Programming Assignment 7 (100 Points)
Due: 11:59pm Thursday, November 20

Start Early!!!

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 grandmother 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. **The more detailed the explanation, the more points you will receive.**

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 (between 0-9), all in one step/line?

STYLE (20 points)

Please see preview programming assignments.

You will be specifically graded on commenting, file headers, class and method headers, meaningful variable names, sufficient use of blank lines, not using more than 80 characters on a line, perfect indentation, no magic numbers/hard-coded numbers other than zero, and use of accessor/mutator methods ***if/when specified***.

CORRECTNESS (70 points total for the 2 programs)

Program 1: Reverse-Recurse (35 Points)

ReverseRecurse.java: Write an application that reads integer values from the keyboard into an array whose size is specified by the user (the user may enter fewer integers than the size specified, but not more) and reverse the elements in the array via two different recursive methods. One method directly modifies the original array. The other method returns a new array with the elements reversed preserving the original array. Both will use an "ends-and-middle" or "edges-and-center" recursion.

You will be given the main test driver that we will use to call/test your solution.

```
~/../public/PA7/TestReverseRecurse.java
```

And a PA7Strings file with all the strings you will use:

```
~/../public/PA7/PA7Strings.java
```

You will write

```
public class ReverseRecurse { … }
public int[] initArray() { … }
public void printArray( int[] array ) { … }

/*
 * The following two reverse() methods must be implemented using
 * recursion.
 */

public void reverse( int[] originalArray, int low, int high ) { … }
public int[] reverse( int[] originalArray ) { … }
```

initArray() will ask the user for a maximum number of integers expected, create an array of integers that size, read at most that many integers from the keyboard using a Scanner object (ignore extra input beyond the size of the array), and return the initialize array. The user must enter a positive integer greater than 0 for the size of the array. Keep asking the user for a positive integer greater than 0 until a valid value is entered.

If the user indicates EOF (see next sentence) or a non-integer input, the program should exit with exit status 1 [System.exit(1);]. If the user hits <Ctrl>+D (Unix/Mac) or <Ctrl>+Z (DOS/Windows) to indicate EOF (no more input) or any non-integer before the entire array has been filled, return an array resized with only the values that were entered (no extra/unfilled array slots). **Hint:** Use System.arraycopy().

To read ints from standard input (**System.in** which is the keyboard by default), use a **Scanner** object (see the Javadocs for class Scanner).

```
Scanner in = new Scanner( System.in );
```

You can then use method **hasNextInt()** to determine if the next token on the input is a valid int and method **nextInt()** to scan the next token as an int.

```
if ( in.hasNextInt() ) {
    size = in.nextInt();
    ...
}
```

printArray() cycles through the passed array printing each integer on the same line separated by a space. Output a newline after all the elements are printed. Use either a **for** loop or the **enhanced-for** loop to iterate over the elements in the array. This method will print "Empty array" if there are no elements in the array (an empty array).

The **first overloaded recursive reverse()** will directly manipulate the array passed in by exchanging the low and high index values and recurse on the remaining middle/center elements of the array by passing modified values of low and high in the recursive call. Set up the Base Case(s) and Recursive Case(s).

The **second overloaded recursive reverse()** will not change anything in the original/passed array. Instead it will copy the first element of the original (passed) array into the last slot of a new array we are building to hold the reverse of the passed array and copy the last element of the original (passed) array into the first slot of the new reversed array. Then copy (think System.arraycopy()) the middle elements of the passed array into another new array (you need to dynamically create this new array based on the size/number of the middle elements) and recursively reverse these middle elements in this new array. (Leap of Faith.) reverse() returns an array with the passed array elements reversed (it does not modify the passed array). After the recursive call to reverse the middle elements, copy the reversed middle elements that was returned by the recursive call to reverse() into the middle section of the new reversed array being built and return this new array that now has all the elements of the passed array reversed. Set up the Base Case(s) and Recursive Case(s).

Make sure that the last two tests in the tester pass with "SUCCESS: No NullPointerException thrown." message by dealing with null array input in your two reverse() methods. The key point here is that you do not manipulate the null object passed in as array in your reverse methods, but immediately return. If you do not deal with this special corner case, the last two tests will print "FAIL: NullPointerException – Fix me!".

Please note that your ReverseRecurse will be run though a script. We stress that you should match the output ***EXACTLY***. In order to make your lives easier, we have included a PA7Strings.java for your convenience. In order to use these strings, you need to call these strings in this manner:

```
System.out.println(PA7Strings.EMPTY);
```

In order to make this work, we made the strings in PA7Strings.java static, so they can be accessed through the class name. Because of this, A PA7Strings.java object DOES NOT need to be instantiated.

