Hints on Decoupling Models and Views

Purpose:

This handout is a note on how one can use the `java.util.function.*` classes to decouple connections between a model and a view.

Implementing the changes suggested here are *not* required for Lab 7.

Method

The goal is to redesign the Register class from Lab 7 slightly, so that it can be connected to pretty much anything.

The main technical trick is to use a "λ-expression" to provide the connection between the register and anything that wants to listen to its changes. The λ-expression is passed to the `Register`.

- The interface that we wish to use comes from the `java.util.function` package:

```
import java.util.function.Consumer;
```

- Next, we want to add a `private Consumer<String> sink` member variable, together with a public `setRegisterListener` setter (and optionally a getter). The `setRegisterListener` method is somewhat like the `addActionListener` of buttons, *et cetera*. However we keep things simple and only allow one listener.
- The `sink` member variable is a `Consumer<String>` object, so has a `void accept(String s)` method. We use it to create a private utility method `update`

```
  private void update() { sink.accept(getDisplayText()) ; }
```

Now whenever `update()` is called, the `Register` listener gets the new text to display.

- We then add the `update()` to the end of every behaviour that changes the `Register` text.
- We must make sure that `Register` objects always have a legitimate `sink` value. We can have the constructor set it to a *do-nothing* value.

```
  ...
  setRegisterListener((s)->{}) ;
```

Linking the `Register` to something

- Suppose that we want to link a `Register` to `JTextField` that it is to display its value. We can do this with code like

```
  private Register myRegister ;
  private JTextField myDisplay ;
  ...
  // in wiring code
     myRegister.setRegisterListener((str)->myDisplay.setText(str)) ;
```

Now the `myDisplay` variable will update whenever the `myRegister` variable is changed.

Review

What have we accomplished?

There is a lot less **coupling**. The listener (here a `JTextField`) knows *nothing* about its model. The `Register` objects only rely on the general purpose `java.util.function.Consumer` interface, and can be connected to *any* other component through a λ-expression.

Finally, a `setRegisterListener` method seems to be cohesive with the general purpose of a `Register` model.