COMS W1007x: Honors Intro to Computer Science
Fall, 2012
Assignment 2: Due Tuesday, October 9, 2011, 1:10pm, in class

Part 1: Theory (44 points)

The following problems are worth the points indicated. They are generally based on the book's exercises at the ends of Chapters 3 and 4, under the section heading "Exercises" and "Programming Projects". They cover concepts involving Classes and some GUIs

"Paper" programs are those which are written on the same sheet that the rest of the theory problems are written on, and are not to be text-edited, compiled, executed, or debugged. For fairness, there will be no extra credit for doing any work beyond the design and coding necessary to produce on paper the objects or object fragments that are asked for, so computer output will have no effect on your grade. The same goes for the Theory Part in general: clear handwriting is sufficient, and computer-generated documents will earn no extra credit.

1) (5 points) Based on 3.3: Write a declaration that initializes a String to that of another String, except that the characters are converted to upper case.

2) (6 points) Based on 3.7: Write random numbers in the range [4, 5] (that is, equally likely to be 4 or 5), and in the range [-1, 1] (that is, equally likely to be -1, 0, or 1).

3) (6 points) Based on 3.10: Do it, except *also* do the answer for a different Locale, so that the output should look different in some way. See the Java API for details.

4) (5 points) Based on 3.12: Write an enum for the borough of New York City (there are 5), and then show how to set a variable using it to Manhattan.

5) (6 points) Based on 4.2 to 4.4: Except do it for the class PizzaShop, from Assignment 1.

6) (6 points) Based on 4.9: Except write a method called lightenColor that accepts a Color, adds a random amount of white to it (that is, adds equal values of red, green, and blue to it), and returns the Color. Make sure that the Color that is returned is legal, that is, each of the red, green, and blue values are within the range [0, 255].

7) (10 points) Draw a UML diagram for what you produce for the Programming Part. See Figure 4.4 to get started. The book also shows some examples in Figure 7.3, 8.3, 9.1, 10.1, etc.

Part 2: Programming (56 points)

This assignment is intended to exercise some the more advanced concepts in class construction and application program GUIs.

Please recall that all programs must compile, so keep a working version of each part before your proceed to the next.

Step 1 (12 points): Getting started.

Using the code available in the book's Online Companion or on Courseworks, implement the class Die

given on pages 165 and 166. But augment the class with the following:

1) Add a constructor that allows a Die to start at a given legal value, that is, public Die(int initial). You must decide what to do if the value is not legal.

2) Add a constructor that allows a Die to be created from a given Die, that is public Die(Die existing). You must decide whether or not to trust the existing die, and what to do if you suspect something is wrong with it.

3) Modify the method setFaceValue so it is likewise safe.

4) Modify the method toString so that it shows, in addition the face value integer, a small string graphic to demonstrate the value. For example, instead of just "5", return "XXXXX", or use the '\n' character to make it fancy to look like a die, like

```
X X
 X
X X
```

5) Implement the method equals, so that a Die can be compared to a Die directly, so that people using your class do not have to extract facevalues. That is, myDie1.equals(myDie2) should be legal.

Do what is necessary in a Tester class to show that your Die class works as required and as documented.

Turn in a hardcopy listing of your code, and some trial runs. Remember to use the Javadoc conventions to properly document your classes, constructors, and methods. Also, submit a softcopy listing of your code, your trial runs, and the *softcopy* of your Javadoc outputs for this Step to the *Assignments* page of Courseworks.

Step 2 (12 points): An exercise in extending a design.

Implement the class PairOfDice, which uses Die.

1) It should have a constructor that creates a PairOfDice from individual Die. That is, public PairofDice(Die myDie1, Die myDie2). You must decide if your constructors for this class trust the Dies they are given.

2) You must decide if it makes sense to allow PairOfDice to have methods that accept faceValues directly in constructors, or to return faceValues directly from methods. As part of this, you should decide if you need a method that takes a PairOfDice and returns the first Die and/or the second Die that make it up.

3) You should have a roll() method which rolls the pair, using the method(s) from Die. What should this return?

4) You should have a toString() method that returns some output based on the individual Die toString() methods.

5) You should have three boolean methods: isSnakeEyes(), that is, the sum of the faceValues of the two dice equals 2; isBoxCars(), the sum is 12; and isSeven(), the sum is 7 . We will be playing a simplified version of the game of Craps, so there is no need to use the internet to find out the "official" meaning of these terms or of the rules, which are much too complicated.