Also, because one of the parts of the string needs to be dynamically set, we are going to introduce a new print method: System.out.printf(); This is the same as the other print statements, except it can be used to insert values into a string. Let's take the following example:

```
ENTER_INTS = "\nEnter up to %d integers:";
```

In this case, %d signifies a decimal placeholder. It needs a decimal value as input in order to place the string. It can be used in the following manner:

```
int decimal = 10;
System.out.printf(PA7Strings.ENTER_INTS, decimal);
```

This will print "Enter up to 10 integers:". Now apply this thinking to make the number of integers that should be read dynamic.

Here are some example test runs: (the following outputs assume the last two tests pass)

```
[cs11fxx@ieng6-201]:~:$ java TestReverseRecurse
Maximum number of integers you wish to enter? -5
You must enter a value > 0; Try again.

Maximum number of integers you wish to enter? 0
```

```
You must enter a value > 0; Try again.

Maximum number of integers you wish to enter? 42

Enter up to 42 integers:
-9 4 33
88 23 53 76
55
-83
7757
```
<-------- User enters **<Ctrl>+D** here to indicate EOF
```
The original array:
-9 4 33 88 23 53 76 55 -83 7757

The array reversed (manipulating array directly):
7757 -83 55 76 53 23 88 33 4 -9

The array reversed again (manipulating array directly)
(should be back in original order):
-9 4 33 88 23 53 76 55 -83 7757

The copy of the original array:
-9 4 33 88 23 53 76 55 -83 7757

The array reversed (reversed array returned vs. direct manipulation):
7757 -83 55 76 53 23 88 33 4 -9

The original array showing original NOT modified:
-9 4 33 88 23 53 76 55 -83 7757

Testing reverse method (direct manipulation) with null array input:
  SUCCESS: No NullPointerException thrown.

Testing with reverse method (returned reversed array) with null array input:
  SUCCESS: No NullPointerException thrown.
[cs11fxx@ieng6-201]:~:$
```

Note the user entered <Ctrl>+D before all 42 integers were entered to indicate no more input. The array returned by initArray() is sized to the number of actual integers entered.

```
[cs11fxx@ieng6-201]:~:$ java TestReverseRecurse
Maximum number of integers you wish to enter? 6

Enter up to 6 integers: ttt
The original array:
Empty array

The array reversed (manipulating array directly):
Empty array

The array reversed again (manipulating array directly)
(should be back in original order):
Empty array

The copy of the original array:
Empty array

The array reversed (reversed array returned vs. direct manipulation):
Empty array
```

```
The original array showing original NOT modified:
Empty array

Testing reverse method (direct manipulation) with null array input:
  SUCCESS: No NullPointerException thrown.

Testing with reverse method (returned reversed array) with null array input:
  SUCCESS: No NullPointerException thrown.
[cs11fxx@ieng6-201]:~:$
```

Note any non-integer input (like "ttt") will terminate the input process. Again, see the example above on how to use Scanner and check for and read an integer from the console.

```
[cs11fxx@ieng6-201]:~:$ java TestReverseRecurse
Maximum number of integers you wish to enter? 5

Enter up to 5 integers:
1 2 3 4 5 6 7 8 9 10

The original array:
1 2 3 4 5

The array reversed (manipulating array directly):
5 4 3 2 1

The array reversed again (manipulating array directly)
(should be back in original order):
1 2 3 4 5

The copy of the original array:
1 2 3 4 5

The array reversed (reversed array returned vs. direct manipulation):
5 4 3 2 1

The original array showing original NOT modified:
1 2 3 4 5

Testing reverse method (direct manipulation) with null array input:
  SUCCESS: No NullPointerException thrown.

Testing with reverse method (returned reversed array) with null array input:
  SUCCESS: No NullPointerException thrown.
[cs11fxx@ieng6-201]:~:$

[cs11fxx@ieng6-201]:~:$ java TestReverseRecurse
Maximum number of integers wish to enter? 5

Enter up to 5 integers:
11
22
33
44
55
```

<-------- Stops reading input after the 5th input

```
The original array:
11 22 33 44 55
```

