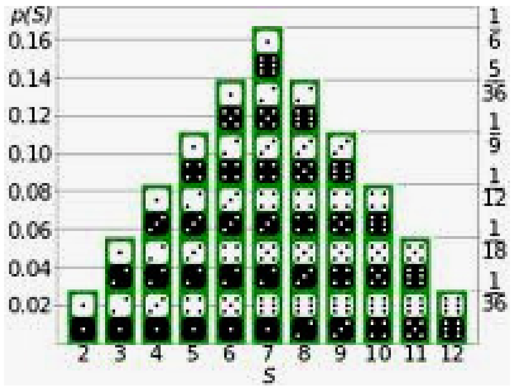
DUE: 3/11/2013

Project #3—Craps Simulation

One of the most popular games of chance is a dice game known as "craps," which is played in casinos and back alleys throughout the world. The rules of the game are straightforward:

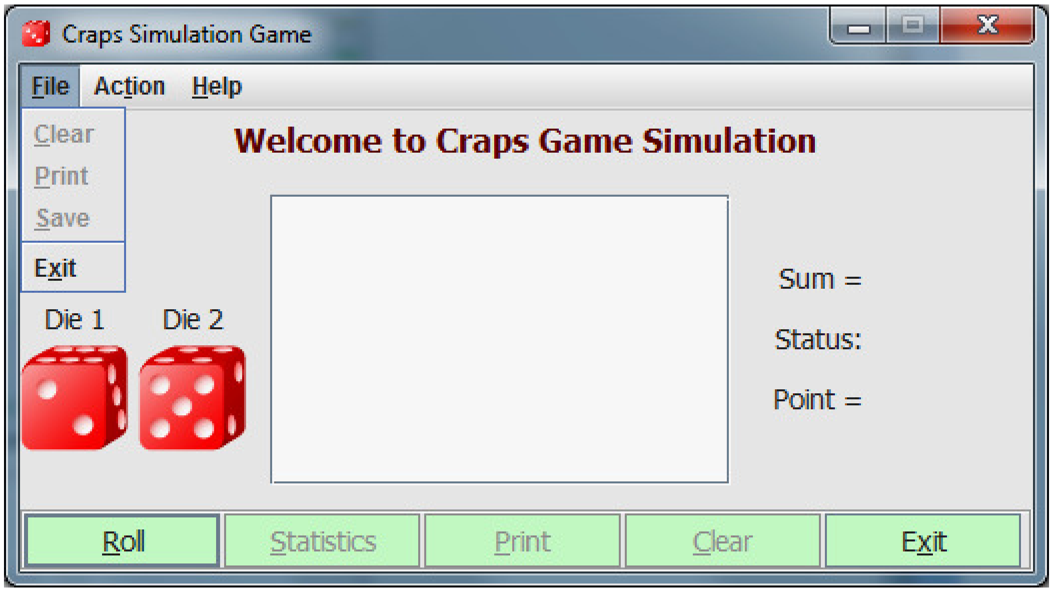
A player rolls two dice. Each dice has six faces. These faces contain one, two, three, four, five, and six spots, respectively. After the dice have come to rest, the sum of the spots on the two upward faces is calculated. If the sum is 7 or 11 on the first throw, the player wins. If the sum is 2, 3 or 12 on the first throw (called "craps"), the player loses (i.e., the "house" wins). If the sum is 4, 5, 6, 8, 9 or 10 on the first throw, that sum becomes the player's "point." To win, you must continue rolling the dice until you "make your point" (i.e., roll your point value). The player loses by rolling a 7 before making the point.



