Off to the Races

Threads & Graphics - Animation

ISTE-121 – Homework 05

Overview:

This assignment is to have pictures race across the screen. Basically it is simple enough in concept, and when broken into enough smaller parts, it turns out to be fairly straightforward. To accomplish this race, you should write down all the parts that would be needed to complete this application, take each part of the problem, think about what part you want to tackle first, design the solution for this part, write the code it, and test it. Once it is working, save a backup copy of your program (that helped me) and go onto the next item in building your application.

Most students tend to over-think the problem and make it more complicated than required. If you think it is getting difficult, look at the demo code, and ask the instructor to review your design. You do have a design, don't you?

All images start on the left side of the frame, and race to the right side of the screen. See next page for the start and finish screens.

Requirements:

1. Threads (this includes Runnable) must be used, one thread per race lane
2. Filename containing main() (for me to run) must be named **Races.java**
3. When run from the command line, you may specify how many threads are to be run. When no number is specified, 5 is to be used as the default. To run 10 icons/threads at once, you would use:
   C:\> java Races 10
4. **Javadocs** for each class and method, public constants, etc. **are required**.
5. Follow the coding standards for this course that are posted in MyCourses
6. Submit work in the dropbox. Include all your java and class files to get your program working. Do not include HTML files, I will generate them
7. Zip the files when sending them to the Dropbox
8. All inner classes require the **protected** access modifier. This allows the JavaDocs to be generated for the inner class and its methods
9. Use a synchronized block or method (not on the run() or paint() methods) to declare one winner, and to prevent more than one winner from being announced.
10. Image name must be **races.gif** or **races.jpg** for generated graphic
11. Icon can be any picture you choose, that is distinctive from the background. Height and width maximum's are 50 pixels, minimum is 20 pixels.
12. At race end, leave the frame displaying so the users can see the completed race. Let the user close the application
13. Finish line must be one continuous non-broken line

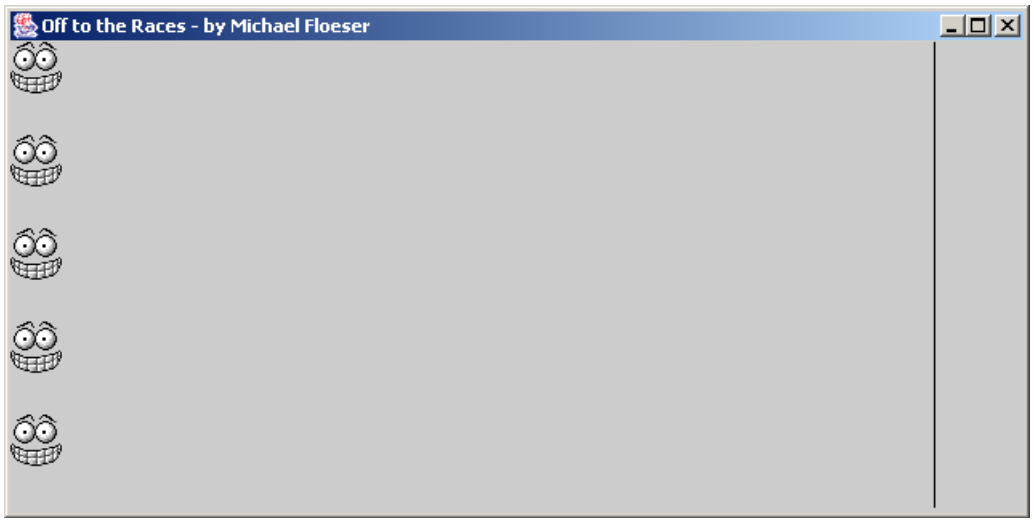
Non-requirements:

1. **UML** is **not** required for this homework.
2. Console output – Not graded. When your program is complete, do not remove or comment out any `System.out.println` statements. Leave them in so I can see what you are doing. Also, do not add any S.O.P lines for my benefit.
3. There are no required controls to "restart" the race, or change the number of racers without restarting. If you'd like to add these items, that would be considered for extra credit. Any other embellishments may also count for extra credit.

"Based on the icon" means using the width and height of the icon. Do not hardcode the icon width and height values into your code. A larger/smaller icon that I use for testing should result in a different size frame.