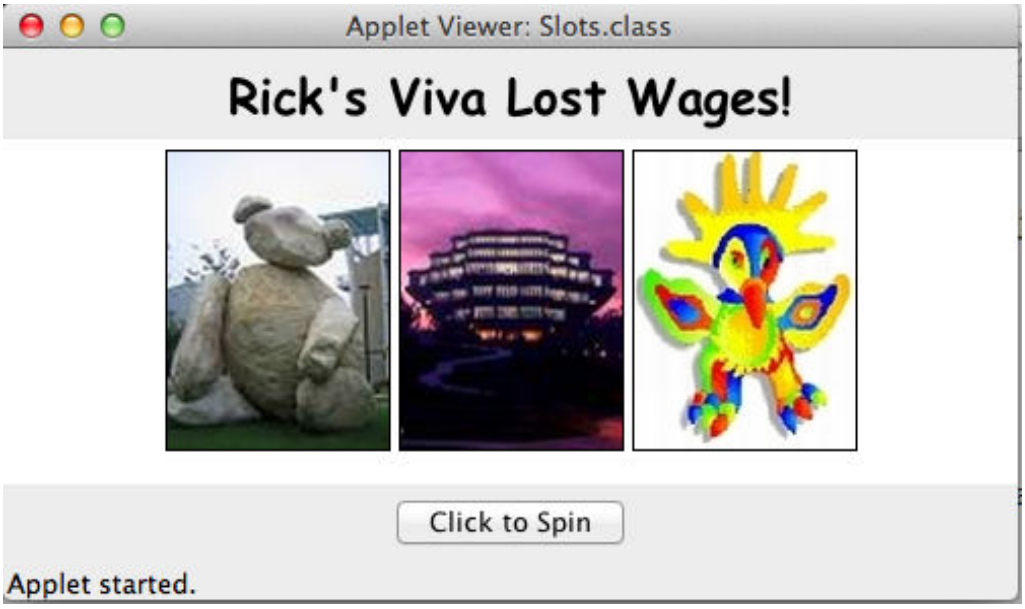
```
The array reversed (manipulating array directly):
55 44 33 22 11

The array reversed again (manipulating array directly)
(should be back in original order):
11 22 33 44 55

The copy of the original array:
11 22 33 44 55

The array reversed (reversed array returned vs. direct manipulation):
55 44 33 22 11

The original array showing original NOT modified:
11 22 33 44 55

Testing reverse method (direct manipulation) with null array input:
  SUCCESS: No NullPointerException thrown.

Testing with reverse method (returned reversed array) with null array input:
  SUCCESS: No NullPointerException thrown.
[cs11fxx@ieng6-201]:~:$
```

Here is also an example where the null array input tests do **NOT** pass (and **should be fixed**):

```
[cs11fxx@ieng6-201]:~:$ java TestReverseRecurse
Maximum number of integers wish to enter? 2

Enter up to 2 integers:
1 2

The original array:
1 2

The array reversed (manipulating array directly):
2 1

The array reversed again (manipulating array directly)
(should be back in original order):
1 2

The copy of the original array:
1 2

The array reversed (reversed array returned vs. direct manipulation):
2 1

The original array showing original NOT modified:
1 2

Testing reverse method (direct manipulation) with null array input:
  FAIL: NullPointerException – Fix me!

Testing with reverse method (returned reversed array) with null array input:
  FAIL: NullPointerException – Fix me!
[cs11fxx@ieng6-201]:~:$
```

Note in the two examples above initArray() should only accept the number of integers specified by the user. This should all just work using something similar to the Scanner example above.

Program 2: Slot Machine Simulator (35 Points)



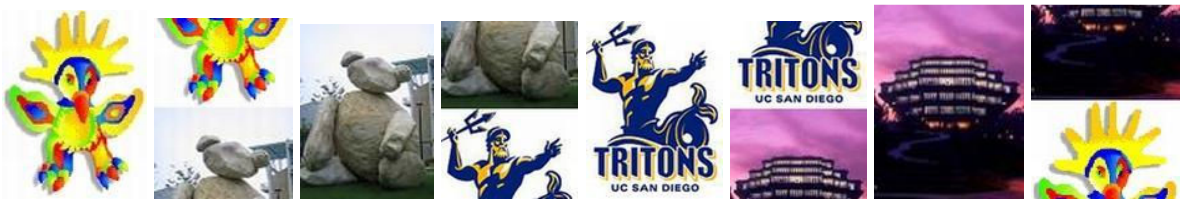
You will certainly want to customize the header label. See the setFont() method and class Font: I used font face "Comic Sans MS", style Font.BOLD, and point size 24.

This program will have two Java source files: **Slots.java** and **SlotWheel.java**.