Do what is necessary in a Tester class to show that your Die class works as required and as documented.

If you do this Step, you do not have to submit separate output from Step 1, as your output on Step 2 takes care of the predecessor step. But this Step still requires the same *kinds* of output that are detailed in Step 1 above.

Step 3 (17 points): Make it into a graphics application.

Using the code segments in Chapter 3 and 4, particularly JLabel and JButton, write a class that interfaces with a player that has the following functionality:

1) A place to display the PairOfDice, using the toString methods. It would be nicer if they were side-to-side rather than top-to-bottom.

2) A place to push a button that rolls the PairOfDice.

3) A place that shows the total number of points on the PairofDice.

4) A place that indicates in some way if the points are "special", that is, if they are one of the three booleans given above.

5) A listener for the JButton (see page 194) that properly works together with the rest of the logic of your program. This can get tricky.

Do what is necessary in a Tester class to show that your Die class works as required and as documented.

If you do this Step, you do not have to submit separate output from Steps 1 and 2, as your output on Step 3 takes care of the predecessor steps. But this Step still requires the same *kinds* of output that are detailed in Step 1 above.

Step 4 (15 points): Creativity Step. Play Craps, and cheat.

Here are a simplified set of rules for Craps. The player rolls a pair of dice. If it isSeven(), the player wins. If it is not, the player takes note of the total points, and rolls repeatedly, trying to reproduce the point total. The player stops if the total comes up the same, in which case the player wins, or if the total isSeven(), in which case the player loses. Otherwise the player rolls again. Please note that isSnakeEyes() and isBoxCars() have been added only for fanciness: they don't affect the play at all, other than to cause grief to the player, since they are the least likely combinations to be reproducible.

Augment your GUI so that it has also:

1) A place to show whether it is a new game or a game in progress.

2) A place to say whether the player should roll, or has won, or has lost.

3) A place to say what the point value (if any) the player is trying to reproduce.

4) You must also have something in Tester class that allows your system to cheat, and to win more often than it should. How you decide to do this is up to you, but you have to clearly document it. It would be good design practice to put all of the code for cheating into a single method, so it can be removed quickly if the authorities (or sore losers) are about to bust your gambling establishment.

If you do this Step, you do not have to submit separate output from Steps 1, 2, and 3, as your output on Step 4 takes care of the predecessor steps. But this Step still requires the same *kinds* of output that are detailed in Step 1 above.

General Notes:

For each Step, design and document the system, text edit its components into files, compile them, debug them, and execute them on your own test data. When you are ready, electronically submit the text of its code, a softcopy of your Javadoc output, a copy of your testing (including any screenshots), and whatever additional documentation is required. Make sure that the source code is clear and any user interface is comprehensive and informative. Make sure you have properly attributed any code that is legal to incorporate from other sources, including the book's online companion.

Write your classes clearly, with a large block comment at the beginning describing what the class does, how it does it, and justify how it will be tested. Put the comments about testing in a Tester class; if possible, show some test data in the comment also. Document each constructor and method.

Clear programming style will account for a substantial portion of your grade for both the Theory Part and the Programming Part of this assignment. For one reasonable set of suggested practices and stylistic guidelines, see the book's Appendix F. But it is far more important to be consistent: PoLS, everybody!

Checklist:

Here is a checklist for submission. Please note that this is designed to be a *general* guide, and you are responsible for *all* things asked for in the text of the assignment above, whether or not they appear below.

For theory: Hard copy, handed in to Prof or TA by the beginning of class. No extra points for using a text editor. Neatly stapled! Name and UNI attached.

For programming: Hard copy, handed in to Prof or TA by the beginning of class. All code, all testing runs (cut and pasted from console, and/or captured screenshots), all Javadoc if hardcopy Javadoc is required in hardcopy (for this assignment, it is not). Neatly stapled, and separately stapled from the theory! Name and UNI attached.

Also for programming: Soft copy, submitted to the *ASSIGNMENTS* page of Courseworks by the beginning of class, in a tarball created by CUIT Unix tar command, given below. Note that "myUNI" should be replaced with your UNI, and that HW2 will change in further assignments, becoming HM3, etc. *Please* use this convention as it makes the TAs' job much easier:

`tar -cvzf myUNI_HW2.tar.gz whateverMyDirectoryIs`

Also for programming: Include the softcopy of Javadoc html. Name and UNI included as comments in *every* class. The code must compile. Your last electronic submission before deadline is the official one that will be executed if needed. Please note: submissions to dropboxes will ignored!