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

ISTE-121

Lab 04: Threads & Progress Bars

Objectives

This lab will have you working with threads, and the progress bar.

Remember, you can use any resource you want for completing the labs. What is not allowed is anything that will violate the RIT or iSchool Academic Dishonesty policy such as copying code.

NOTE: Based on speeds of the computers, and number of processors, the following lab may work differently on different computers. Contact the instructor or TA with questions.

Part 1a - Write a simple Thread

Write a NON-GUI class, Lab04Part1, that has a main, which calls the Lab04Part1 constructor. The constructor creates two instances of an inner class called Lab04Inner that extends the thread class.

- When instantiating the Lab04Inner constructor, pick a name for each thread and pass the name of the thread to the constructor. Have the constructor save this in an attribute.
- The inner class's **run** method must print, "This ran thread " followed by the thread name, then return.

Back in the Lab04Part1 constructor, after instantiating the two threads, start them both. Then have the constructor print "Program finished".

Copy the output to a file named Lab04Part1a.txt.

Part 1b - not so fast, slow down and yield()!

At the beginning of the **run()** method, place a call to **yield()**. What is the output now? (Machines with multiple CPU's may not change any results).

Copy the output to a file named Lab04Part1b.txt.

Part 1c - Calculate something

If everything went as expected, the "Program finished" came out first.

- With a multi-core processor, it possibly didn't come out first.
- To the Lab04Part1 class, add an **int** attribute that will be a counter. Initialize it to zero.
- Change the "Program finished" line to print out "Program finished, count = ", and the counter.
- To the end of the run() method in Lab04Inner, add one to the counter.

Now what is the output?

Copy the output to a file name Lab04Part1c.txt.

Why was the counter zero?

Append your answer at the end of file Lab04Part1c.txt

Part 1d - Wait for the computation to finish, then join the party

To the Lab04Part1 constructor add code between the **start** and print, that makes the code wait until each of the threads has completed its execution. You may **not** use the **sleep()** method.

What was the output now?

Copy the output to a file named Lab04Part1d.txt.

What method did you use to do this?

Append your answer to the end of file Lab04Part1d.txt.

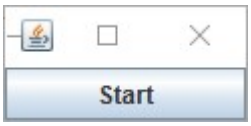
Submit the 4 text files (Lab04Part1a.txt, Lab04Part1b.txt, Lab04Part1c.txt, Lab04Part1d.txt) to the Lab04 Assignment folder.

Part 2 - How much progress are you making?

Upon clicking start, two+ threads update the progress bars based on a random number.

This part of the assignment is to help you better understand threads, progress bars, GUI's, and some other strange GUI/Thread interactions. What is shown in Java can be applied to most programming languages, as many react the same way.

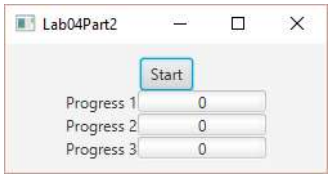
Step 1 - Place Your GUI in the Grid



Create a simple GUI, Lab04Part2, whose root layout is a GridPane. Add one button, with a label you like, to row 0, column 0. This example will use "Start".

Use 2 for your window size.

Step 2 - Create a classy panel that runs



To the GUI you want to add two (or more) progress bar lines as shown. But for the overall program to eventually work, we need to add them in a specific way.

Create an inner class, named InnerProgress that **extends** FlowPane. Create two Labels and a ProgressBar as attributes in this class. Create a StackPane and add the ProgressBar and one of the labels to the StackPane. This Label will be used to display the amount of work done so far, numerically.

Add the other label and the StackPane to the FlowPane you are creating ("this") in its constructor. The label should read "Progress N" where the 'N' is a number passed into the constructor (see example window above).

Since we also need to use threads to move the progress bars, InnerProgress also needs to be used as a thread. How would you do this? Remember, a class can only **extend** one other class.

In a file name Part2.txt, add the line from your code that starts with the following:

class InnerProgress extends FLowPane _____

The constructor for this inner class takes an **int**, which is the line number for this GridPane. Use it to create the Label, such as "Progress N:". Remember, this class **IS-A** FlowPane. Don't write accessors or mutators.

In the Lab04Part2 start method, define and instantiate three InnerProgress objects, passing in the **ints** for line 1, line 2 and line 3. Then add these InnerProgress objects to the GUI. Your GUI should now look like the picture above.

Answer the following question in Lab04Part2.txt:

> Why can we add InnerProgress objects to the GUI?

Step 3 - CLICK THE BUTTON!

Time to make the GUI do something and make sure we have threads that start.

In the button handler, use the InnerProgress objects to create Thread objects, and start them. To the run() method, add a print line that says we are running and include the attribute which is the line # (**int**).

Run the code, to see if you get the threads to print the lines. If you do not, try to first fix it, then contact the instructor or TA. It should be working.

Run the code again, and CLICK THE BUTTON! It should print the two lines every time you click the button.

NOTE: The Thread creation must be within the button click event, because a thread cannot be (re)started once it has finished. This way, new threads are created each time you click the button, and these threads are created from the old FlowPane/Runnable objects, so they are still referring to the original GUI objects.

In the Lab04Part2.txt file, add the following line once it is working:

> I verify the button click properly runs the threads - <your name>

Step 4 - Don't just sit there with a blank stare on your progress bar, move it!

To the run method, add a 'for' loop that counts i from 1 to 100. Within the loop, have it sleep for a random number between 0 and less than 1, multiplied by 100 milliseconds. (Hint: Math class) Then add a `setProgress(i / 100.0);` to set the value of the progress bar, and update the label in the StackPane with `"" + i;`

When the for loop terminates, have it again print the thread name, and the finishing millisecond value. For this time you may use `System.currentTimeMillis()`.

PROBLEM

Doing the setProgress and updating the Label inside the run method changes the UI, so this must be done inside a Platform.runLater(). However, if you use an anonymous inner class for the Runnable, when you try to access 'i' inside the anonymous inner class, you are told you can access local variables (i) only if they are final.

To solve this, create a copy of 'i' that is final, before the Platform.runLater() statement, something like this:

```
final int finalI = i;

Platform.runLater(new Runnable() {
   public void run() {
      pbBar.setProgress(finalI / 100.0);
      lblValue.setText("" + finalI);
    }
 } );
```

Step 5 - When the first progress bar stops, everybody stops

So how do we communicate from within a thread, to the other threads, when we finish running, and have the other threads stop running?

This could be by thread groups, and/or interrupts, but there is a far easier way to do this.

1. Create a boolean attribute called something like *keepGoing* in the Lab04Part2 class, and set it to *true*.
2. To the 'for' loop in the run method, modify the loop so it will continue looping as long as the number is <=100 and *keepGoing* is true.
3. Immediately after the 'for' loop, set *keepGoing* to false.

Compile and run. Click the button. Observe that one progress bar ends at 100%, the others end very near, but less that 100%.

Step 6 - How much has completed before starting? I'm Indeterminate about that

Between when the program starts and the button is clicked, we now want the progress bars to show they are indeterminate. This is an animation.

At the end of the the constructor for InnerProgress, set the value of the ProgressBar to ProgressBar.INDETERMINATE_PROGRESS. This will change when the thread changes the value of the ProgressBar to some other value.

Step 7 - What!?! There's more? - Yes, if your brain isn't full, this should fill it!

Try this, click on the button several times very fast. All those threads are being started, and running at the same time. Fun, isn't it? Do it some more. Wouldn't it be good to disable the button until after all the threads have completed, and then enable it again? Protect the program from users who click the button several times very fast.

To handle this problem, disable the button:

```
btnStart.setDisable(true);
```

in the button handler. To re-enable the button:

```
btnStart.setDisable(false);
```

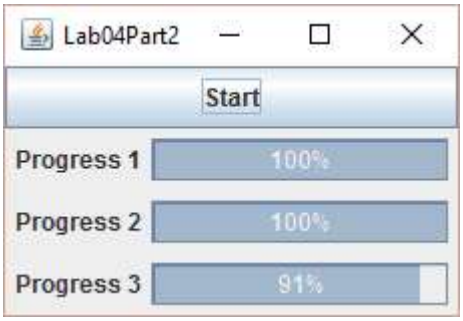
when keepGoing is set to false.

Step 8 - Add 1, 2 or more progress bars, and see what happens.

Let's add more progress bars to the GUI, but not quite yet.

Before adding the code, add your answers to the following to Lab04Part2.txt:

- List all the objects that needed to be created.
- What places need to have code added?



Now add the code and get it working. How close were you to your original thoughts? Notice the items you missed, so you can remember them next time.

Submit your working code (.java file(s) only) to the Lab04 Assignment folder when you have gone as far as you can (partial credit is available).