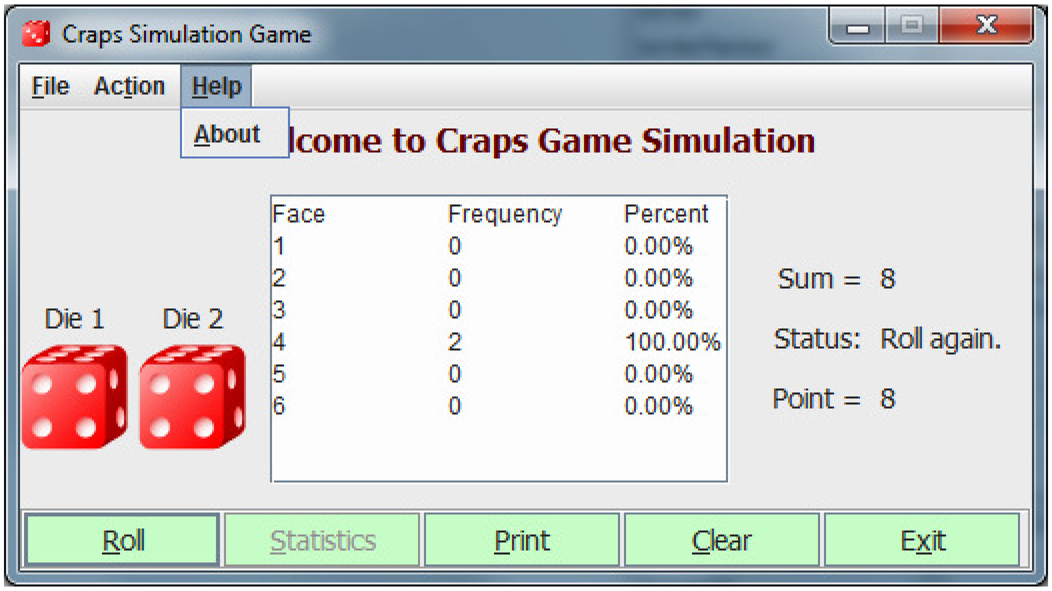
When a pair is tossed, there are 36 different possible outcomes (6 outcomes on the first die, and six outcomes on the second for each outcome of the first). Write a Java GUI-based program (similar to the GUI provided below) that simulates the game of craps.

Screen Captures:

Start-up screen (note disabled buttons and menus):



First roll screen (note that Statistics button and menu choice are disabled because the game is not over yet):



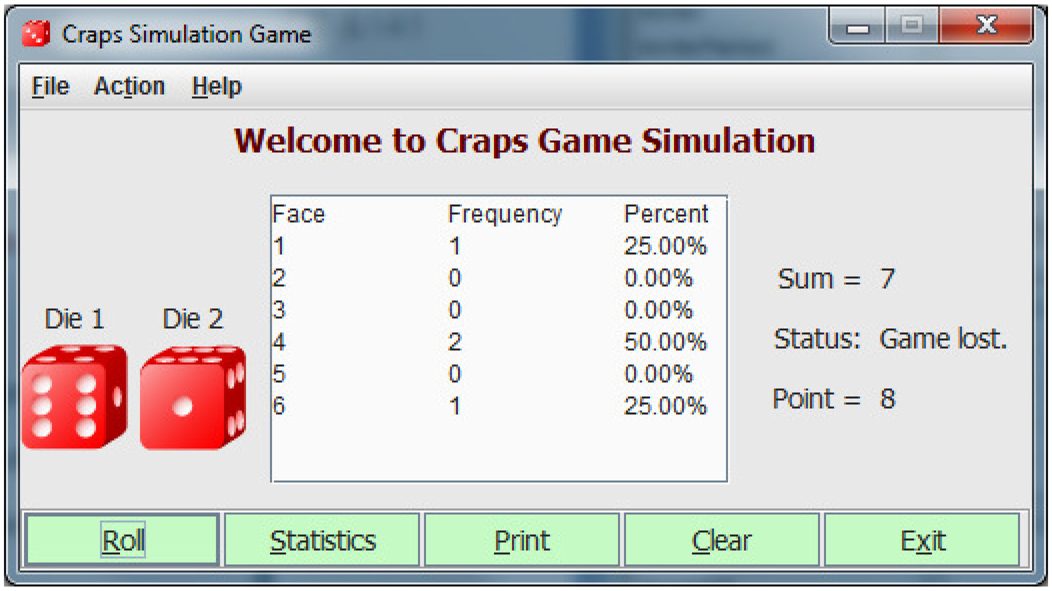
```
Face              Frequency       Percent
1                 0               0.00%
2                 0               0.00%
3                 0               0.00%
4                 2               100.00%
5                 0               0.00%
6                 0               0.00%
```

Sum = 8

Status: Roll again.

Point = 8

Second roll (note correct images of dice rolled is showing up and the Statistics button and menu are enabled):



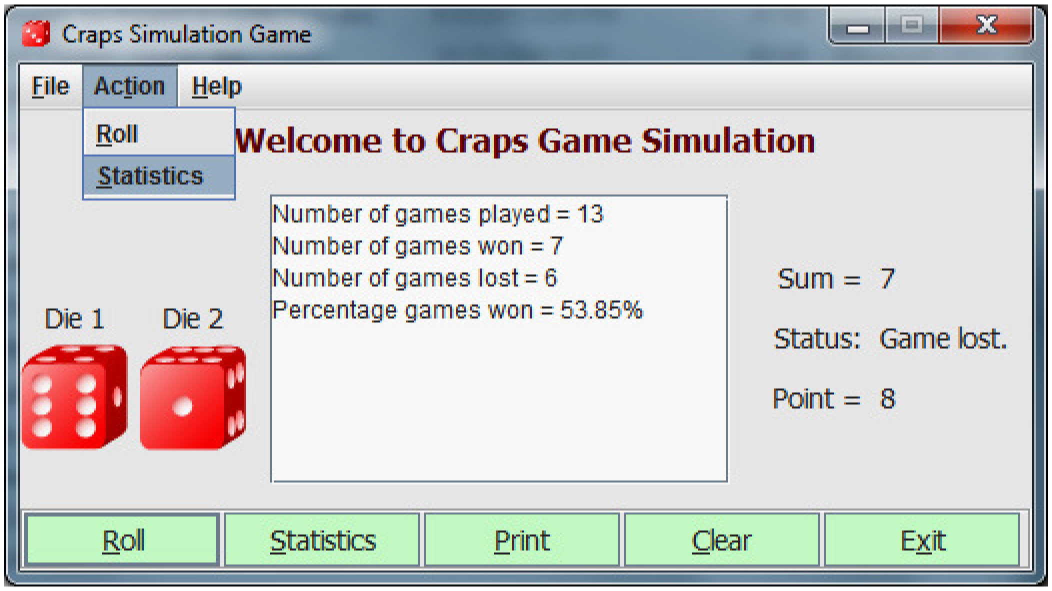
```
Face              Frequency       Percent
1                 1               25.00%
2                 0               0.00%
3                 0               0.00%
4                 2               50.00%
5                 0               0.00%
6                 1               25.00%
```

Sum = 7

Status: Game lost.

Point = 8

Many rolls later with Statistics button pressed (note the display in the JTextArea keeps track of the frequency and percent of each face on the dice)



```
Number of games played = 13
Number of games won = 7
Number of games lost = 6
Percentage games won = 53.85%
```

Required Features: Your Java program should have at least the following features and capabilities:

- ❖ Displays the result of rolling two dice. This includes the images of the two dice rolled, frequencies and percents for each face, the sum of the two dice, game status and point (if any).
- ❖ At least two separate classes: CrapsGUI (driver) and a Die class.
  - Your GUI does not have to look exactly like the one given, but it must provide all the functionality.
  - Two arrays in CrapsGUI to hold the frequencies and percentages for each face of the dice.
  - Design the Die as you see best, but it must have at least an instance variable for the face showing up on it, default and overloaded constructors, a roll method that returns a random integer 1 to 6, representing a die face, and a set and get method for the face of the die.
- ❖ Displays a simulation form (similar to the one shown below), with buttons Roll, Statistics, Print, Clear and Exit. The Roll and the Exit buttons are the only ones enabled initially (and their corresponding menus).
  - The Roll button rolls the two dice and display the appropriate image of the dice outcomes. The frequencies and percents are updated with each click on the Roll button.
  - The Statistics button displays number of games played, won, lost, and the percent of won games. For extra credit, display the error from the theoretical probability of winning (which you need to find) .
  - The Print button prints the content of the JTextArea; the Print menu prints the GUI.

  - The Clear button clears the form and resets the game to start from the beginning.
  - Each button should have a tool tip explaining its function.
- ❖ The form contains JMenus for File, Action and Help with the following menu items.
  - The File Menu should contain Clear, Print, Save and Exit menu items that duplicate the functionality of the buttons. Initially, until at least one roll is performed, only the Exit menu item is enabled. Note however that there is no Save JButton, so the Save menu item should save the results of the game to an external text file. Also the Print menu differs from the Print JButton: it prints the entire GUI and not the content of the JText Area (which is the function of the Print JButton).
  - The Action menu should contain roll and statistics menu items that duplicate the corresponding buttons on the form. The statistics menu item is also disabled initially.
  - The Help menu should contain an About menu item for the About form which should explain the rules of the game, among other features (images, copyright, etc.
- ❖ The form should also contain:
  - A splash screen that shows the logo image and closes itself.
  - A nontrivial About form. Include at least a logo image, instructions for playing the game, credits, version, and copyright information.
  - Same icons for all forms.
- ❖ The main form should also start centered on the screen. It should not be resizable.
- ❖ The code should be well documented with abundant but relevant Javadoc comments.
- ❖ Follows Java naming conventions for the form(s), variables, methods, and constants.

1. The project is worth 30 points and it will be graded on accuracy, completeness, presentation, comments, and compliance with the project requirements.
2. Make sure that your name, project name, date and hours of completion, platform, and due date appear as comments in your code (use Javadocs).
3. Verify that the results of running the simulation are correct.
4. Submit the project zipped with all appropriate NetBeans folders and files via Blackboard.

EXTRA CREDIT (Optional—3 points). Modify the craps program to allow wagering. Initialize variable bankBalance to $1,000 . Prompt the user to enter a wager. Check that wager is less than or equal to bankBalance, and if it not, have the player reenter wager until a valid wager is entered(i.e., verify a valid wager). After a correct is entered, run one game of craps. If the player wins, increase bankBalance by wager and display the new bankBalance. If the player loses, decrease bankBalance by wager , check whether bankBalance has become zero and, if so, display the message "Sorry, you busted!" As the game progresses display various messages to create some "chatter," such as "Oh, you are going for broke, huh?" or "aw c'mon, take a chance!" or You're up big. Now is the time to cash in your chips!". Implement the "chatter" as a separate method that chooses the appropriate message to display.

Project Guidelines

Problem Statement: The following list reviews the GUI design and programming guidelines you have learned so far. You can use this list to verify that the applications and projects you create adhere to the standards outlined herein.

- Information should flow either vertically or horizontally, with the most important information always located in the upper-left corner of the screen.
- Maintain a consistent margin of two or three dots from the edge of the window.
- Related controls should be grouped together using either white space or a group box.
- Position related controls on succeeding dots.
- Buttons should either be centered along the bottom of the screen or stacked in either the upper-right or lower-right corner.
- If the buttons are centered along the bottom of the screen, then each button should be the same height; their widths, however, may vary.
- If the buttons are stacked in either the upper-right or lower-right corner of the screen, then each button should be the same height and the same width.
- Use no more than six buttons on a screen.
- The most commonly used button should be placed first.
- Button captions should:
    - ✓ be meaningful
    - ✓ be from one to three words appear on one line
    - ✓ be entered using book title capitalization
    - ✓ contain & for “hot” keys activation
- Use labels to identify the controls in the interface.
- Identifying labels should:
    - ✓ be from one to three words
    - ✓ appear on one line
    - ✓ be aligned by their left borders
    - ✓ be positioned either above or to the left of the control that they identify
    - ✓ end with a colon (:)
    - ✓ be entered using sentence capitalization
- Align labels and controls to minimize the number of different margins.
- If you use a graphic in the interface, use a small one and place it in a location that will not distract the user.

- Use no more than two different font sizes, which should be 10, 12 or 14 points.
- Use only one font type, which should be a sans serif font, in the interface.
- Avoid using italics and underlining.
- Use light colors such as white, off-white, light gray, pale blue, or pale yellow for an application's background, and very dark color such as black for the text. Alternately, use dark colors for the form's background with light colors for the text—contrast is the name of the game.
- Use color sparingly and don't use it as the only means of identification for an element in the interface.
- Set each control's nextFocusableComponent property to a number that represents the order in which you want that control to receive the focus (begin with 0).
- Assign a unique access key to each essential element of the interface (text boxes, buttons, and so on).
- Document the program internally using Javadocs.
- Use variables to control the preciseness of numbers in the interface.
- Test the application with both valid and invalid data (test the application without entering any data; also test it by entering letters where numbers are expected). Test the application for correct results with known data.
- Center a form by including the appropriate formulas in the form's Load event.
- Display an hourglass or an arrow and an hourglass if a procedure will take a long period of time.
- Remove excess or duplicated code from the application.

Grading Guidelines: Your project will be graded on the following:

- Correct solution to the proposed problem.
- GUI design (as outlined above).
- Elegance, simplicity, style, and readability of the algorithms and code.
- Comments:
    - ✓ Use of header.
    - ✓ Description of project.
    - ✓ Description of procedures.
    - ✓ Description of complicated code.

Hints for Project 3:

1. Make sure your project has at least two classes: CrapsGUI, a JFrame for a driver, and a Die class for the 2 dice instantiated and used by the driver.
2. The required About form is another JFrame class. You could use the -splash switch for the Java VM to set the splash screen or you can use the Splash class provided in one of our labs.
3. The Die class should represent the essential features and functionality of a single six-sided die. It should contain at least the following:

  a. A variable to indicate the face on the die (int type is allowed for simplicity, but it should be a byte).
  b. Two constructors, the default constructor and an overloaded constructor with a parameter to set the face value of the die it instantiates .
  c. A roll method that rolls the die and returns a random number for the result of the roll (1-6).
  d. A getSide(0 and setSide() methods to get and set the face value of the coin.
  e. A toString() method that overrides the Object's toString(). It might not be used in this application, but it should a String version of the roll() method with a message perhaps.