Slots.java will have the main controller object (extends WindowController) that does the GUI layout with a label on a panel at the top and a button on a panel at the bottom with the drawing canvas in the center. It will need to initialize an array of Image(s) - the images are given to you and are available to be copied to your pa7 directory from

```
~/../public/PA7-images/
```

The images need to be arranged in the array such that the intermediate images reflect a transition from one full image to the next. For example:



sungod.jpg **sungod-bear.jpg** **bear.jpg** **bear-triton.jpg** **triton.jpg** **triton-library.jpg** **library.jpg** **library-sungod.jpg**

Slots.java also needs to calculate the Location where each slot wheel should be located such that the middle slot wheel is centered in the middle canvas area in the x coordinate and down 5 pixels offset in the y coordinate. There should be 5 pixels space between each slot wheel.

You do not need to worry about the applet resizing and recalculating the locations of the slot wheels. Just make sure your Slots.html file specifies a large enough applet area to hold the slot wheels. My Slots.html file used width="500" and height="250", but do not assume this dimension. Dynamically calculate the locations of the slot wheels at program start-up based on the canvas's width.

Next Slots.java needs to create each SlotWheel object passing as a minimum the following information to each SlotWheel object:

- the array of Image(s) initialized in Slots
- the number of ticks (images) this slot wheel will use as it rotates
- the delay (in milliseconds) this slot wheel will use in its run() thread
- the location of this slot wheel (location of the wheel image and frame - border around the image)
- a reference to the drawing canvas to place the slot wheel (wheel image and frame)

Here are some constants I used in Slots.java that may be helpful:

```
private static final int NUM_OF_IMAGES = 8;
private static final double IMAGE_WIDTH = 110;
private static final double IMAGE_HEIGHT = 145;
private static final double WHEELS_Y_OFFSET = 5;
private static final double SPACE_BETWEEN_WHEELS = 5;
private static final int WHEEL_1_TICKS = 22;
private static final int WHEEL_2_TICKS = WHEEL_1_TICKS + 6;
private static final int WHEEL_3_TICKS = WHEEL_2_TICKS + 6;
private static final int WHEEL_1_DELAY = 100;
private static final int WHEEL_2_DELAY = WHEEL_1_DELAY + 25;
private static final int WHEEL_3_DELAY = WHEEL_2_DELAY + 25;
```

To make the images look a bit more realistic (like they are images on a slot wheel that is turning) the slot wheels need to "turn" at different rates (the wheel delays) and they stop at different times (left slot wheel stops first [least number of ticks] then the middle slot wheel stops and lastly the right slot wheel stops). All the images are exactly 110 x 145 pixels.

And the last thing the Slots controller needs to do is register each slot wheel it just created as a listener for events on the spin button.

SlotWheel.java will define the simulated slot wheel. Each slot wheel is an ActiveObject (so we can run it as an animation in its own thread) and needs to handle the action event fired by the spin button back in the controller. This is similar to the ResizableBall objects from the last assignment.

The constructor should initialize instance variables associated with the values passed into its formal parameters and create a double (pseudo)random number generator object that will generate (pseudo)random values between 0-1. Call a private method getWheelIndex() (detailed below) that returns an index into the array of Image(s) passed into the constructor and use this random index/Image to create a VisibleImage object to display as this slot wheel's image. Create a border around this slot wheel image (FramedRect). As with all ActiveObjects, the last statement in the constructor should be start();.

getWheelIndex() should take a double value between 0-1 and return either 0, 2, 4, or 6 to indicate which whole image from the array of Image(s) to use as the image on this slot wheel. If the parameter value is between 0 and 0.25, return 0 ... and so on.

actionPerformed() will be called in response to the spin button being clicked back in the controller object. This method should simply reset the number of ticks this slot wheel should spin and pick another random starting Image index.

The **run()** method performs the animation/simulation of the slot wheel spinning. In a forever loop check if this slot wheel still has some ticks left before it is supposed to stop spinning. If there are ticks left on this spin, set the index into the array of Image(s) to the next index. Since we are treating this array as a circular array, when we are at the last index in the array the next index will be back at index 0. So you need to wrap back around to the head of the array when you are incrementing the index past the end of the array. Hint: Think mod operator. Once we have a new index in the array of Image(s), set the Image on the slot wheel to this new Image (setImage()). Finally decrement the number of ticks left for this slot wheel spin. Don't forget to pause the animation using this slot wheel's delay in milliseconds that was passed in to the constructor.

The animation of slot wheels spinning looks better when each wheel spins at a slightly different speed (different pause() delays). The thrill and excitement of a possible win is enhanced when the left wheel stops spinning first, then the middle wheel, and finally the last wheel. [Slots are one of the worse gambling games to play! A common nickname for a slot machine, especially back in the day where you actually had to pull a handle to get the wheels to spin, is "One-Armed Bandit". But they are fun to program.]

