Instructions to run my code:

```
// compile
javac -cp :../lib/*: myGUI.java

// execute
java -cp :../lib/*: myGUI
```

Instructions to run my tests:

```
// compile all tests
javac -cp :../lib/*: *_Test.java

// run specific test
java -cp :../lib/*: org.junit.runner.JUnitCore SimplifiedTrackers_Test
java -cp :../lib/*: org.junit.runner.JUnitCore AllEvents_Test
```

How I tested these components, and why I chose to test it like that:

SimplifiedTrackersComponent

This component of my GUI displays, for each tracker, simplified GPS events by stripping their altitude information. To strip the altitude information from each GPS event, I created a function that converts a stream of GpsEvents to a stream of SimpleGpsEvents by mapping a GpsEvent object to a new SimpleGpsEvent object, created with that initial event.

Since we can check the SimpleGpsEvent class file itself to see that there is no altitude field, I decided that I only needed to test that the function correctly carries over the fields. I suppose it also checks that none of the fields were set to the altitude as well. Testing this function was done by creating a StreamSink that we could send an event to later, and the converted SimpleGpsEvent stream. I then set up cells for each of the desired fields, sent the event, and finally compared the event's information to the cell's sampled information.

Now that we can see that the information is correct, I wanted to test whether the information was being displayed in the correct output. To do this, I created an array of GpsEvent streams and a SimplifiedTrackersComponent object. I then sent down two events in separate streams. I have written my program such that the trackers are in order and that the cells used in the display are stored, so when sending events down an array of two streams, there should be two groups of cells. By checking the first two group of cells and comparing their values to the event downstream, I was able to verify that the GpsEvent data were being outputted to the correct sub-displays. Viewing the display also shows that the trackers are in ascending order.

AllEventsComponent

This component of my GUI displays each GpsEvent as it passed to the GUI, at the time it occurs. To meet the spec, it outputs the event as a comma-delimited string of 4 items, presenting only one event at a time and clearing that event after about 3 seconds if it is not overwritten by another event. To display every event in the desired format, I merged the streams into one using the Sodium function orElse() and mapped the GpsEvent to a string by modifying its toString() function.

To clear the eventString after about 3 seconds, I used a periodic timer to check whether enough time has passed since the last event occurred. A more efficient way to implement this is to have a timer set for 3 seconds after each event occurs, clearing the eventString if another event doesn't come in and cancel that timer. I believe this can be done with Sodium but there was not an example I could try learning from in the book. Thus, I prioritized a working solution. The eventString uses orElse() to prioritize events from the merged streams, but accept events from the clearStream (which fires an empty string after 3 seconds of no events). The clearStream is set up to only fire one empty string once its conditions are met, to avoid unnecessary firings.

To test that the streams are merged correctly and that the display is receiving events from it correctly, I created StreamSinks and sent events down them, checking that the eventString cell updates to the expected value. I didn't create a test to explicitly check that only one event is shown at a time, but since there is only one cell being displayed and that cell is being checked anyway, it isn't necessary. Also, Sodium's orElse() guarantees only one event is sent at a time anyways.

To test that the eventString cell is cleared after 3 seconds, I sent an event down a StreamSink, checked that the cell has a value before the 3 seconds pass, and then checked that its value is the empty string after a total of ~3 seconds pass. I used Thread.sleep() to make the test wait for the specified times.