# EECS285, Fall 2013

# Programming Project 3 Specification

## Overview:

For this project, you will implement a version of the popular game show “Wheel Of Fortune”. Details of the game play will follow.

This project is due ***Monday, November 11, 2013 no later than 2:59pm***. Use the same submission process as was provided for project 2, and be sure to use packages in the same way you did for project 2, with the package name being "eecs285.proj3.uniqname" where uniqname is replaced with your UM uniqname.

## How The Game Works

Wheel Of Fortune has been a popular televised game show for over 30 years. The game centers around a word puzzle that the players have to solve. Initially, the puzzle is presented a series of blank spots, one per letter of the puzzle. Punctuation, such as spaces, commas, etc., are displayed. Players fill in the puzzle by one of two ways. For consonants, players spin a large wheel made up of spaces of varying monetary amounts, along with some “bad” spaces, such as “Lose a Turn” and “Bankrupt”. If the spin results in a monetary amount, the player guesses a letter. If the guessed letter exists in the puzzle, all instances of the letter are displayed, and the player is awarded an amount of money equal to the number of instances of the letter times the monetary amount resulting from the wheel spin. For vowels, the player does NOT spin the wheel. Instead, the player announces they wish to “buy a vowel”. A set amount (typically $250, which should be what your program uses, but make this a named constant so its easy to change if the rules change) is deducted from their account and the player selects a vowel (vowels can’t be bought if the player has less than the required cost of a vowel). If the vowel is in the puzzle, all instances are illuminated. No money is awarded for illuminating vowels, and the cost of buying a vowel is the same regardless of the number of instanced of the vowel illuminated. Regardless of consonant or vowel, if the letter chosen by the player is in the puzzle, the player’s turn continues, otherwise, play moves to the next player.

If the player spins the wheel and lands on the “Lose A Turn” space, the player is not able to guess a letter, and play advances to the next player. If the player’s spins lands on “Bankrupt”, any money the player has accumulated is lost (their accumulated winnings are set to 0), and play advances to the next player.

At any time, the active player may choose to solve the puzzle rather than spinning or buying a vowel. The player is given the chance to say what they believe the full puzzle is. If correct, the player wins the full amount of money they have accumulated, and all other players win nothing. If the guess is incorrect, the player loses their turn, and play advances to the next player. Players must be careful, though, because once they opt to try to solve the puzzle, they must make a guess – they may not “change their mind” and spin the wheel instead.

## Details

Your implementation of the popular game show will use a graphical user interface written using Swing components in Java. The game will be implemented as a Java application (as opposed to an Applet). When a new game is first started, the player(s) will be prompted to enter an integer seed for the random number

generator (if you were to publish the software, you would get a seed in a different manner, such as from the system clock). If the user enters something other than the requested int type, they will be re-prompted until valid input is found. Next, the player(s) will be prompted to enter names of the players. While the real game show is played with three players, your implementation will allow for any number of players greater or equal to 1 (playing with only 1 player isn't much fun though). Next, someone (presumably someone not currently playing the game) is prompted to enter the puzzle for this round. After this, the game is displayed, and play commences as described.

To make the game a little more "realistic", your implementation will display an image of the wheel space resulting from a spin. The wheel is generally made up of 24 equally sized spaces (that is a spin result is determined by a uniform random draw of 24 possibilities). The images for the board spaces will be provided to you, and the names of the images are in the format "<spaceNumber>_<spaceLabel>.jpg". For example, if space number 5 is a $700 space, the image will be named "5_700.jpg".

Following is a list of other aspects to the real Wheel Of Fortune that you *need not worry about* for this program implementation:

- Free spins: In the TV show, there are certain spaces which contain free spin tokens, which players keep and can use later to continue a turn that would otherwise have ended. You don't need to worry about free spins in your implementation.
- Indicating when only vowels remain in the puzzle: In the TV show, when the puzzle has all the consonants displayed, and only vowels remain, a buzzer goes off and Pat (the host) announces that only vowels remain, and people don't spin the wheel any more – they only buy vowels or guess the puzzle. This does not need to be supported in your implementation – the players can continue to play until someone correctly guess the puzzle.
- Multiple rounds: When someone wins the round on TV, another round is played with a new puzzle with a different first player. This continues until time runs out. For your program, once someone wins a round, the game ends. (Note: you could simulate multiple rounds by running the program multiple times with a different ordering of players).
- Bonus games: Some puzzles are "clues" that lead to the winning player being able to guess what the clue is indicating for additional winnings (for example, the puzzle might be "Prestigious University In Ann Arbor", and the winning player would win a couple thousand dollars if they came up with "University of Michigan" after correctly solving the puzzle. Your simplified version of Wheel of Fortune does not support bonus games.
- Special wheel spaces: In recent seasons of Wheel of Fortune, there are special spaces in the wheel (like a tiny $10,000 space smashed in between two tiny bankrupts). There are also "Jackpot rounds" where if the player lands on a Jackpot space and solves the puzzle immediately they win a Jackpot – none of this is required for your program.

In other words, you are to implement a very basic version of a single round of Wheel Of Fortune. If you'd like to support any of these things, or other features, you can certainly add to your program after submitting the project.

## Requirements

A runnable version of my implementation of the program will be provided. Your program should mimic the program as closely as possible. If your program looks and works exactly like mine, then you will get a full grade for correctness. If your program looks slightly different, but acts the same way, you will get full credit for correctness as well. However, if your program is missing any features, points will be taken off. This includes major things, such as allowing the user to buy a vowel when they don't have enough money, to minor things such as not clearly indicating which player is the active player as play progresses. If you want to make changes to make the interface look nicer, you can, but don't change the general functionality of the program. If your interface looks less nice, or is difficult to use, points will be deducted. Since this program is mostly about GUI development, make sure your GUI looks nice and is easy to use.

Note: If "issues" are discovered in the provided program, they will be fixed as soon as possible and the program will be reposted.

## Dealing With the Images

Your images must be in an "images" subdirectory in the same directory as your .java source files. If using Eclipse, its easiest to just "drag-and-drop" the images folder (the whole folder, not the files in the folder!) to the package for this project in Eclipse. If using Linux command line, just make a folder in the same directory as your .java source files called images. To support easily updating wheel spaces to allow them to change from round to round of the game (would require multiple executions of this project), we will read in the images with the specifically expected names described above and set up each space based on the filename of the image. In other words, "2_800.jpg" will represent space number 2 and a money value of 800 dollars. Rather than hardcode that space 2 is an 800 dollar space, parse the filename to determine this. You need to make sure this works as expected, because during testing, we may change the names of the images and your program is expected to handle things appropriately.

To avoid confusion, I will provide a bit of code that I used in my solution that you can feel free to use in yours if you'd like. See the text file attached to the ctools project page for this support code. Even if you choose to write your own version of this code, you should use the same "getClass().getClassLoader().getResource(…)" line that is in there to find the images to ensure grading goes as smoothly as possible.

It is ok to assume there are 24 spaces on the wheel for the purposes of this project, although using a linked list or similar data structure would make it pretty simple to read in however many images were found in the directory. You do NOT need to worry about this for this project – feel free to use an array 24 elements long for storing wheel spaces.