Programming Assignment 7 (100 Points)

Due: 11:59pm Thursday, November 10th

README (10 points)

You are required to provide a text file named **README,** NOT Readme.txt, README.pdf, or README.doc, with your assignment in your pa7 directory. There should be no file extension after the file name "**README**". Your README should include the following sections:

Program Descriptions (5 points) : Provide a high level description of what each of your programs do and how you can interact with them. Make this explanation such that your grandpa or uncle or someone you know who has no programming experience can understand what these programs do and how to use them. Do not assume your reader is a computer science major.

Short Response (5 points) : Answer the following questions:

Vim Questions:

1. In vim/gvim, what commands will indent N consecutive lines (starting from the cursor line) by one level where the indent level is defined to be two spaces? This will take two vim commands: one to set the number of spaces to indent with each indent level (default is 8), and one to actually indent N consecutive lines. Likewise what command will shift N lines left (de-indent N lines)?
2. In vim/gvim, what command will indent an entire curly-bracket block one level, while the cursor is currently on either the open or close curly bracket of the block? Likewise what command will shift an entire curly-bracket block one level left (de-indent block)?
3. How do you open a new line below and insert (one keyboard key for both steps)?

Unix Question:

4. On the Unix command line, how can you capture (redirect) the program's output into a file named "output"?

Java Question:

5. How can you create an array of ints in Java and initialize it with the values of all single digit odd positive numbers (inclusively between 0-9), all in one step/line?

STYLE (20 points)

Please see preview programming assignments; especially PA2's writeup. You will be specifically graded on commenting, file headers, class and method headers, meaningful variable names, judicious use of blank lines, not having more than 80 characters on a line, perfect indentation, no magic numbers/hard-coded numbers other than -1, 0, +1, using setters and getters, etc (**see the style section of PA2's writeup**).

You will lose significant points if you have static methods or static variables for the Spin100 program (except, constants are okay). Instead you should create getters and setters to change the variables for an object.

CORRECTNESS (70 points)

Start by creating your pa7 directory as usual. Then copy over the starter code from the public folder by doing the following:

```
$ cp ~/../public/PA7indrome.java ~/pa7
$ cp ~/../public/PA7-images/* ~/pa7
```

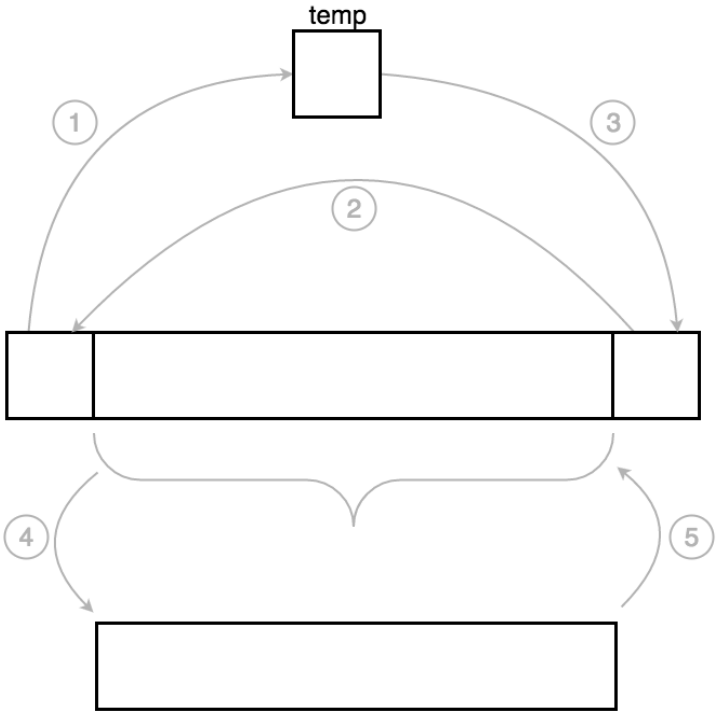
Program 1: PA7indrome (20 Points)

You must use recursion and implement the methods listed below or you will lose all 20 points for this program. Also, do not edit the main() method in PA7indrome.java as it will negatively affect your grade.

Write a command line application that checks whether an array of strings is a palindrome. A palindrome is a word, phrase, number, or other sequence of characters which reads the same forwards and backwards. The program will take in a file as a command line argument. Each line of the file will be converted into a string, where each string will be placed as an element in an arraylist then converted to an array (this part has already been written for you in the main() method--do not edit anything in this method). Your task will be to then determine if the *array* of strings is a palindrome or not (non-case sensitive). You will be using two recursive methods to implement this functionality.

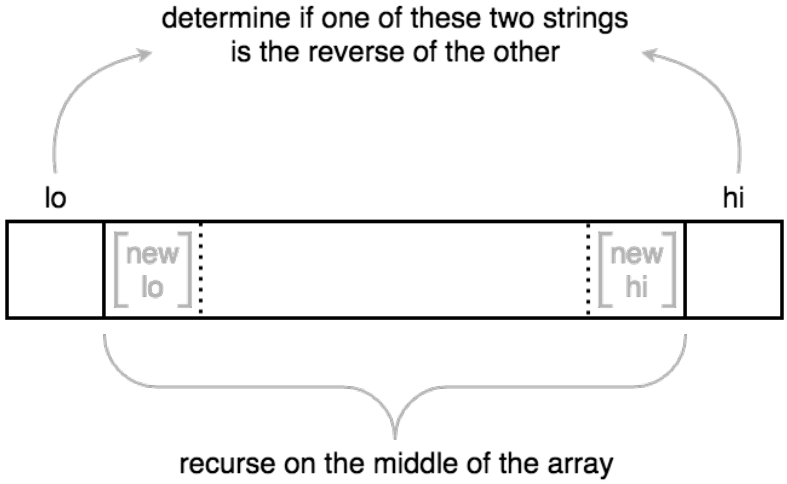
`public static char[] reverse( char[] toRev ) {...}`

- Overall goal: reverse the array of characters passed in using recursion.
- If the length of the character array passed in is 1, just return the array right away.
- Otherwise, swap the first and last characters in the array (#1-3 below)
- Copy the middle of the array into a new array and recursively call reverse on this new array (#4 below).
- Place the returned array back into the middle of the original array of characters, and return this reversed array of characters (#5 below).
- Hint: you can use System.arraycopy() (look up the javadocs).
- **This method must use ends-and-middle recursion.**



`public static boolean isPalindrome( String[] words, int lo, int hi ) {...}`

- Overall goal: determine if the array of strings passed in is a palindrome using recursion.
- Determine if the string at the low index in the words array is the reverse of the string at the high index in the words array. To do so, you **must** use your reverse() function, after converting the string to a char array and passing it in. This comparison should be **non-case sensitive**.
- If the string at the low index is indeed the reverse of the string at the high index, recurse on the remaining middle portion of the array by passing modified values of the low and high index in the recursive call. Otherwise, return false.
- **This method must use ends-and-middle recursion.**



You must use these methods as they are defined above or you will lose points.
We will test for edge cases when grading this assignment, so it is up to you to test your program thoroughly.

Similar to previous assignments, your output MUST match exactly to our sample output!

Sample Output 1: Not a palindrome (user input in **BOLD**)
For this example, the input file is named `notPalindrome`. If you open the file in vim, it should look like this:

```
start
early
start
often
```

The output from running PA7indrome will give you the following:

```
[cs11fxyz@ieng6-201]:pa7$ java PA7indrome notPalindrome
Input array of strings:
start, early, start, often

The file is not a palindrome.

[cs11fxyz@ieng6-201]:pa7$
```

Sample Output 2: Odd-length palindrome (user input in **BOLD**)

Input file named `oddPalindrome`:

```
StArT
EaRlY
sTaRt
oFtEn
ufotofu
NeTfO
TrAtS
yLrAe
tRaTs
```

The output from running PA7indrome will give you the following:

```
[cs11fxyz@ieng6-201]:pa7$ java PA7indrome oddPalindrome
Input array of strings:
StArT, EaRlY, sTaRt, oFtEn, ufotofu, NeTfO, TrAtS, yLrAe, tRaTs

The file is a palindrome.

[cs11fxyz@ieng6-201]:pa7$
```

The sequence of characters in the array of Strings generated from the input file reads the same forwards and backwards (non-case sensitive), so it is a palindrome. Note that because the length of the array is odd, the middle string must also be a palindrome (in this case the middle string is “ufotofu”, which is a palindrome).

Sample Output 3: Even-length palindrome (user input in **BOLD**)

Input file named `evenPalindrome`:

```
StArT
EaRlY
sTaRt
oFtEn
NeTfO
TrAtS
yLrAe
tRaTs
```

The output from running PA7indrome will give you the following:

```
[cs11fxyz@ieng6-201]:pa7$ java PA7indrome evenPalindrome
Input array of strings:
StArT, EaRlY, sTaRt, oFtEn, NeTfO, TrAtS, yLrAe, tRaTs

The file is a palindrome.

[cs11fxyz@ieng6-201]:pa7$
```

Sample Output 4: Palindrome with new lines (user input in **BOLD**)
Input file named newLinesPalindrome. **Note**: New lines will be cut off (ignored) by the scanner.

```
StArT
EaRlY

sTaRt

oFtEn
NeTfO
TrAtS
yLrAe
tRaTs
```

The output from running PA7indrome will give you the following:

```
[cs11fxyz@ieng6-201]:pa7$ java PA7indrome newLinesPalindrome
Input array of strings:
StArT, EaRlY, sTaRt, oFtEn, NeTfO, TrAtS, yLrAe, tRaTs

The file is a palindrome.

[cs11fxyz@ieng6-201]:pa7$
```

Program 2: Spin100 (50 Points)

You are to design a game that models itself after the mini game found in the Price is Right.
https://www.youtube.com/watch?v=mFJCfNib2ns

Main idea of the game:

- There are two players: Player 1 and Player 2. Player 1 gets to spin the wheel first. Player 1 gets to spin until they have reached over 100 points or have clicked the "Finish Player 1" button. At that point, it becomes Player 2's turn. Player 2 then gets to spin until they have reached over 100 points or have clicked the "Finish Player 2" button.
- Once Player 2 goes over 100 points or clicks the "Finish Player 2" button, the game ends.
- Whichever player is closer to 100 points wins. Once a player goes over 100 points then they've lost. If both players go over 100 points then it's a tie. If both players have the same score, then it's a tie.

Example scores and their outcomes:

Player 1	70	120	20	100	100	95
Player 2	90	110	120	100	105	100
Winner	Player 2	Tie	Player 1	Tie	Player 1	Player 2

Implementation Details:
This program will have two Java source files: **Spin100Controller.java** and **Spin100Wheel.java**. **Spin100Controller.java** will be the main controller which extends WindowController and handles the GUI layout. **Spin100Wheel.java** will be the actual wheel object that defines what a wheel can do, how it spins, etc.

1. Setting up the GUI:
Start by first getting the GUI set up in the begin() method of Spin100Controller.java.

The top panel will have 3 labels:

- "Spin100" - You can customize the font of this label--don't worry about whether the font is the default or not! Just make sure your program doesn't throw errors or have an incorrect layout / text upon start.
- "Player 1's score: " - followed by player 1's score (initially 0).
- "Player 2's score: " - followed by player 2's score (initially 0).
    - Note the space following the ':' for each player's score

The bottom panel will have 5 buttons: (make sure the text matches exactly).

- "Click to Spin P1"
- "Finish Player 1"
- "Restart"
- "Click to Spin P2"
- "Finish Player 2"

The wheels on the center canvas:

- The center of the window will have the canvas, similar to PA6. The width and height of the canvas should be 840 x 660. Note that the **canvas will not be resized** during testing. On the canvas we will be displaying two wheels (one for each player). In order to create the wheels, we need to initialize an array of Image(s), using the images you copied over from the public folder. **Make sure your images are directly inside your pa7 directory, and not in a subdirectory called PA7-images.**

- You will want to load the 20 images into an array of Images, where image width = 185 and image height = 99. To ensure you arrange the images in the correct order, we suggest creating the following array. This will ensure you load the images in the correct order and will also be helpful for scoring later on. Load the images corresponding to these score values in the order in which they appear in the array. The array of images and the array of score values will be parallel arrays.

```
private static final int[] SCORES = { 50, 85, 30, 65, 10, 45, 70, 25, 90, 5,
                                      100, 15, 80, 35, 60, 20, 40, 75, 55, 95 };
```

- The wheel should always start with 50 in the center and should spin in a downward direction. This means that when the wheel spins for the first time, the red arrow will first point to 50, then 85, then 30, and so on.

- You will want to create each wheel by calling the Spin100Wheel constructor. Some parameters you will probably want to pass to the constructor are:
    - The array of Image(s) corresponding to the SCORES values.
    - Location of the wheel (location of the top number). The center of the left wheel should be horizontally aligned with the center of the left half of the canvas. In other words, you need to find the x-coordinate that will center the left wheel in the center of the left half of the canvas. Similarly, you need to find the x-coordinate that will center the right wheel in the center of the right half of the canvas. The y-coordinate of the top number for both wheels should be 10 (make sure the layout of your wheels matches the screenshots exactly).
    - Reference to the Spin100Controller so that controller methods can be called from the wheel (this will be useful for setting the score after the wheel stops spinning each time)
    - Reference to the drawing canvas so the wheel can be placed on the canvas

The red arrows on the center canvas:

- The arrows are red-filled arcs with start angle = -15, arc angle = 30, height = 150, and width = 150.
- The red arrow positions are translations of the Location of the upper left-hand corner of the wheel they point to – so the location of the arrow = <location of its wheel> translated by (100, 176).

Messages declaring winner on the center canvas:

- Create a hidden text object saying "Tie" 10 pixels to the left of the center of the canvas, and 5 pixels down from the top of the canvas. The text should be blue, bold, and size 16.
- Create a hidden text object saying "P1 Winner" at position (20, 5). The text should be green, bold, and size 16.
- Create a hidden text object saying "P2 Winner" 100 pixels from the right edge of the canvas, and 5 pixels down from the top of the canvas. The text should be green, bold, and size 16.

See the screenshot section below to see what all of this looks like.

2. Wheel Functionality

Because the wheels are animated, the Spin100Wheel class needs to extend ActiveObject (and therefore must contain a run() method).

Creating a wheel:

- Each wheel needs an array of 5 VisibleImages (generated from the array of images passed in as a parameter to the constructor) to store the 5 images that will be visible on the canvas at any given time.
- The location passed to the constructor is the location of the topmost VisibleImage. The next VisibleImage will be 99 pixels below that, and the next 99 pixels below that, and so on.
- Each wheel needs a random double generator object to generate random numbers for the amount of ticks to spin when the spin button is clicked (see more details below in the Spinning the wheel section).
- Don't forget to save a reference to the Spin100Controller object to help with scoring.
- Set the number of ticks to 0, so the wheel begins stationary.
- As with all ActiveObjects, the last statement in the constructor should be `start();`.

Spinning the wheel:

- Initially, the wheel's number of ticks should be set to zero. When a wheel's spin button is clicked, the controller should update the number of ticks so the wheel will start spinning. This means you will want a method to generate a random number of ticks for the wheel to spin.

- One way to implement this is to have a method called **getRandomTicks()**. This method should:
    - First generate a random double between 0 and 1
    - Based on the random number, return the minimum number of ticks (50) + a number of additional ticks, which indicates how many ticks a wheel should spin for. This is to allow the wheel to stop on different numbers each time by randomizing where a wheel stops cycling through the array of images.
    - If the random number is between 0 and 0.05, return MIN_TICKS + 1. If it's between .05 and .1, return MIN_TICKS + 2. You should keep adding one more tick until you return MIN_TICKS + 13 for when the value is between .6 and .65. From there you should subtract one less tick until you return MIN_TICKS + 6 when the value is between .95 and 1.
        - Range of ticks to add (over the .05 increments) is
          (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 12, 11, 10, 9, 8, 7, 6)

- After getting the random number of ticks, the controller needs to set the wheel's number of ticks using a setter method.

- To make the wheel actually spin, the run method will need to constantly check its number of ticks. If the number of ticks is greater than 0, increment the index of each visible image displayed to the next index in the images array; in other words, for every tick, shift the images down one. Once you reach the last index of the images array, wrap back around to the beginning (**Hint: think mod operator**). Then decrement the number of ticks left for this wheel to spin and pause to slow down the animation.

- At the beginning of each spin, the initial delay should be set to 1. Then when the wheel delay is less than 20, the wheel delay should increment by 1 on every iteration of the forever loop. When the wheel delay is greater than or equal to 20 and less than 200, the wheel delay should increment by 5. When the wheel delay is greater than or equal to 200 and less than 500, the wheel delay should increment by 20. This will give the illusion of a faster spin at the start which eventually slows down and stops once the number of ticks is zero.

- Once the number of ticks is 0, the wheel needs to call a public method in controller to let the controller know that the wheel's score has changed and that the Text object for the score needs to be updated.

- Some additional methods to include in the class would be setters and getters for the ticks, delay, and score. This is so that the controller can call these methods and change the wheel's variables' values when something like the spin, finish, or reset button is pressed.

3. Controller Functionality

You will implement Spin100Controller.java to handle creating the buttons and listening to actions performed on the buttons.

- Upon start, only the "Click to Spin P1" and "Restart" buttons should be enabled. So in your **begin()** method, in addition to setting up the layouts of the buttons, you should also disable the "Finish Player 1" button and all buttons associated with Player 2.

- Once you have clicked "Click to Spin P1", *all* buttons are disabled until the wheel stops spinning. Once the wheel stops spinning, update Player 1's score. Now that you have spun Player 1's wheel at least once, the "Finish Player 1" button should be enabled.

- As long as Player 1's score is less than or equal to 100, Player 1 can choose to continue spinning by clicking "Click to Spin P1" again, or click "Finish Player 1" to finish their turn. Once Player 1's score is over 100, the controller will automatically finish the turn for Player 1.

- Once Player 1 has finished, all of Player 1's buttons are disabled and the "Click to Spin P2" button is enabled for Player 2 to start spinning. The spinning process follows the same condition as stated for Player 1.

- Once Player 2 has also finished, *all* but the Restart button should be disabled. A game over message should be displayed. The message could announce a Tie, or announce that either Player 1 or Player 2 won (see screenshots for detail).

Summary of the buttons

- The Spin buttons: The spin button disables itself when:
  - The wheels are spinning
  - The player's score has passed 100
  - The player has clicked finish after at least one spin
- The Finish buttons: Each player's finish button is disabled until they have spun at least once. It is also disabled when the wheel is spinning, as well as after they have finished their turn.
- The Restart button: This button is always enabled except when a wheel is spinning. Once clicked:
  - It changes the scores back to 0
  - It resets the wheels so that 50 is at the center again
  - The "Click To Spin P1" button is enabled again and the buttons are reset to the original state.

Implementation Requirements for Button Event Handling

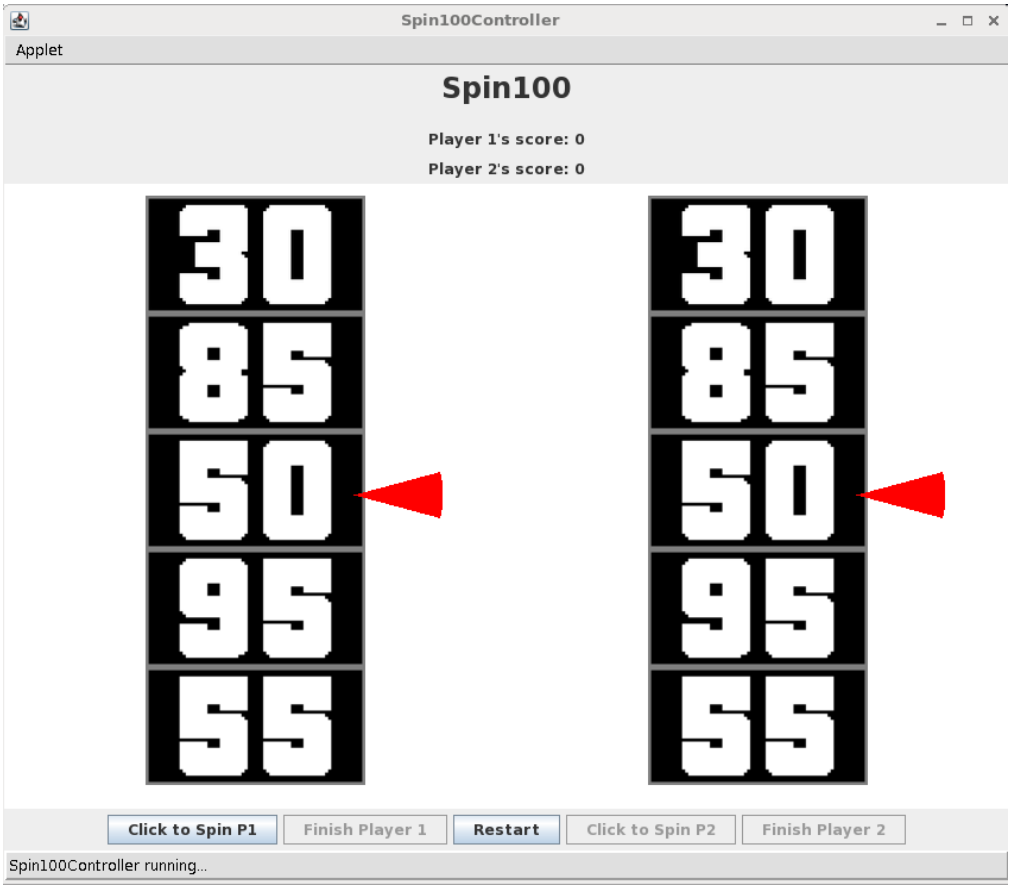
To manage the behaviors of the 5 buttons, you must define 5 private member inner classes *inside* the Spin100Controller class. Use your notes from class as a reference.

- Each private class must implement the ActionListener interface.
- The ActionListener interface enforces that each class has the actionPerformed(ActionEvent e) method declared and implemented.

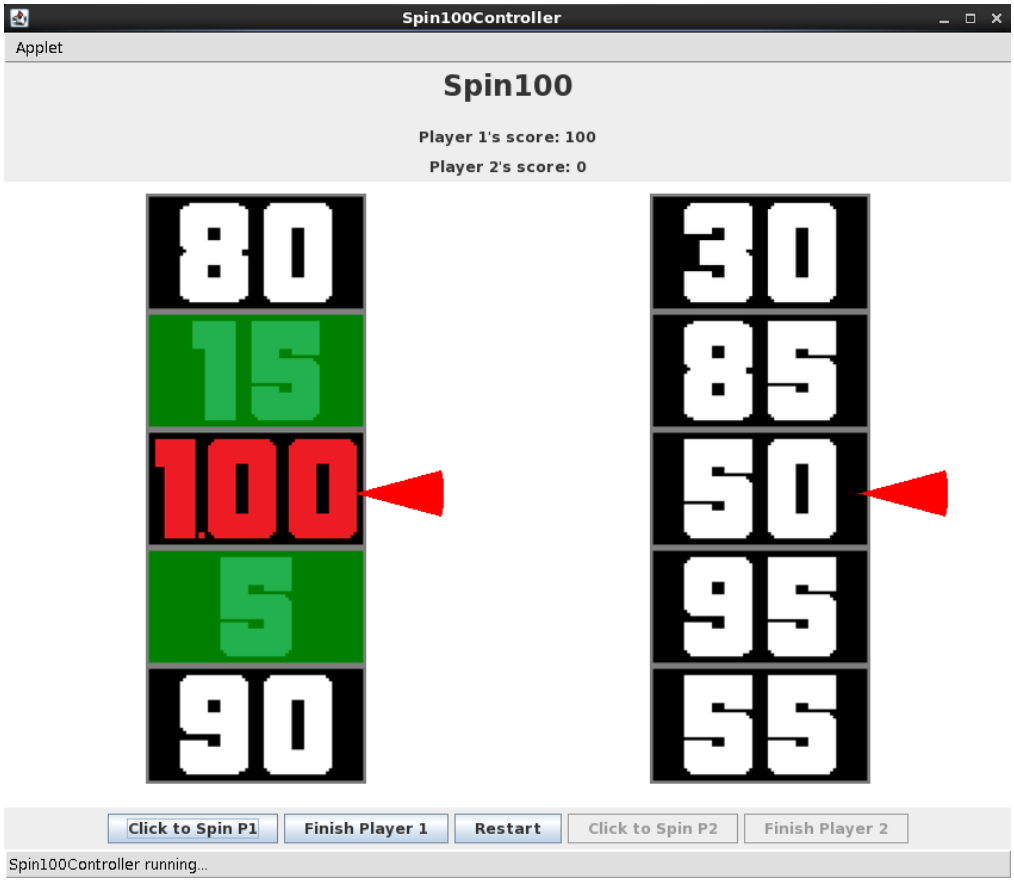
Having separate inner classes for each button is much better than handling *all* button events in one actionPerformed() method. This is because there is a one-to-one relationship between buttons and listeners, resulting in cleaner code that is less prone to errors.

Note: Remember to use getters/setters vs. directly accessing variables by name.

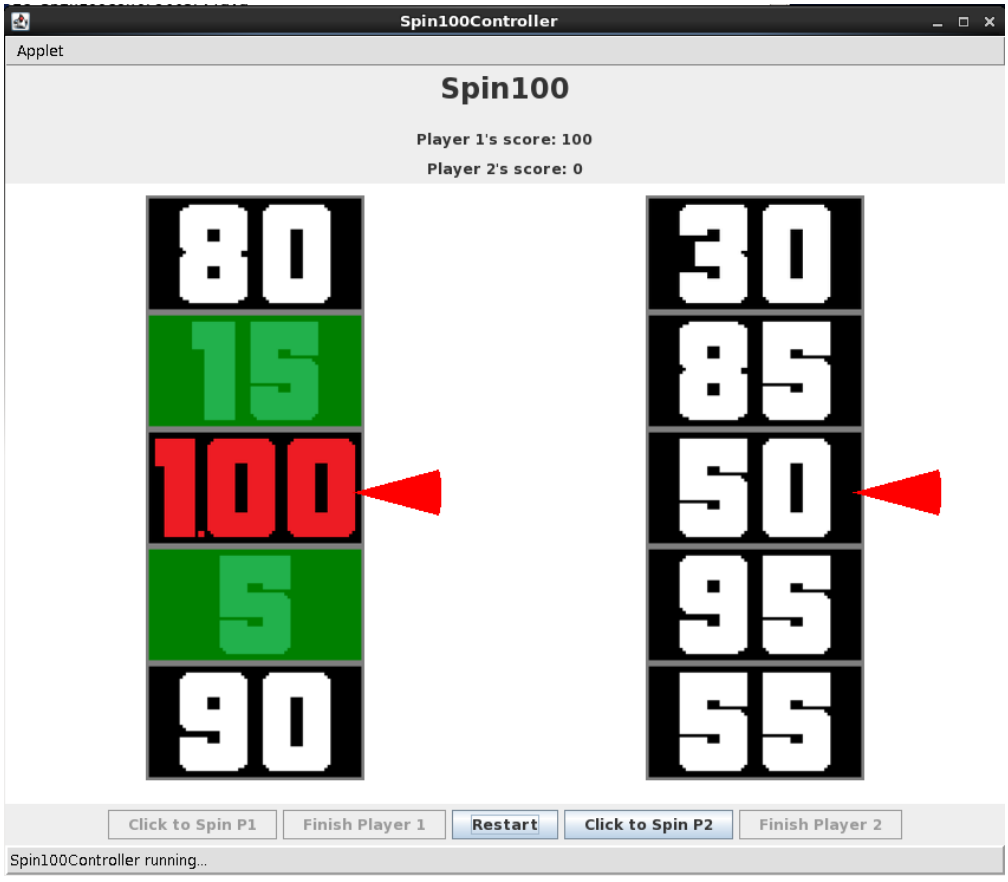
4. Sample Screenshots:



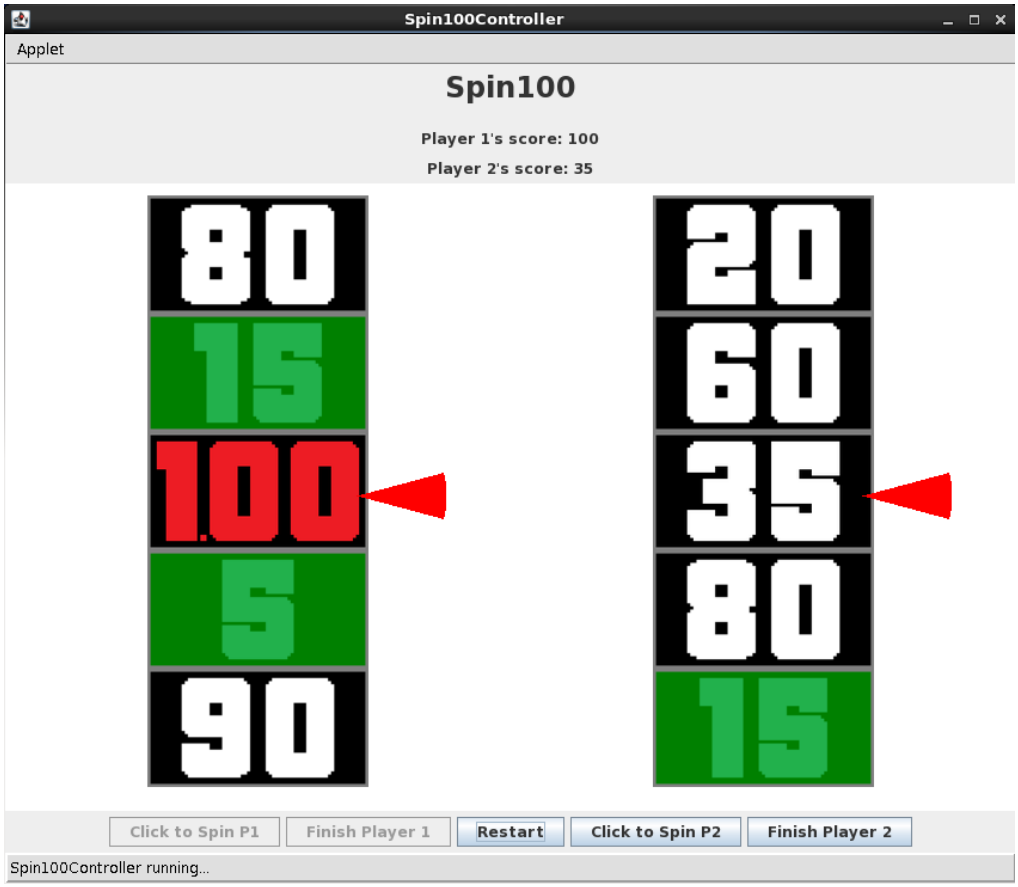
The start of the program. Both player's scores are set to 0, with the arrows pointing to 50. Only the buttons: "Click to Spin P1" and "Restart" are enabled.



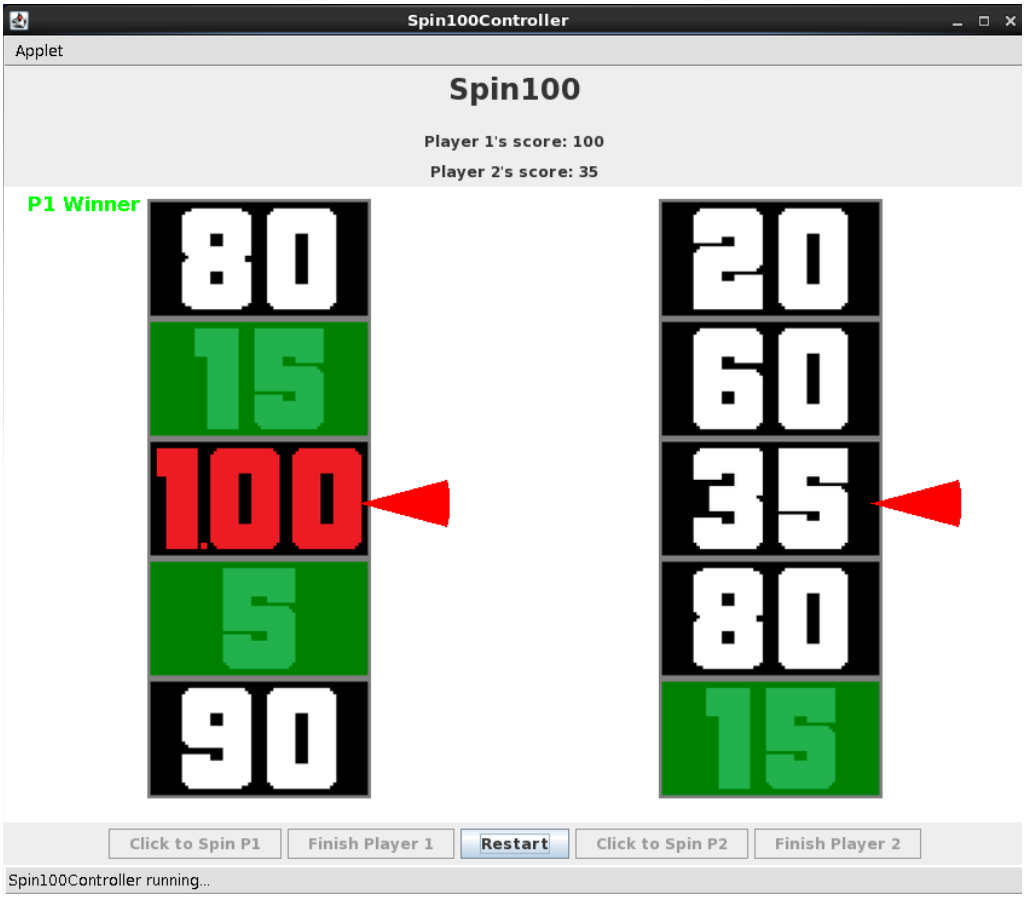
After Player 1 clicks “Click to Spin P1”, the score for Player 1 is updated. The “Finish Player 1” button is now enabled after 1 spin.



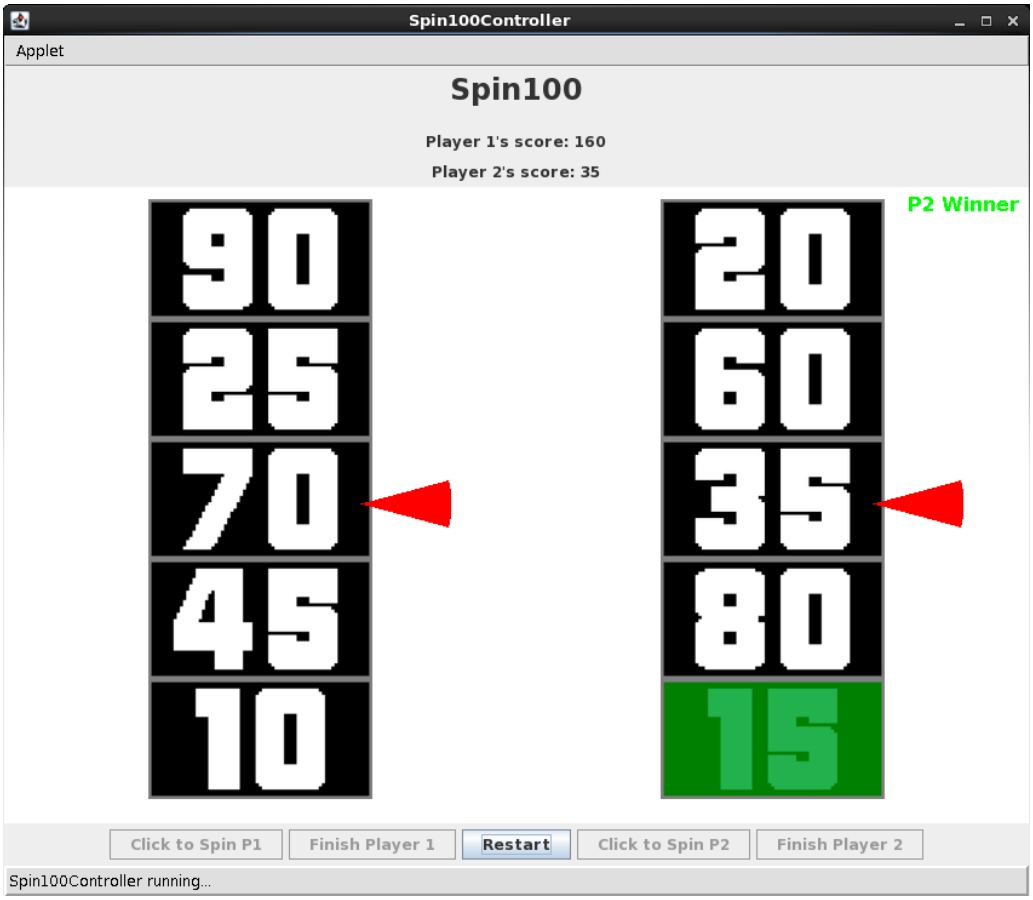
Player 1 clicks “Finish Player 1”, and now only “Restart” and “Click to Spin P2” buttons are enabled.