Starting race screen requirements:

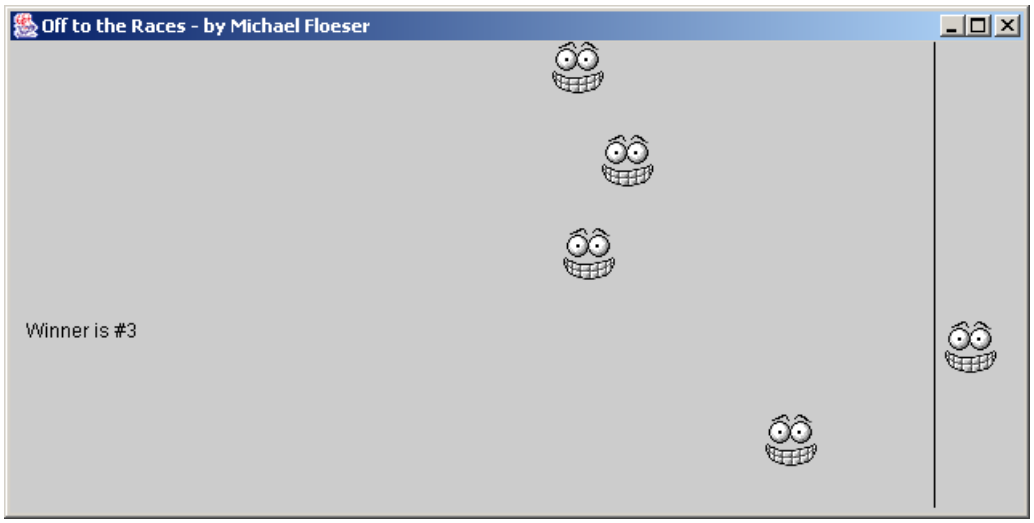
1. The GUI should look like the screen shot below in its starting view and start the application centered in the window.
2. After the screen becomes visible the application pauses for ONE second before starting the race. This is for the user to see it before the icons move
3. Frame title is to have the project name and your name
4. The continuous vertical black line on the right hand side is the finish line
5. The screen width should be a calculated value based on the Icon used. For this icon (32 pixels), the screen width is 20 times the Icon width, or 640 pixels. For example, if a smaller pixel width icon is used (25 pixels) the overall frame width is would be calculated at 25*20 or 500 pixels wide
6. The finish line **MUST be** a calculated value based on percentage of icon width and placed about 1.5 to 2 icon-width's from the right.
7. The frame height is dependent upon how many threads are racing. For this example the frame may need to be set to 1.? to 2.0 times the Icon height.



Above is start of race positions (without any extras)

End of race requirements:

1. Below is how the screen may look at the end of the race. (See note below)
2. The winner ("Winner is #3") is displayed on the screen at the end of the race. This is in the path of the winner, somewhat to the left of the screen. This example uses 0 based indexing and reporting, you must use 1 based counting.
3. Once a thread is declared the winner, all other threads must be halted at once!



Winner 3 is zero based, you should use 1 based counting

Note: If you run your program several times, you should see the Icon that crosses the finish line first is generally announced the winner. Depending upon when you update your display, combined with the timing it takes to shut down the other threads, one or two Icon's may "jump" ahead of the stopped winner. Such a finish is shown on the next page when 10 threads are run. This slightly possible glitch is acceptable for this homework. What is NOT acceptable is to have two racers announced as the winner, or have Icons continue to the finish line.

Recommendations/hints:

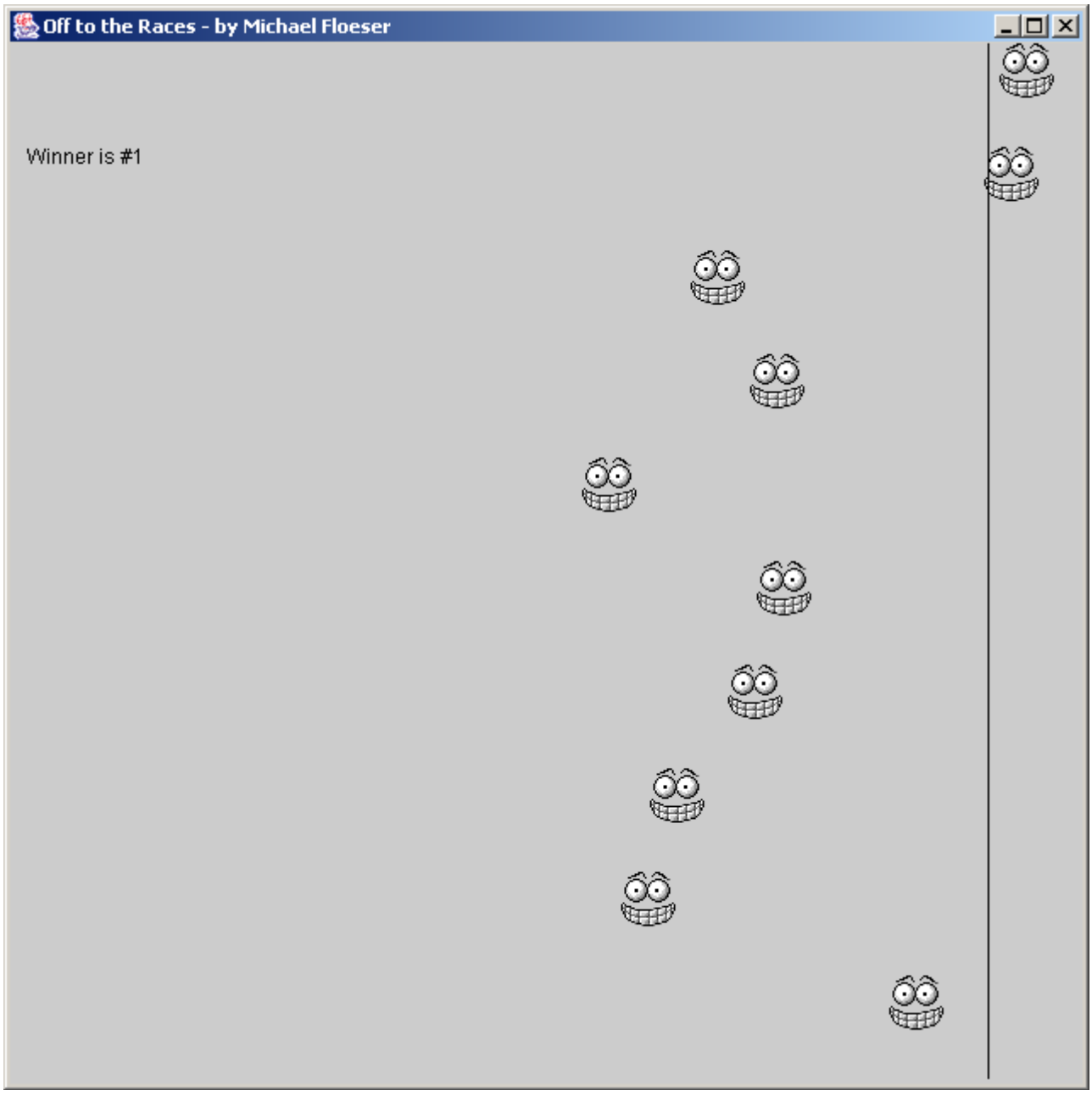
1. Make a thread inner class a JPanel that is Runnable. Add as many JPanel's to the JFrame as you have racers.
2. Add as many threads as required to the JFrame which could be set to GridLayout(racers,1). One cell wide, and "racers" number deep. Could also use 0 here for racers, and let the .add() determine how big it will get
3. Centering your frame on the screen. After setting the frame size and making it visible, execute the JFrame method, setLocationRelativeTo(null).
```
setVisible(true);
setLocationRelativeTo(null);
```
4. Use this statement to get a picture/icon into your program. This is not a typo, it says `Icon` on the left and `ImageIcon` constructor on the right
```
Icon aPic = new ImageIcon("embars.gif");
```
5. To advance the icon/picture across the panel use a Java generated random number between a low and high constant value that you feel will move the image reasonably fast, but not too fast, across the screen. To prevent the images from speeding across the screen, also use a random wait timer (sleep or timer) with some upper and lower limit constants. Each of these two random numbers should be different each time they are needed. Use Math.random() for your random numbers
6. Draw the image icon on the JPanel from within the paint() method using "g" as the "Graphics" object and racePos is the X coordinate of the race, 0 can be the Y. Use aPic Icon from a previous step
```
        //                    X-pos    Y-pos
aPic.paintIcon( this, g, racePos,  0     );
```
7. The paintIcon() method comes from the Icon interface. Read the Icon interface's JavaDocs to find how to get the width and height of an Icon
8. To announce the winner on the panel, display the message using the graphics object, lookup the drawString method in the Graphics class. Also example in ShapeThing.java we covered in class
```
g.drawString("Winner is: "+ threadName, xLoc, yLoc );
```
If you cannot get this to draw on the panel, announcing the winner using JOptionPane will provide 2 points, or announce on the DOS window for 1 point
9. Look at the homework conference for ongoing questions and answers you may have on this and other homeworks

