PA7: ReverseRecurse & Slots

02/22/2013

Part 1: ReverseRecurse

```
import java.util.*;

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

  /* The following two reverse methods must be implemented
     with recursion */
  public void reverse(int[] originalArray, int low, int high) { ... }
  public int[] reverse(int[] originalArray) { ... }
}
```

Part 1: ReverseRecurse

- Prompt the user to enter the max size of the array
- Use the Scanner class
  - `Scanner input = new Scanner(System.in);`
  - `hasNextInt() // tells you if the input is an int or not`
  - `nextInt()    // gives the int to you`
- Store the input into the array

Part 1: reverse(int[], int, int)

- Directly manipulate array by swapping the first and last elements
- Think edges to middle recursion
  - [a,b,c,d,e] -> swap a and e
  - [e,b,c,d,a] -> swap b and d
  - [e,d,c,b,a] -> base case (1 element left)

Part 1: reverse(int[])

- Reverse the array WITHOUT modifying the original array
- Return the copy, not the original
- Must use recursion
- Base case is when length 0 or 1

Part 1: reverse[int[]]

- Recursive case:
  - Think edges to middle again using a copy
  - Copy first element of original array into the last slot of the new array
  - Copy the last element of the original array into the first slot of the new array.
  - Recursively reverse the middle and copy into the new array

Look up System.arraycopy

Part 2: Slot machine

- Files you need to create:
  - Slots.java
    - public class Slots extends WindowController
  - SlotWheel.java
    - public class SlotWheel extends ActiveObject implements ActionListener

Part 2: Slots.java

- Creates the wheels
  - At the center of the canvas
  - 5px between each wheel
- Loads up the images into an array to pass to the slot wheels

- Kind of like ResizableBallController from before

Part 2: Slots.java

- Various helpful constants [listed in HW write-up]

```
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;
```

Part 2: SlotWheel.java

- Constructor
- getWheelIndex[]
- actionPerformed[]
- run[]

Part 2: SlotWheel.java constructor

- Make sure to pass in the canvas and the array of pictures
- Initialize variables, etc
- Call start[]

Part 2: SlotWheel.java getWheelIndex()

- If between 0.00 and 0.25 -> return 0
- If between 0.25 and 0.50 -> return 2
- If between 0.50 and 0.75 -> return 4
- If anything else -> return 6

Part 2: SlotWheel.java actionPerformed()

- Called when the user clicks the spin button
- Reset the ticks
- Pick a new random picture to start the spin with

Part 2: SlotWheel.java run()

- Use an infinite loop again
- Ticks count down
  - If the ticks are 0, the wheel stops
  - If ticks > 0, change to the next picture and decrement the ticks
- Make sure to use pause() between each iteration of the loop

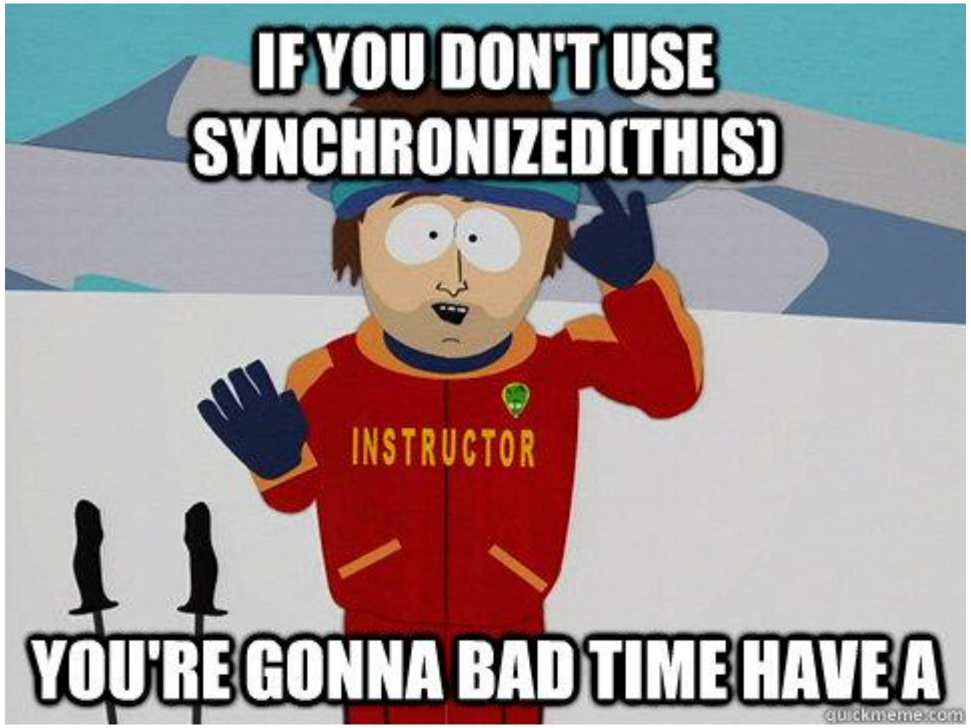
Part 2: RACE CONDITIONS

- You will run into a common problem called a race condition
- A race condition occurs when two threads compete and the correctness of the output depends on who wins the "race"
- Fix this with synchronized(this) {} blocks in your code when changing wheel ticks

Part 2: RACE CONDITIONS

- The root of this problem is that you write to two variables in actionPerformed()
- The key to understanding the problem is that actionPerformed() and run() are taking turns, so this can happen:
  - actionPerformed() ticksLeft = 34
  - run(): ticksLeft–
  - run(): currImageIndex++
  - actionPerformed() currImageIndex = 0
- Ruh roh! ticksLeft is now 33, and currImageIndex=0, so we end on an odd number (half image)

Part 2: RACE CONDITIONS


IF YOU DON'T USE SYNCHRONIZED(THIS)
INSTRUCTOR
YOU'RE GONNA BAD TIME HAVE A
quickmeme.com