4. Aside the obvious instance variables to be included in the CrapsGUI class (total number of games, number of games won and lost, sum of the two dice, point, etc.) consider including the following additional instance variables and constants in the CrapsGUI:
   a. Two arrays: one for the frequencies of the faces showing up and one for the corresponding percentages.
   b. A boolean variable to keep track if the game is played for the first time.
   c. Three constants to indicate the state of the game: WON, LOST and CONTINUE.
   d. Two instances of the Die class initialize to some random faces.
5. Aside from the event handlers for the JButtons and JMenuItems, consider implementing the following methods in the CrapsGUI class:
   a. An enableJButtons() method with a boolean parameter to control which buttons and menu items are enabled and disabled as the game progresses.
   b. A setPercent() method to calculate the percentages and assign them to the percent array .
   c. A rollDice() method that rolls the two dice and returns the sum of the two dice.
   d. An optional (for the extra credit) checkBalance() method to check if the bankBalance is zero.
   e. A setLabel() method to set the correct image of the die rolled on a label that contains the die image.
   f. A displayResults() method to display the frequencies and percentages of each die rolled. It should also display the status of the game with directions of what to do next.
   g. An optional (for the extra credit) chatter() method to encourage/discourage the player (or just to chatter).
6. Implement all of the event handlers for all JButtons and JMenuItems. By far the most crucial is the rollJButtonActionPerformed() event handler. In it you should:
   a. Keep track of how many dice are rolled--note that with each roll you toss two dice, so increment the total accordingly.

   b. Roll the two dice, assign sum and images of the rolled dice to the labels that hold them and perform the algorithm for playing the game.
   c. This algorithm should check if it is the first roll in the game. If so, it should update assign appropriate values to the variables in question if the game is won (7 or 11), lost (2, 3, or 12), or if it is to continue. Consider using a switch statement on the sum of the two dice for this part.
   d. If, on the other hand, the game is not played for the first time, you need to check if the point is matched (in which case player wins) or if 7 is rolled (in which case the player loses)
   e. Regardless of the outcome of the roll, the frequency and percentage arrays need to be undated in this handler, the results of the roll should be displayed in the JTextArea, and the visibility of the controls and menus should be set.
   f. An optional (for the extra credit) chatter() method to encourage/discourage the player (or just to chatter).

7. Make sure you set the rollJButton as default.
8. Include images of the two dice rolled with each click on the Roll JButton.
9. Save the results of playing the game to an external file. Include in it the statistics result plus bank balance (if extra credit is implemented).
10. Prevent the form from maximizing. Center it on the screen as it starts.
11. Provide a ToolTip with appropriate messages for specified controls and menus.
12. Declare all constants--no Houdini numbers!
13. Provide icons for all forms.
14. Create and external text file, DiceSimulation.txt, for saving the dice game results.
15. Finally, pay attention to these comments (used as a grading rubric):

```
/'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'Comments by the prof:
'Great effort.  Here are suggestions for improving:
'1. Use Javadocs comments through the program, not just at the top heading
'Use comments to describe the class each method and complicated code.
'2. Missing Die class with appropriate constructors and methods.
'3. Make the roll button default.
'4. Correct the results from playing the game--check your game algorithm.
'5. It is good style to tab code inside a method.
'6. Use final definition for all constants, including game status.
'7. Provide functional menus with specified menu items.
'8. Provide Roll, Statistics, Clear, Print and Quit buttons.
'9. Set enabled/disabled properties of the buttons and menus correctly.
'10. Not displaying correct images of dice rolled.
'11. Statistics missing or incorrect.
'12. Clear should reset the form and reset all variables.
'13. Need two separate print methods: Print JButton should print the content of
JTextArea; the Print menu item should print the form.
'14. Follow the Java naming convention for naming variables.
'15. Avoid excessive class variables-declare local variables inside methods.
'16. Clean up the empty methods code.
'17. Need to save the results of playing the game to an external file.
'18. Cumulate results of games played.
'19. Missing frequency and percent arrays.
'20. Implement a separate setPercent() method for percent calculations.
```

```
'21. The setDice() method should use the setSide() methods for each die.
'22. Not keeping the point value correctly.
'23. Center the form as it loads.
'24. Use monospaced font in JTextArea (or JListBox) such as Courier New--it helps
line-up columns correctly.
'25. Missing About form.
'26. Missing Splash screen.
'27. Provide ToolTip.
'28. Provide icons for all form.
'29. Use boolean variable firstRoll, initially set to true.
'30. Prevent form from resizing or maximizing.
'
'The ones that apply to your project are:
'1, 25, 30
'27+1=28/30
'
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```