RIT | Golisano College of Computing and Information Sciences
School of Information

ISTE-121
HW04 - Off to the Races

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 for 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 screens below** for a better idea of what is happening, or **run my solution** in HW04.jar in the HW04Downloads.zip file on my courses.

Requirements:

1. One AnimationTimer. Behind the scenes, this is a thread that calls a handler that is executed, repeatedly. On each call, you update the animation and return.
2. File containing main() (for me to run) which 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 racers at once, you would use:
   java Races 10 (in Command Prompt or Terminal)
4. **Javadocs** for each class and method, public constants, etc. **are required**.
5. Follow the coding standards for this course that we have discussed in class.
6. Submit work in the Assignment folder. Zip all of your *.java files and any supporting files (i.e. image files) together. Do not include HTML files.
7. All inner classes require the **protected** access modifier. This allows the JavaDocs to be generated for the inner class and its methods.
8. Use a synchronized block or method to declare one winner, and to prevent more than one winner from being announced.
9. Image name must be **races.gif** or **races.jpg** for generated graphic.
10. 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.
11. At race end, leave the frame displaying so the users can see the completed race. Let the user close the application.
12. Finish line must be one continuous non-broken line

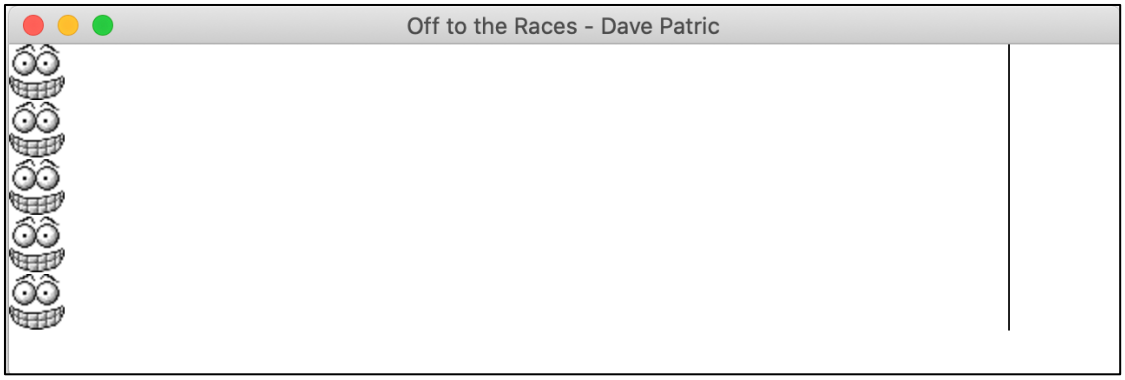
Non-requirements:

1. **UML** is **not** required for this homework.
2. Console output - 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.
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 than I use for testing should result in a different size frame.

Starting race screen requirements:

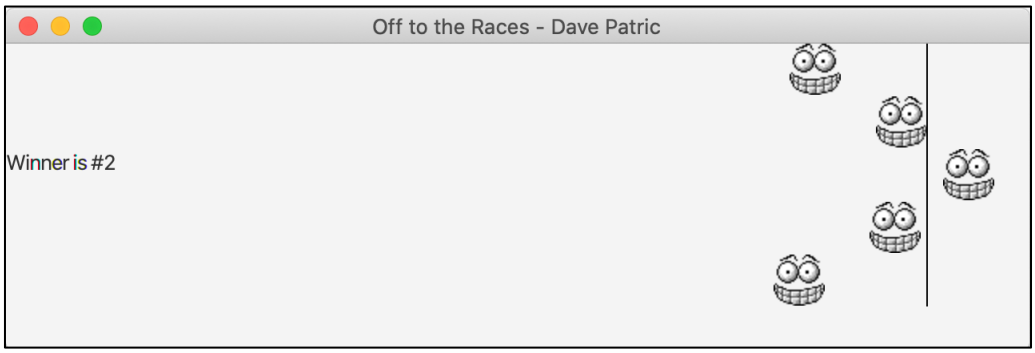
1. The GUI should look like the screen shot below in its starting view and start the application centered in the window.
2. After the screen becomes visible the application pauses for ONE second before starting the race. This is for the user to see it before the icons move.
3. Frame title is to have 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 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 1.5 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, on the left side of the screen.
3. Once a thread is declared the winner, all other threads must be halted at once!



Note: If you run your program several times, you should see the Icon that crosses the finish line first is generally announced the winner (in the console output). 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. 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 an inner class extend Pane to represent each Racer. Add as many Panes to the root as you have racers.
2. Add as many threads as required to the root, which could be a simple VBox() (note, no vertical gap).
3. Use this statement to get a picture/icon into your program:
   ```
   Image aPic = new Image(new FileInputStream(ICON_IMAGE));
   ```
4. To display the icon, make it an ImageView, as in:
   ```
   ImageView aPicView = new ImageView(aPic);
   ```
   This can be displayed with:
   ```
   racer.getChildren().add(aPicView);
   ```
   where `racer` is the Pane. You can also move the icon with:
   ```
   aPicView.setTranslateX(newXPositionOfIcon);
   ```
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. Pass this number to `setTranslateX`.
6. Only add the `ImageView` object to the Pane once. Move it with `setTranslateX()`.
7. To announce the winner in the console output, use `System.out.println`.

8. 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

ISTE-121 HW04 Gradesheet

Off to the Races

Grading topics	Max Pts	Pts Earned
Races:		
• One **AnimationTimer** used for all racers.	15	
• Specify number of threads from **command line**.	4	
• 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.	4	
• 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.	3	
• 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.	3	
• The screen height is dependent upon how many racers there are.	3	
The Pane class: (can be inner class)		
• Pane 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.	5	
• First image across is declared the winner and shown in the console output.	5	
• Use "synchronized" on some simple shared object to block other thread from declaring another winner.	5	
• All other threads are stopped.	5	
Displays and runs as expected.	10	
Style:		
• JavaDocs for every class and method.	5	
• Follow coding style: indentation, use of white space for readability per Coding standards discussed in class.	5	
• Variables of appropriate access (private).	4	

Grading topics	Max Pts	Pts Earned
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 Assignment folder. • 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: