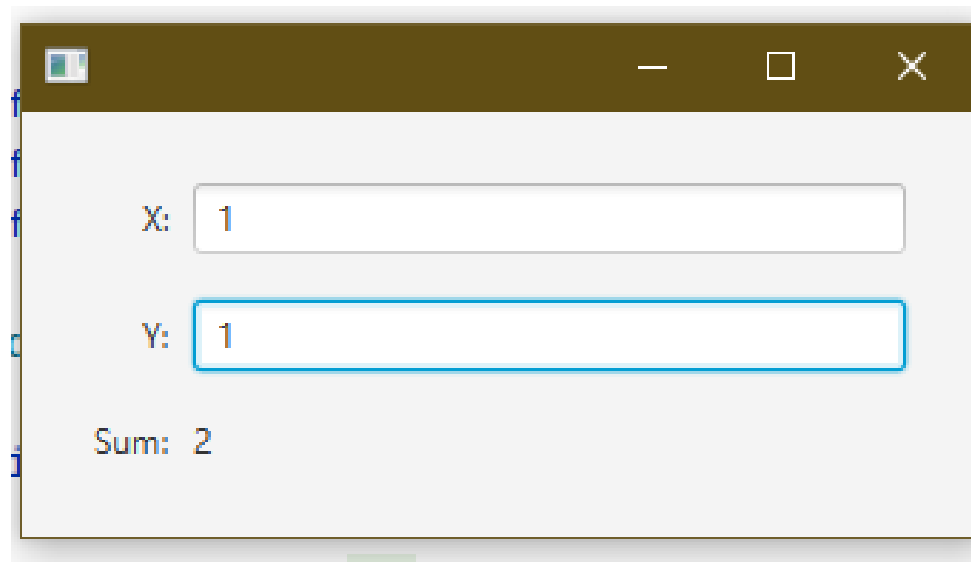


# JavaFX without FXML

# Using JavaFX controller without FXML

- ▶ You can create the classes and tie everything together with listeners.
  - But this is not the best choice
- ▶ You can flow MVC design pattern here as well.
- ▶ We will see an simple example:
  - Using a couple of text fields and a label for displaying their sum.



# JavaFX Concurrency Example

**Reference:** <https://examples.javacodegeeks.com/desktop-java/javafx/javafx-concurrency-example/>

# Introduction

- ▶ The nodes representing UI in a **Scene** Graph are not thread-safe.
  - Advantage: they are faster, as no synchronization is involved.
  - Disadvantage: they need to be accessed from a single thread to avoid being in an illegal state.
- ▶ JavaFX puts a restriction that a live Scene Graph must be accessed from one and only one thread, the JavaFX **Application Thread**.
- ▶ This restriction indirectly imposes another restriction that a UI event should not process a long-running task, as it will make the application unresponsive.
  - The user will get the impression that the application is hung.

# The given Problem

- ▶ The program displays a window as shown in the GUI. It contains three controls.
  - A Label to display the progress of a task
  - A Start button to start the task
  - An Exit button to exit the application
- ▶ When you click the Start Button, a task lasting for 10 seconds is started.
- ▶ The logic for the task is in the runTask() method, which simply runs a loop ten times. Inside the loop, the task lets the current thread, which is the JavaFX Application Thread, sleep for 1 second.

# The given Problem

```
01 import javafx.application.Application;
02 import javafx.event.ActionEvent;
03 import javafx.event.EventHandler;
04 import javafx.scene.Scene;
05 import javafx.scene.control.Button;
06 import javafx.scene.control.Label;
07 import javafx.scene.control.TextArea;
08 import javafx.scene.layout.HBox;
09 import javafx.scene.layout.VBox;
10 import javafx.stage.Stage;
11
12 public class FxConcurrencyExample1 extends Application
13 {
14     // Create the TextArea
15     TextArea textArea = new TextArea();
16
17     // Create the Label
18     Label statusLabel = new Label("Not Started...");
19
20     // Create the Buttons
21     Button startButton = new Button("Start");
22     Button exitButton = new Button("Exit");
23 }
```

# The given Problem

```
24 public static void main(String[] args)
25 {
26     Application.launch(args);
27 }
28
29 @Override
30 public void start(final Stage stage)
31 {
32     // Create the Event-Handlers for the Buttons
33     startButton.setOnAction(new EventHandler <ActionEvent>()
34     {
35         public void handle(ActionEvent event)
36         {
37             runTask();
38         }
39     });
40
41     exitButton.setOnAction(new EventHandler <ActionEvent>()
42     {
43         public void handle(ActionEvent event)
44         {
45             stage.close();
46         }
47     });
48
49     // Create the ButtonBox
50     HBox buttonBox = new HBox(5, startButton, exitButton);
```



# The given Problem

```
52      // Create the VBox
53      VBox root = new VBox(10, statusLabel, buttonBox, textArea);
54
55      // Set the Style-properties of the VBox
56      root.setStyle("-fx-padding: 10;" +
57                  "-fx-border-style: solid inside;" +
58                  "-fx-border-width: 2;" +
59                  "-fx-border-insets: 5;" +
60                  "-fx-border-radius: 5;" +
61                  "-fx-border-color: blue;");
62
63      // Create the Scene
64      Scene scene = new Scene(root, 400, 300);
65      // Add the scene to the Stage
66      stage.setScene(scene);
67      // Set the title of the Stage
68      stage.setTitle("A simple Concurrency Example");
69      // Display the Stage
70      stage.show();
71  }
```



# The given Problem

```
73 public void runTask()  
74 {  
75     for(int i = 1; i <= 10; i++)  
76     {  
77         try  
78         {  
79             String status = "Processing " + i + " of " + 10;  
80             statusLabel.setText(status);  
81             textArea.appendText(status+"\n");  
82             Thread.sleep(1000);  
83         }  
84         catch (InterruptedException e)  
85         {  
86             e.printStackTrace();  
87         }  
88     }  
89 }  
90 }
```

# The given Problem

- ▶ The program has two problems. The first one:
  - Click the **Start Button** and immediately try to click the **Exit Button**.
  - Clicking the Exit Button has no effect until the task finishes. Once you click the Start Button, you cannot do anything else on the window, except to wait for 10 seconds for the task to finish.
- ▶ The second one:
  - Inside the loop in the **runTask()** method, the program prints the status of the task on the standard output and displays the same in the Label in the window. You **don't see the status updated** in the Label.

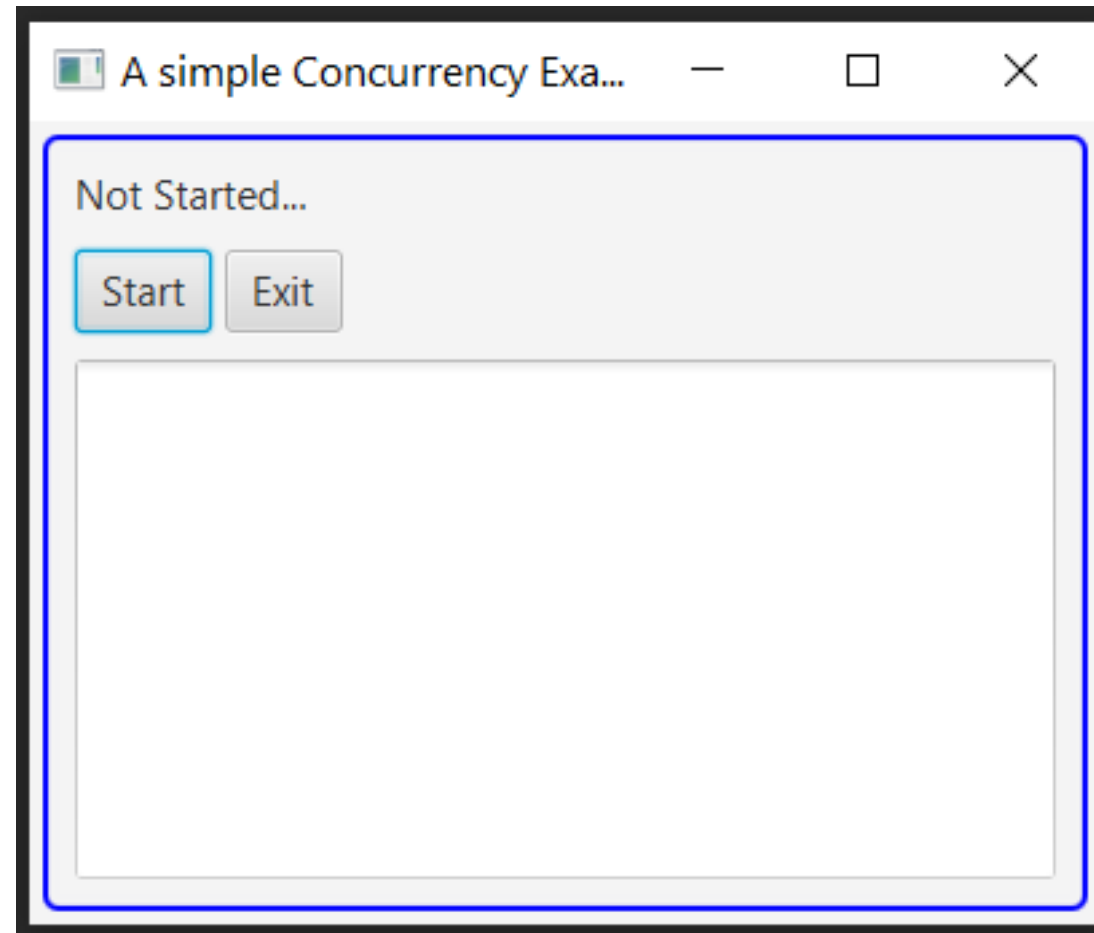
# What happened?

- ▶ All UI event handlers in JavaFX run on a single thread, which is the JavaFX Application Thread.
- ▶ When the Exit Button is clicked while the task is running, an `ActionEvent` event for the Exit Button is generated and queued on the JavaFX Application Thread.
  - The `ActionEvent` handler for the Exit Button is run on the same thread after the thread is done running the `runTask()` method as part of the `ActionEvent` handler for the Start Button.
- ▶ A pulse event is generated when the Scene Graph is updated.
  - The pulse event handler is also run on the JavaFX Application Thread.
  - Inside the loop, the text property of the Label was updated ten times, which generated the pulse events.

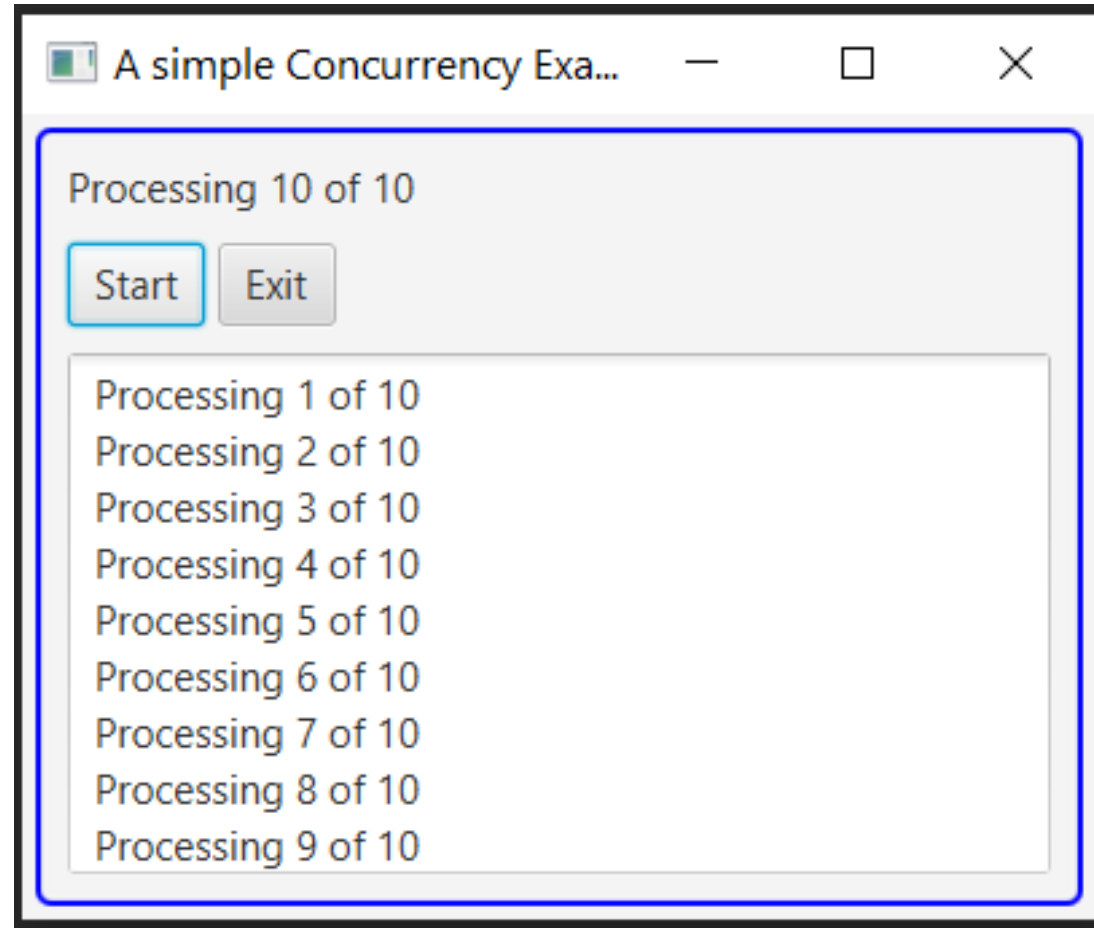
# What happened?

- ▶ However, the Scene Graph was not refreshed to show the latest text for the Label
  - The JavaFX Application Thread was busy running the task and it did not run the pulse event handlers.
- ▶ Both problems arise because there is only one thread to process all UI event handlers and you ran a long-running task in the ActionEvent handler for the Start button.
- ▶ What is the solution?
  - You cannot change the single-threaded model for handling the UI events.
  - You must not run long-running tasks in the event handlers.
  - **Run the long-running tasks in one or more background threads**, instead of in the JavaFX Application Thread.

# The GUI after clicking the Start Button



# The GUI after execution



# A first Solution Approach

```
029  @Override
030  public void start(final Stage stage)
031  {
032      // Create the Event-Handlers for the Buttons
033      startButton.setOnAction(new EventHandler <ActionEvent>()
034      {
035          public void handle(ActionEvent event)
036          {
037              startTask();
038          }
039      });
040
041      exitButton.setOnAction(new EventHandler <ActionEvent>()
042      {
043          public void handle(ActionEvent event)
044          {
045              stage.close();
046          }
047      });
048
049      // Create the ButtonBox
050      HBox buttonBox = new HBox(5, startButton, exitButton);
```



# A first Solution Approach

```
073     public void startTask()
074     {
075         // Create a Runnable
076         Runnable task = new Runnable()
077         {
078             public void run()
079             {
080                 runTask();
081             }
082         };
083
084         // Run the task in a background thread
085         Thread backgroundThread = new Thread(task);
086         // Terminate the running thread if the application exits
087         backgroundThread.setDaemon(true);
088         // Start the thread
089         backgroundThread.start();
090     }
```

# A first Solution Approach

```
092     public void runTask()  
093     {  
094         for(int i = 1; i <= 10; i++)  
095         {  
096             try  
097             {  
098                 String status = "Processing " + i + " of " + 10;  
099                 statusLabel.setText(status);  
100                 textArea.appendText(status+"\n");  
101                 Thread.sleep(1000);  
102             }  
103             catch (InterruptedException e)  
104             {  
105                 e.printStackTrace();  
106             }  
107         }  
108     }
```

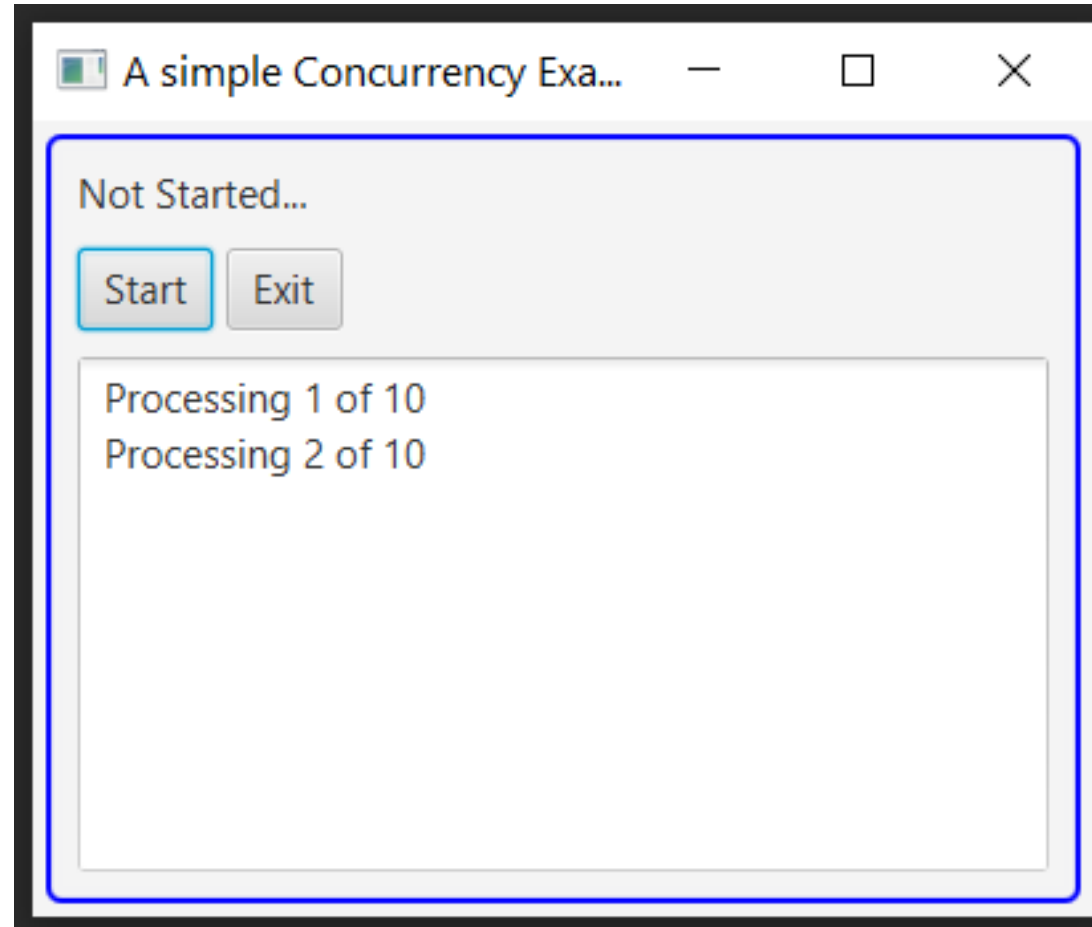
# A first Solution Approach

- ▶ The above program is an incorrect attempt to provide a solution.
  - The `ActionEvent` handler for the Start Button calls the `startTask()` method, which creates a new thread and runs the `runTask()` method in the new thread.
- ▶ Run the program and click the Start Button.
  - A runtime exception is thrown as follows:

```
Exception in thread "Thread-4" java.lang.IllegalStateException: Not on FX application thread; currentThread = Thread-4
at com.sun.javafx.tk.Toolkit.checkFxUserThread(Toolkit.java:236)
at com.sun.javafx.tk.quantum.QuantumToolkit.checkFxUserThread(QuantumToolkit.java:423)
at javafx.scene.Parent$2.onProposedChange(Parent.java:367)
at com.sun.javafx.collections.VetoableListDecorator.setAll(VetoableListDecorator.java:113)
at com.sun.javafx.collections.VetoableListDecorator.setAll(VetoableListDecorator.java:108)
at com.sun.javafx.scene.control.skin.LabeledSkinBase.updateChildren(LabeledSkinBase.java:575)
...
```

- ▶ The following statement in `runTask()` method generated the exception:
  - `statusLabel.setText(status);`

# The GUI after execution



# A first Solution Approach

- ▶ The JavaFX runtime checks that a live scene must be accessed from the JavaFX Application Thread.
- ▶ The `runTask()` method is run on a new thread, named Thread-4 as shown in the stack trace, which is not the JavaFX Application Thread.
- ▶ How do you access a live Scene Graph from a thread other than the JavaFX Application Thread?
  - The simple answer is that you cannot.
  - The complex answer is that when a thread wants to access a live Scene Graph, it needs to run the part of the code that accesses the Scene Graph in the JavaFX Application Thread.

# The Solution

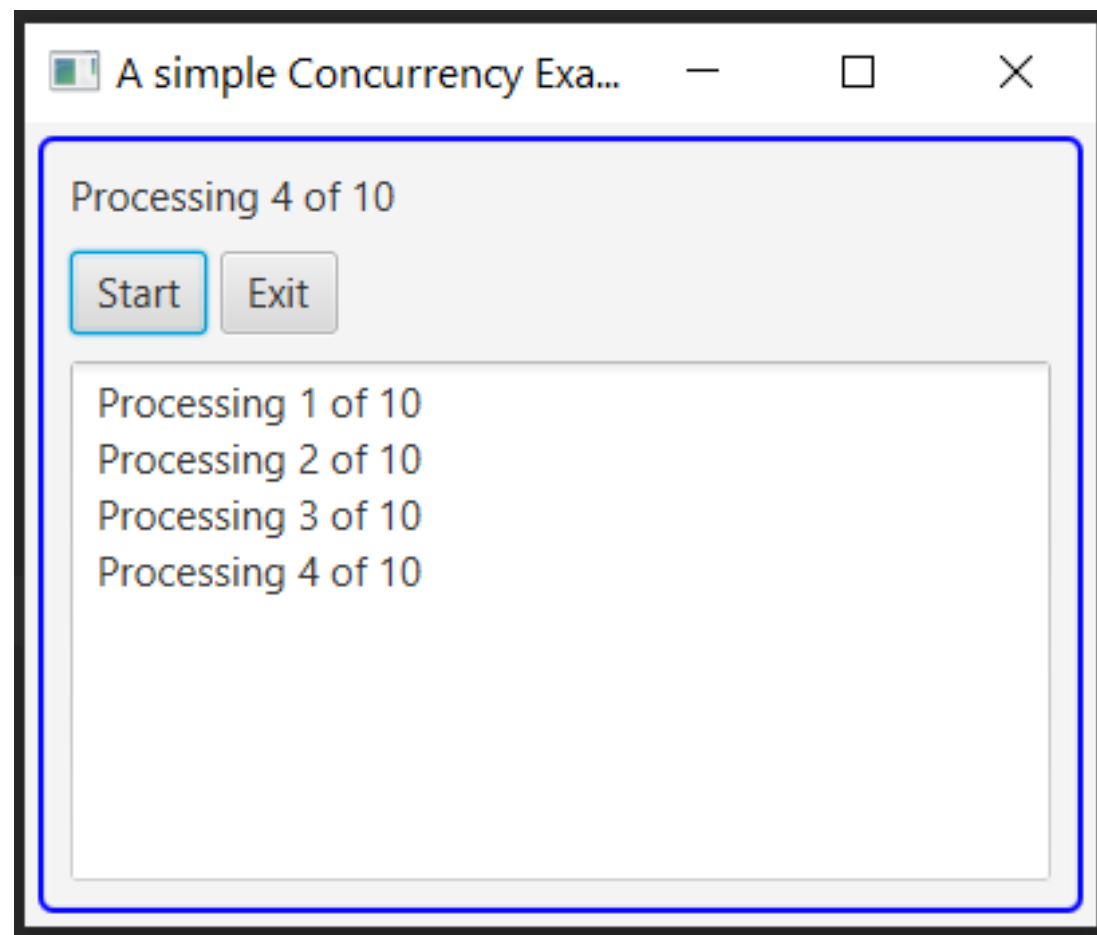
- ▶ The Platform class in the javafx.application package provides two static methods to work with the JavaFX Application Thread.
  - public static boolean isFxApplicationThread()
  - public static void runLater(Runnable runnable)
- ▶ The **isFxApplicationThread()** method returns true if the thread calling this method is the JavaFX Application Thread. Otherwise, it returns false.
- ▶ The **runLater()** method schedules the specified Runnable to be run on the JavaFX Application Thread at some unspecified time in future.

# The Solution

```
093 public void runTask()  
094 {  
095     for(int i = 1; i <= 10; i++)  
096     {  
097         try  
098         {  
099             // Get the Status  
100             final String status = "Processing " + i + " of " + 10;  
101  
102             // Update the Label on the JavaFx Application Thread  
103             Platform.runLater(new Runnable()  
104             {  
105                 @Override  
106                 public void run()  
107                 {  
108                     statusLabel.setText(status);  
109                 }  
110             });  
111  
112             textArea.appendText(status+"\n");  
113  
114             Thread.sleep(1000);  
115         }  
116         catch (InterruptedException e)  
117         {  
118             e.printStackTrace();  
119         }  
120     }  
121 }
```



# The GUI after execution



# The Summary

- ▶ Did you overcome the restrictions imposed by the event-dispatching threading model of the JavaFX?
- ▶ The answer is yes and no.
  - You used a trivial example to demonstrate the problem.
  - You have solved the trivial problem.
  - However, in a real world, performing a long-running task in a GUI application is not so trivial.
- ▶ Your task does not return a result, nor does it have a reliable mechanism to handle errors that may occur.

# The Summary

- ▶ Your task cannot be reliably cancelled, restarted, or scheduled to be run at a future time.
- ▶ The **JavaFX concurrency framework** has answers to all these questions.
  - The framework provides a reliable way of running a task in one or multiple background threads and publishing the status and the result of the task in a GUI application.
- ▶ More information:
  - <https://examples.javacodegeeks.com/desktop-java/javafx/javafx-concurrent-framework/>