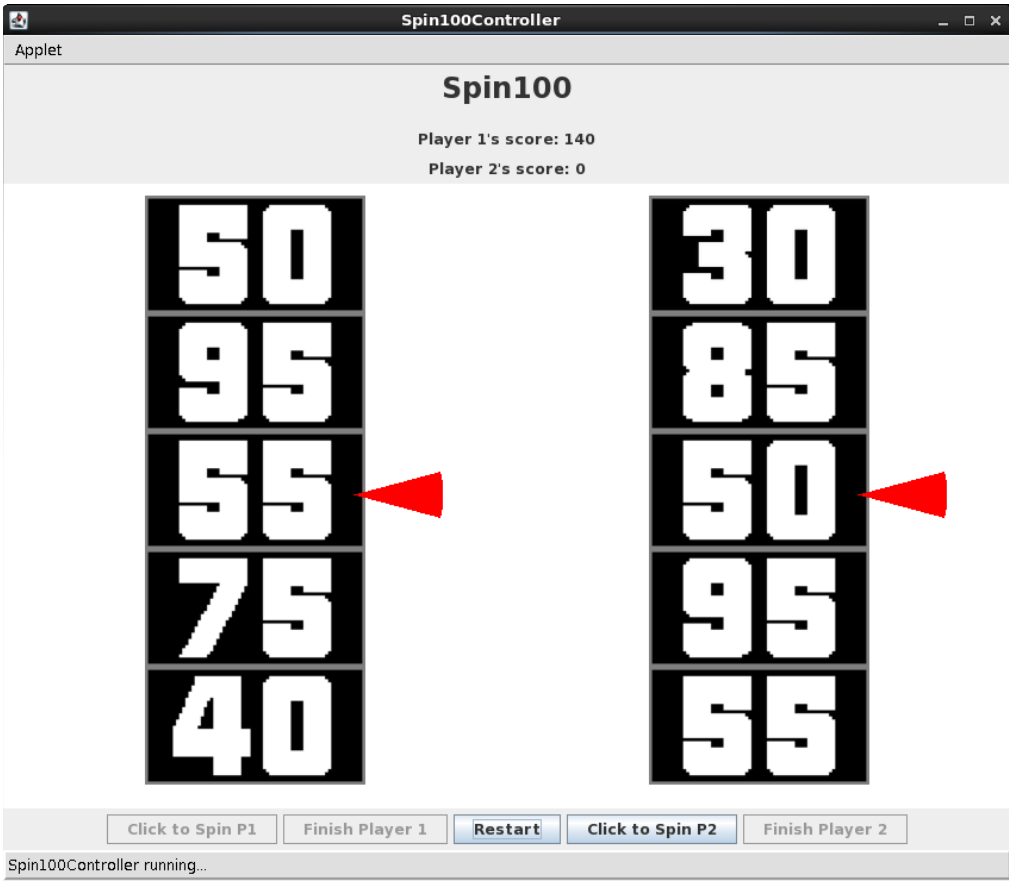
Player 2 now clicks “Click to Spin P2”, the score for Player 2 is updated. “Finish Player 2” button is now enabled after the first spin.



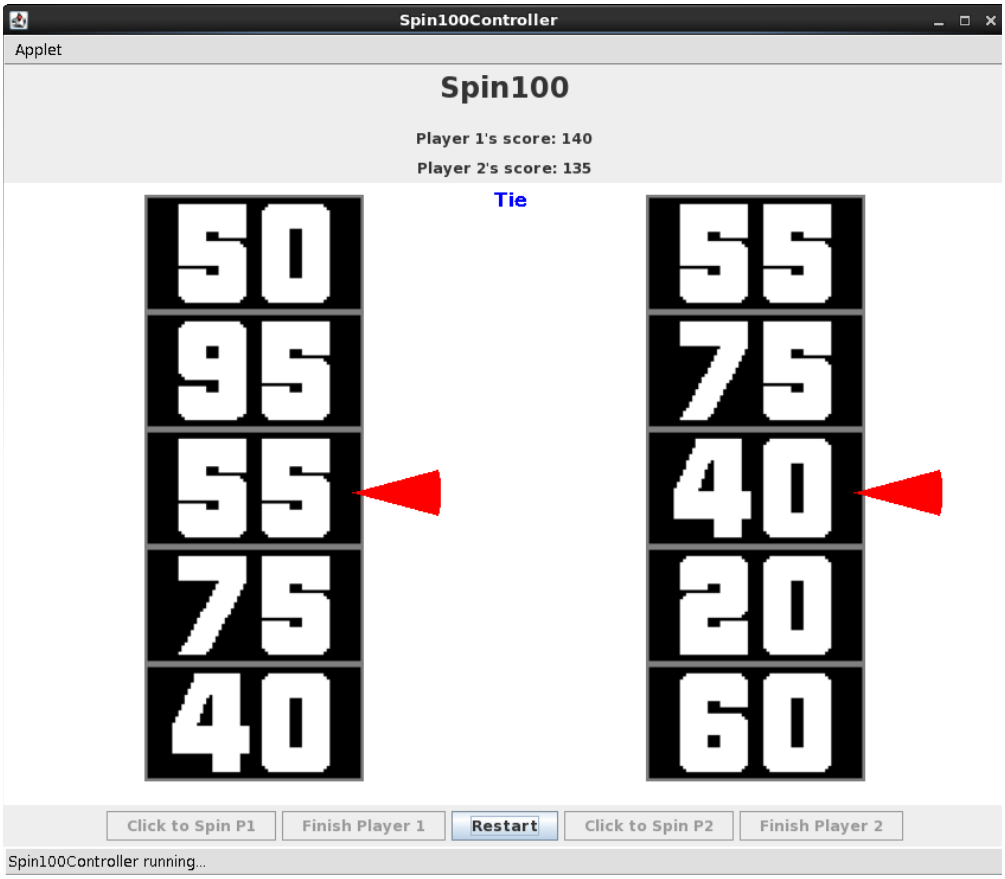
After Player 2 clicks “Finish Player 2” The winner text appears. The only button enabled is “Restart”.



This is the case when Player 1's score is over 100, thus Player 2 wins.



In the case that Player 1's score exceeds 100, the buttons "Click to Spin P1" and "Finish Player 1" are automatically disabled, allowing P2 to begin.