10. You may find your coding will duplicate some operations. Such as getting the width/height of an icon. Sometimes this is unavoidable. Try your best to not duplicate code, but better to get it to work.

Biggest hint of them all – Design globally, build incrementally

Student: *"Professor, it looks like thread 0 won, not thread #1. What's wrong?"*
Professor: *"Nothing may be wrong with your program."*

Notice that icon 0 is closer to the right than icon 1. Icon 1 was found to actually cross the finish line first. This is due to Thread 1 won and was stopped one move before thread 0 knew it was to stop. Between the times thread 1 won and the time it told everyone to stop, thread 0 took one (or two) more moves.
Note: Before your icon moves, have it check to see if anyone else has won the race. This may help prevent this kind of strangeness.



121 Homework 5 – Races (Threads and 2D Graphics) – Gradessheet

Name: _____ **Section:** _____

Grading topics	Max Pts	Pts Earned
Races:		
• **Threads** used, one per racing icon	15	
• Specify number of threads from **command line**.	3	
• When no number is specified, 5 is the default.	3	
• All fixed values (numbers and others) are properly defined as **constants**	3	
• **Calculated** values, such as move-distance, frame width, are based on the size of the image	3	
• Frame is centered in display screen	3	
• Frame title has "project and Your name" in it	3	
• The vertical black line is drawn on the right hand side of the frame, and is continuous, no gaps.	2	
• The finish line is calculated based on a percentage of screen size, or icon width (recommend: about 1.5x to 2x image width from right)	3	
• The screen width must be a calculated value based on the Icon used.	2	
• The screen height is dependent upon how many threads are racing. (For this set each JPanel is set to 1.5 to 2 times the Icon height.)	3	
The JPanel and Runnable class: (can be inner class)		
• JPanel and Runnable for this class	6	
Functionality:		
• Images start on left of screen, races to the right	3	
• Images pause one second before starting	3	
• Image movements are based on random distance calculated from the icon width.	4	
• Image move occurrences are based on a random time wait. (Time is for you to decide.)	4	
• First image across is declared the winner and shown on screen, written in the JPanel. (2-pts for JOptionPane use)	4	
• Use "synchronized" on some simple shared object to block other thread from declaring another winner.	5	
• All other threads are stopped	5	

Grading topics	Max Pts	Pts Earned
Displays and runs as expected.	10	
Style: • JavaDocs for every class and method • Follow coding style: indentation, use of white space for readability – see Coding standards posted in MyCourses conference for this section • Variables of appropriate access (private)	 5 5 3	
Other Extra credit items: Extra: • Menu controls: Restart, choosing number of racers… • Usable controls: such as sliders for speed, etc… • Dynamic graphics drawn vs. icon • Animated (changing) 2D drawn graphic that moves Other items:	 +3…+10+ +3+ +3+ +5+	
Ways to lose points: **(doing these will deduct indicated points)** • Program *not* named **Races.java** • Image file not named races.gif or races.jpg • Using Timers vs. required Threads • Incorrectly sending your work to dropbox • Program does not compile – Grade of zero • Program does not have adequate documentation • JavaDoc warnings • Errors show up on command window • Program shows a run time (execution) error	 -5 -2 -15 -3 -100 -5..-10 -2 -1 to -5 -5	
Total Points	**100**	

Note: JavaDocs are required for this homework

Additional Comments: