Exercises week 9

Last update 2020-10-21

Goals

The goals of this week is that you get:

- a basic understanding of *reactive programming* and some
- experience with RxJava
- some experience with Java Swing (to illustrate the reactive concepts)
- to write Java programs using RxJava (and Swing).

Do this first

The exercises rely on having an RxJava library installed on your computer. This can be done in many ways depending on the Java setup/IDE you are using. This tutorial shows you how to make an RxJava setup: `https://www.tutorialspoint.com/rxjava/rxjava_environment_setup.htm`.

The lecture and these exercises are based on JavaRx version 2 (libraries rxjava-2.2.4 and reactive-streams-1.0.2).

Some of the exercises you will work with simple Swing based user interface based on Java `https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html`. If you are not familiar with Swing, you may find an introduction here: `https://www.javatpoint.com/java-swing`.

Exercise 9.1 In the file `Code/Stopwatch.java` you find a complete Java version of the stopwatch example used in the lecture and material for this week.

1. *Green* Revise the stopwatch, so it can measure 1/10 th of a second.
2. Make a version (Stopwatch2) that has two independent stopwatches, each with their own buttons and display.
3. *Yellow* Make a version (StopwatchN) that have n independent stopwatches, each with their own buttons and display. Choose n, so one row of stopwatches fit on your screen.

Exercise 9.2 In week 3 (exercise 3.1) you made a bounded buffer with this interface:

```
    interface BoundedBuffer<T> {
    void insert(T elem);
    T take();
}
```

In this exercise you will create and use a modification of the BoundedBuffer to create a class for message passing with this interface:

```
    interface MessageBuffer<T> {
    void sendMessage(T elem);
    T receiveMessage();
}
```

1. *Green* Make an implementation of the `MessageBuffer`.

2. Make a version of the Stopwatch in `Code/Stopwatch.java` using message passing between the thread running the clock (called `t` in the code) and the (main) thread updating the display. The two threads should *not* share any objects (other than an instance of `MessageBuffer`).
3. *Yellow* Make a version of the stopwatch based on message passing that have two independent stopwatches, each with their own buttons and display.
4. *Red* Make a version of the stopwatch based on message passing that have n independent stopwatches, each with their own buttons and display. Choose n, so one row of stopwatches fit on your screen.

Exercise 9.3 This exercise makes sure that you have a working version of RxJava and is able to use it to run a few simple examples.

1. *Green* Make sure you can run the simple examples in steps 6 and 7 from:
`https://www.tutorialspoint.com/rxjava/rxjava_environment_setup.htm`.
Make sure that you get the same result as in the tutorial.
2. Run the example from:
`https://www.tutorialspoint.com/rxjava/rxjava_single_observable.htm`.
Make sure that you get the same result as in the tutorial.
3. *Yellow* Run the example:
`https://www.tutorialspoint.com/rxjava/rxjava_from_scheduler.htm`
Write down your own explanation of what happens in this example.

Exercise 9.4 In this example you should use the RxJava concepts to make some versions of a stopwatch. In the file `Code/StopwatchRx.java` you will find (most of) the code for a RxJava based version of the stopwatch.

1. *Green* Replace the line //TO-DO in `Code/StopwatchRx.java` with code that uses the Rx classes (`display` and `timer`) to make a working version of `StopWatchRx`.
2. *Yellow* Revise the code from the first step of this exercise so that all buttons are made into observables. (Hint: You may use `Code/rxButton.java` as an inspiration.

Exercise 9.5 In this exercise you should make an RxJava based solution of (part of) exercise 6.2 from week 6.

1. *Green*
   (a) Make an observable `Observable<String> readWords` that can read the file `english-words.txt` file. It should override:
   `public void subscribe(ObservableEmitter<String> s)` so that each `s.onNext` provides the next line from `english-words.txt`.
   (b) Make an observer `Observer<String> display= new Observer<String>()` that will print the word emitted from `Observable<String> readWords` i.e. one string every time `onNext` is called.
   (c) Write a Java program that prints the first 100 word from `english-words.txt` using the the observable `readWords` and the observer `display`.
2. *Yellow* Write a Java program to find and print all words that have at least 22 letters.
3. *Red* Write a Java program to find all palindromes and print them (use the `isPalindrome`) method from Exercise 6.2.