One more thing you will have to worry about is a race condition when assigning the wheel ticks and wheel index while the slot machine is currently running. You will need to use Java's synchronization blocks. Use:

```
synchronized(this) { /* code that needs synchronizing */ }
```

to surround where you are assigning your wheel ticks and wheel index when another spin is initiated (in actionPerformed()). Use this again to surround code where you are setting the next wheel index and decrementing the wheel ticks in your while loop (in run) - do not synchronize the while statement and the pause() at the end of the while loop; only synchronize the code dealing with the wheel index and wheel ticks inside the while loop. This will allow one set of code to finish before the other is started. Since the run() thread and the event handling thread are potentially running "at the same time" as independent threads, we need to guard any statement that can change/access state that is used in both threads (wheel ticks and wheel index). What could happen if you didn't synchronize?

Running

Reverse-Recurse (application):

You should already know how to run this program.

Slot Machine Simulator (applet):

To run (and view) your applet, first create an html file named **Slots.html.** Please see previous assignments for similar code snippets. You will need to adjust the width and height of the applet to make the slot machine look similar to the screenshots provided above.

Turnin

To turnin your code, navigate to your home directory and run the following command:
> **turnin 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.

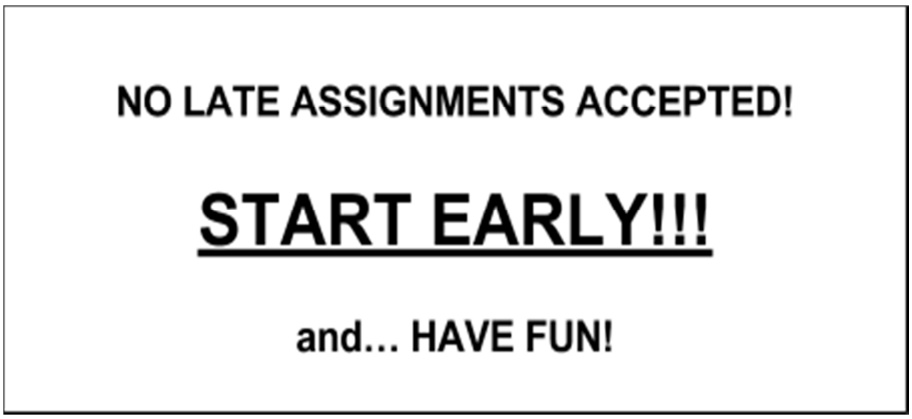
Verify

To verify a previously turned in assignment,
> **verify 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:

Source Files:	**Image Files:**	**Misc:**
ReverseRecurse.java TestReverseRecurse.java Slots.java SlotWheel.java **Slots.html**	bear.jpg bear-triton.jpg library.jpg library-sungod.jpg sungod.jpg sungod-bear.jpg triton.jpg **triton-library.jpg**	**objectdraw.jar** **PA7Strings.java** **README**



Extra Credit (5 points)

*For **Slots** Program:*

Various amounts of extra credit. With extra credit, you are allowed to (and probably need to) change the look and feel of the applet (for example, larger applet window and additional GUI components) and what parameters are passed (for example, to the SlotWheel constructor). Enhance the Slot Machine applet to include:

1 Point - Detect and announce a Win (all three slot wheels stop on the same image). Use a Text object to announce the Win. The Text object announcing the Win should be hidden with the next Spin.

1 Point - Keep track of the number of Spins and the number of Wins in read-only text fields. You need to do both # of spins and # of wins to get this extra credit.

1 Point - Have the Win! announcement be an image that gets displayed with the detection of a Win and hidden on the next Spin. Create the Win message with any drawing/paint tool of your choice (do not just find an image on the Net and use it). If you do this extra credit, you will automatically get the previous "Win detection" extra credit (you do not need to do both a Text object Win and an image object Win message).

2 Points - Allow the user to enter a deposit amount (in whole dollars) for a bankroll (say with a text field), and allow the user to enter how many dollars to bet on each spin up to a max of five dollars (say with a combo box). Payoff 15 times the bet. This works out to a 93.75% payout ... Vegas odds. [There are 4*4*4 = 64 different combinations of the four images coming up across the three wheels. There are four different win combinations (all three wheels having the same image). So on average you would expect a win 1 out of every 16 spins (64/4 = 16). A real gambling operation will take a house cut.]