Finch myFinch = new Finch();

      // Write some code here!

      // Always end your program with finch.quit()
      myFinch.quit();
      System.exit(0);
      }
   }
```

Finch Security

Write a program, `FinchSecurity.java`. Use the Finch as its own security system. Place the Finch in front of a door (or obstacle representing a door). Using the object sensors, while the door is shut, Finch should keep still and blink a red led. If the door is open (obstacle gone) then the Finch will go crazy with light, sounds and movements. Test your program by putting an obstacle (representing the door) in front of the Finch, then take the obstacle away.

Put your algorithm inside a `while` loop and continue acting as a security system until you tilt the Finch up (look at the `isBeakUp()` or `isFinchLevel()` methods).