Faculty of Arts & Sciences
Department of Computer Science
CMPS 200—Introduction to Programming
Assignment 13 – Due Friday Jan 4, 2013

Reading Material

Chapter 6. The chapter covers the built-in Java I/O datatypes including the `File`, `Scanner`, and `PrintStream` datatypes. In this assignment, all I/O must be done with the standard Java I/O APIs i.e. File, Scanner and PrintStream. You may not use the `In/Out` classes we used in previous assignments.

Exercises

1. Courses. The goal of this simple exercise is to illustrate how to handle data types whose instance variables are of object type. The Course data type below whose variables include TextBook and Instructor. Write the classes and answer the question below with an appropriate diagram.

```
public class TextBook
private String title, author, publisher;                          // instance variables
public         TextBook(String titl, String auth, String pub)     // basic constructor
public         TextBook(TextBook t)                               // copy constructor
public void    set(String title, String author, String publisher) // mutator method
public String toString()                                  // string representation of textbook
```

```
public class Instructor
private String lastname, firstname, office;
public          Instructor(String lname, String fname, String office) // basic constructor
public          Instructor(Instructor inst)                           // copy constructor
public void     set(String lname, String fname, String office)
public String   toString()                          // string representation of textbook
```

```
public class Course
private String title;
private Instructor instr;
private TextBook  text;
public            Course(String name, Instructor ins, TextBook txt)   // basic constructor
public String     getName()                                           // accessor method
public Instructor getInstructor()                                     // accessor method
public TextBook   getTextBook()                                       // accessor method
public String     toString()                                          // string representation of course
```

```
public class CourseTest
{
   public static void main(String[] args)
   {
      // Create Instructor objects
      Instructor instructor1 = new Instructor("Turkiyyah", "George", "Bliss 116");
      Instructor instructor2 = new Instructor("Jaber", "Mohamad", "Bliss 309");

      // Create TextBook objects
      TextBook text1 = new TextBook("Building Java Programs", "Stepp", "Pearson");

      // Create Course objects
      Course course1 = new Course("CMPS 200", instructor1, text1);
      Course course2 = new Course("CMPS 200", instructor2, text1);

      // Display the courses information.
      System.out.println(course1);
      System.out.println(course2);
   }
```

Question: Explain what problems might occur if the constructor of the `Course` class does not call the copy constructors for its `instructor` and `textbook` instance variables. *Draw appropriate diagrams.*

2. Rotating Sentences. You are asked to rotate a series of input sentences 90 degrees clockwise, i.e., instead of displaying them from left to right and top to bottom, your program will display them from top to bottom and right to left. Write RotatingSequence program that takes the names of the input and output files as command line arguments. The file can contain a maximum of 100 sentences. Legal characters include: newline, space, any punctuation characters, digits, and lower case or upper case English letters. (NOTE: Tabs are not legal characters.) The output file should have the last sentence printed out vertically in the leftmost column; the first sentence of the input would subsequently end up in the rightmost column.

Hint.

Write a data type JaggedArrayChar that implements the methods whose signatures are shown below.

```
public class JaggedArrayChar {
    private int nbLines; // number of lines
    private int maxLineLen; // maximum line's length
    private char[][] jaggedArray = new char[100][];

    // constructor, takes a file name and fill its content in the jaggedArray.
    // Also, this constructor stores the number lines in nbLines and the maximum line's length in maxLineLen.
    public      JaggedArray(String filename) {...}
    public char get(int i, int j) {...}     // returns jaggedArray[i][j]
    public int  getLineLength(int i) {...}  // returns jaggedArrray[i].length
    public int  getNbLines() {...}          // returns nbLines
    public int  getMaxLineLen() {...}       // returns maxLineLen
}
```

Sample Input:

```
Rene Decartes once said,
"Cogito ergo Sum", which means,
"I think therefore I am."
```

Sample Output:

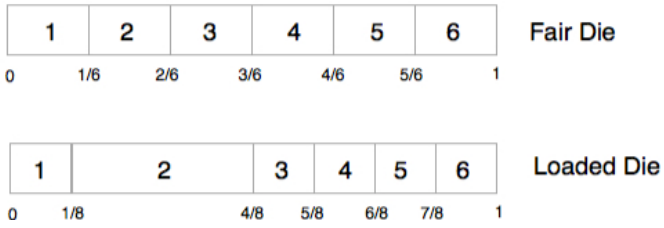
```
""R
ICe
 on
tge
hi
itD
noe
k c
 ea
trr
hgt
eoe
r s
eS
fuo
omn
r"c
e,e

Iws
 ha
aii
mcd
.h,
"
 m
 e
 a
 n
 s
 ,
```

3. Dice

In this problem, you are going to perform some experiments that result from rolling two six-sided dice. You will start by creating two classes `Die` and `DicePair` that will be used to answer follow-up questions. `Die` and `DicePair` are given as follows:

```
public class Die {
    public   Die()                          // basic constructor
    public   Die(int face, int n)           // Constructs a die where a given face is "loaded, i.e.,
                                            //  is n times more likely to occur than the others
   public int roll()                        // returns a random integer between 1 and 6
}
```

To implement a die where all six faces are equally likely to occur (fair die), the interval 0.0 - 1.0 could be divided into six segments of equal size. A random number between 0.0 and 1.0 can then be generated and and the segment it falls in is the roll of the die. If one of the faces is loaded, then the segment sizes have to be adjusted accordingly. For example if a 2 is three times more likely to occur than the other faces, the size of the segment corresponding to 2 has to be three times larger than the others. See figure below.




```
public class DicePair {
    private Die[] dies;                 // instance variables
    public      DicePair()  {..}        // constructor
    public      DicePair(...)  {..}     // constructor for a loaded pair of dice
   public int  roll() {..}              // returns the sum of the values returned from rolling each Die
}
```

3.1 DiceRun

Write a program DiceRun.java to roll a pair of dice a certain number of times and store the results in a file. The program accepts two parameters from the command line: an integer N and a file name f. It will start by writing the integer N to the file. It will then call the roll() method of the DicePair class N times and store each roll value in the file on a separate line. Note that you will get different results at every run.

```
C:> java DiceRun 5 dice-pair-runs.txt
```

After running the command above `dice-pair-runs.txt` contains something like:

```
5
7
3
3
10
9
```

3.2 CountOccurrences

Write a program CountOccurrences.java that accepts 2 arguments from the command line: the first is an integer (call it key) between 2 and 12 inclusive and the second is an integer N denoting the number of rolls to perform. Your program should roll the dice pair N times and output the number of occurrences of key in the rolls.

```
C:> java CountOccurrences 3 6000
3 was rolled 306 times out of 6000
C:> java CountOccurrences 12 10000
12 was rolled 276 times out of 10000
```

3.3 FairDie

Write a driver class called FairDie that checks the values of n rolls of a single Die and then decides whether the die is fair or not. The program takes an integer N that represents the number of times the die is rolled. If the Die is fair, the program should display on the console "Fair Die". Otherwise, it will display "Unfair Die". A Die is considered fair if all six outcomes (1 through 6) are equally likely to occur. The chi-square test is usually used to check whether the die is fair or not. The following formula calculates chi-square:

$$X^2 = \frac{1}{6n^2}\sum_{i=1}^{6}(6C_i - n)^2$$

where X^2 is the chi-square statistic (find it on Wikipedia), C_i is the number of occurrences of *i*, i.e. the number of occurrences of *1* is C_1, the number of occurrences of *2* is C_2, the number of occurrences of *3* is C_3, etc., and *n* is the number of times the die was rolled. A Die is considered fair if the value of X^2 is less than 0.02. Otherwise, it is not fair. You program should display if the die is fair or not and the value of Chi-Square.

```
C:> java FairDie 10000
Chi-Square = 1.2656E-4
Fair Die
```