If both player's scores exceed 100, the game results in a tie.

Turnin
To turnin your code, navigate to your home directory and run the following command:

> **cse11turnin pa7**

You may turn in your programming assignment as many times as you like. The last submission you turn in before the deadline is the one that we will collect.

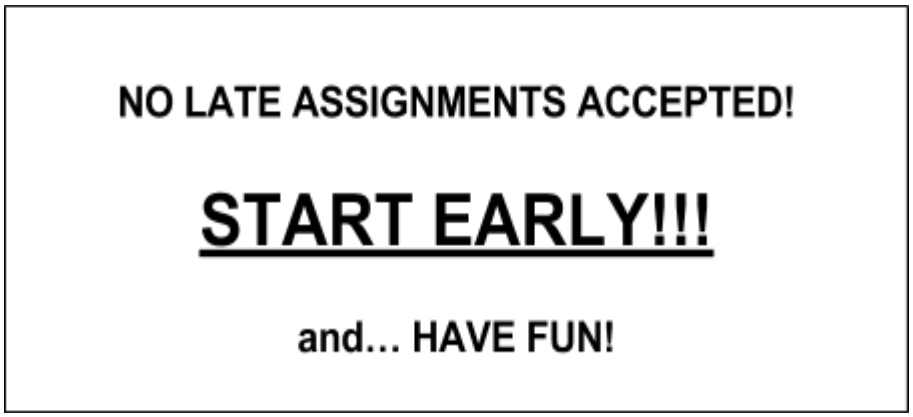
Verify
To verify a previously turned in assignment,

> **cse11verify pa7**

If you are unsure your program has been turned in, use the verify command. We will not take any late files you forgot to turn in. Verify will help you check which files you have successfully submitted.
It is your responsibility to make sure you properly turned in your assignment.

Files to be collected:
Note: You must turn in all the files necessary for your program to run, including the .jar files and .png files.
Note: Also, if any exceptions are thrown in the terminal, you will lose points.

Source Files:	Image Files:	Misc:
PA7indrome.java Spin100Controller.java Spin100Wheel.java	*.png (20 of them)	objectdraw.jar Acme.jar README EC_Spin100Controller.java (extra credit) EC_Spin100Wheel.java (extra credit)



Extra Credit (5 points)

Setup: Create copies of your regular source files and name them **EC_Spin100Controller.java** and **EC_Spin100Wheel.java**.

Along with the functionality listed below, you will need to change the size of the window if you are adding the extra buttons. Set the canvas width to 1000 and the canvas height to 660.

Overall Wins: 1 Point

Keep track of the overall wins with a message that appears at the end of the game:

Overall Wins P1: #, P2 @

where # is a non-negative number indicating how many times player 1 has won the game and @ is a non-negative number indicating how many times player 2 has won the game. This message cannot overlap either wheel and it should be horizontally centered between the wheels. It should be 200 pixels above the vertical center of the canvas. The message can be in whatever color you would like besides black. In the case of a tie, neither player's win count should change.

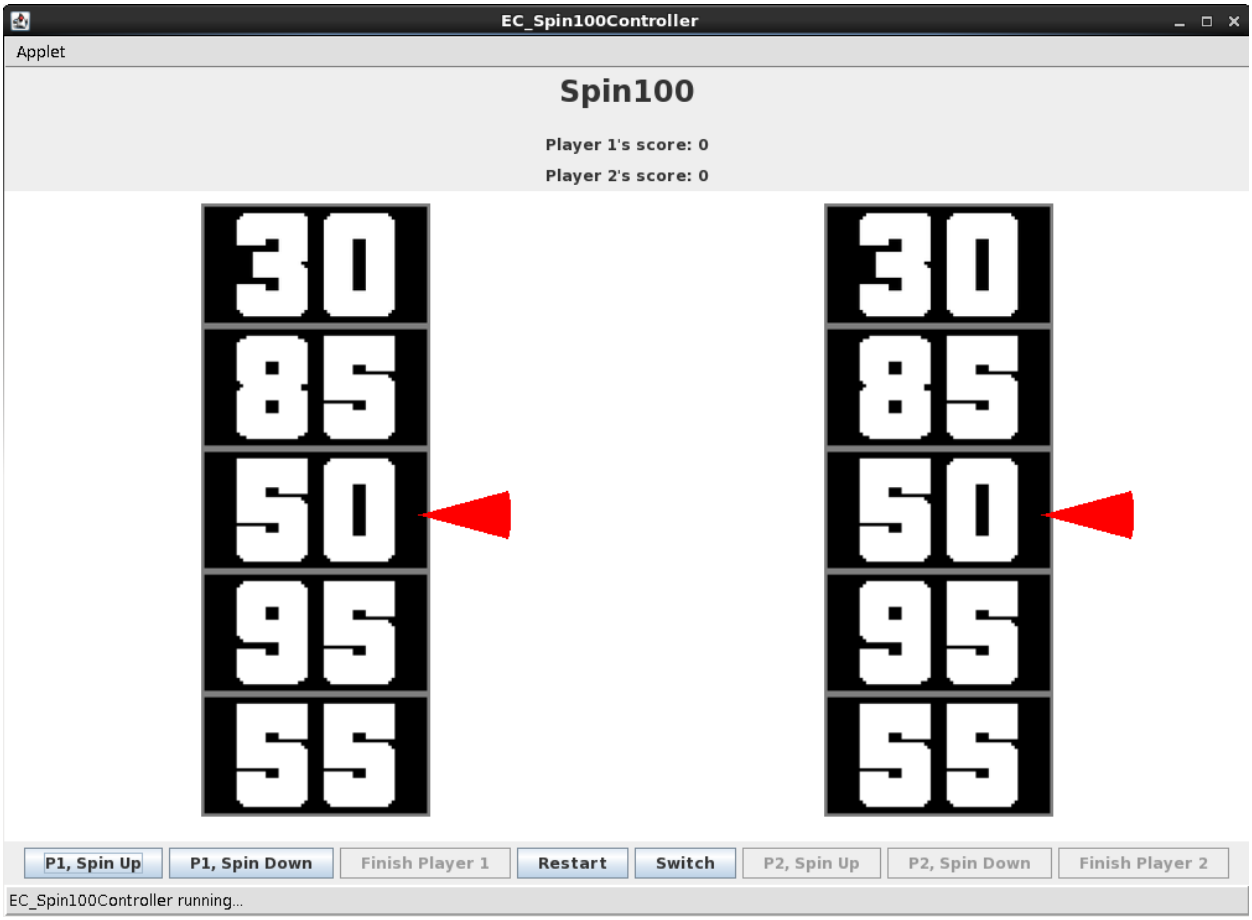
Change Spin Direction: 1 Point

Add some new buttons and change the old ones so that each player has the option to spin up or down. There should be a “P1, Spin Up” button, “P1, Spin Down” button, “P2, Spin Up” button, and “P2, Spin Down” button. To clarify, spinning up is where the numbers appear to be moving up (so the 50 goes towards the top) whereas spinning down is where the numbers appear to be moving down (so the 50 moves towards the bottom). You can see the new buttons in the screenshots below. Disabling the buttons should still work the same as it does in the normal portion of the PA, just make sure to disable both the Up and Down spin buttons.

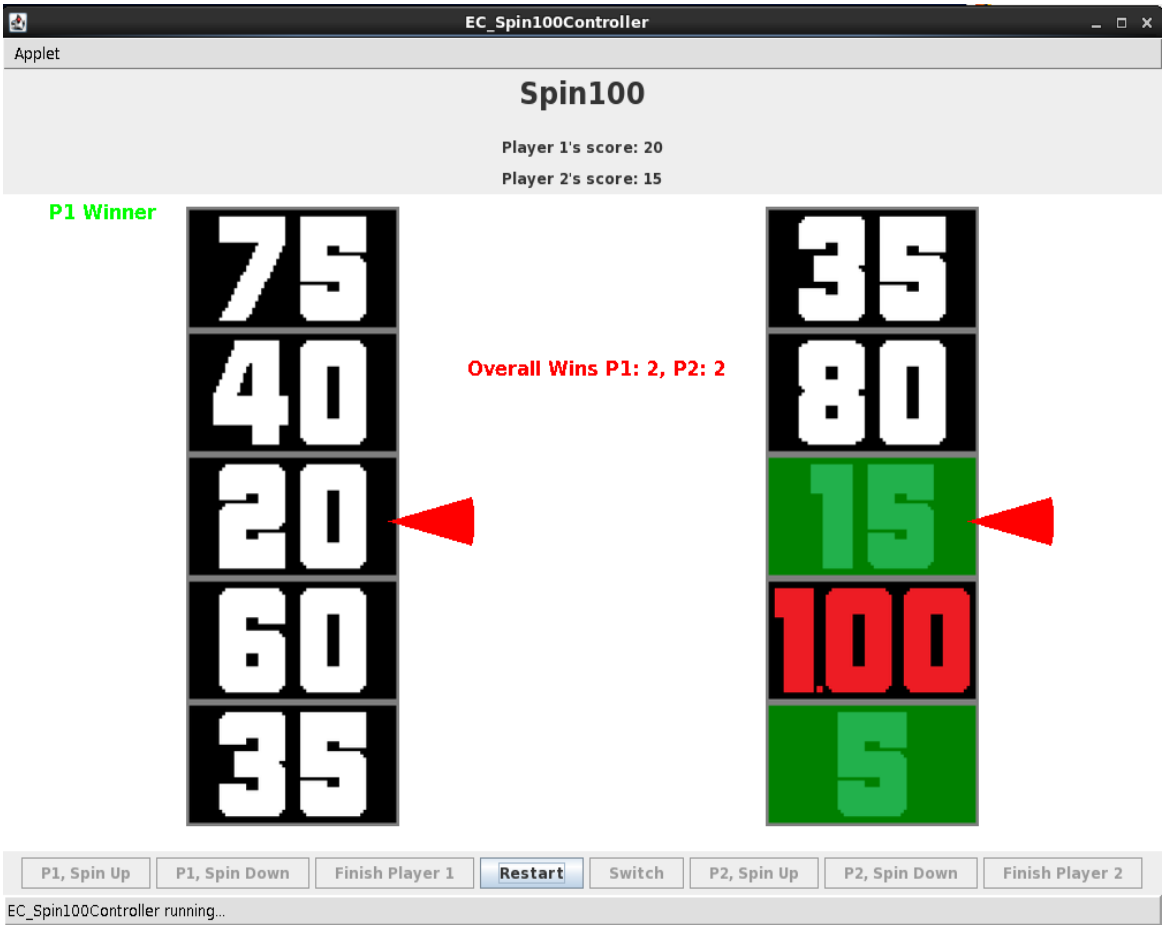
Switch Starting Players: 3 Points

The “Switch” button is enabled only at the start of each round of the game. This button should be disabled in the middle of a game (i.e. if a spin has occurred). This allows the player who spins first to switch with the other player. For instance, if Player 1 was going to start first and the switch button is clicked. then the game will restart and Player 2 will now start first. If the restart button were then clicked, Player 2 would still be going first.

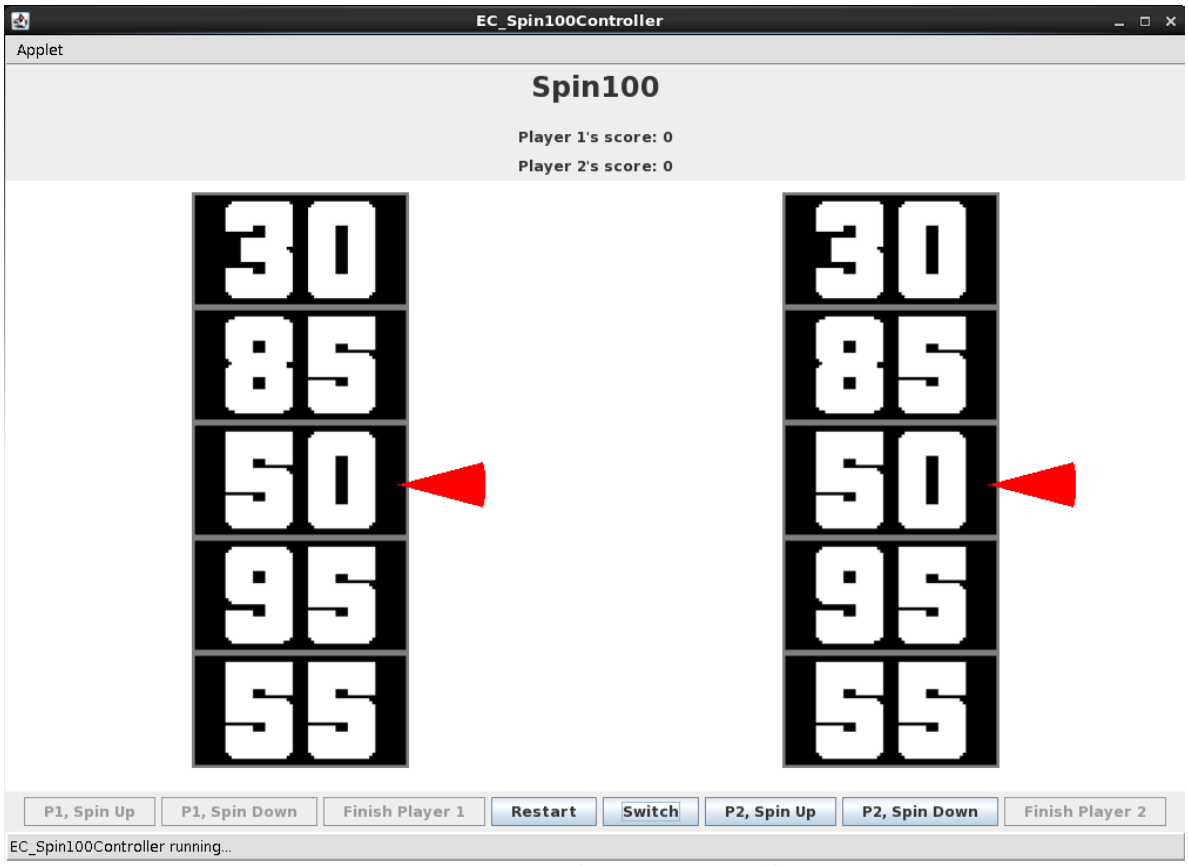
Sample Screenshots:



If all extra credit features are implemented, this is the initial state of the program.



The overall number of wins for each player is shown when a round finishes.



Player 2 can begin the round of a game when “Switch” is clicked.