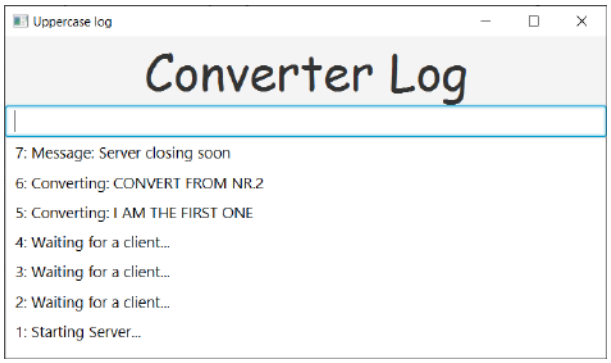
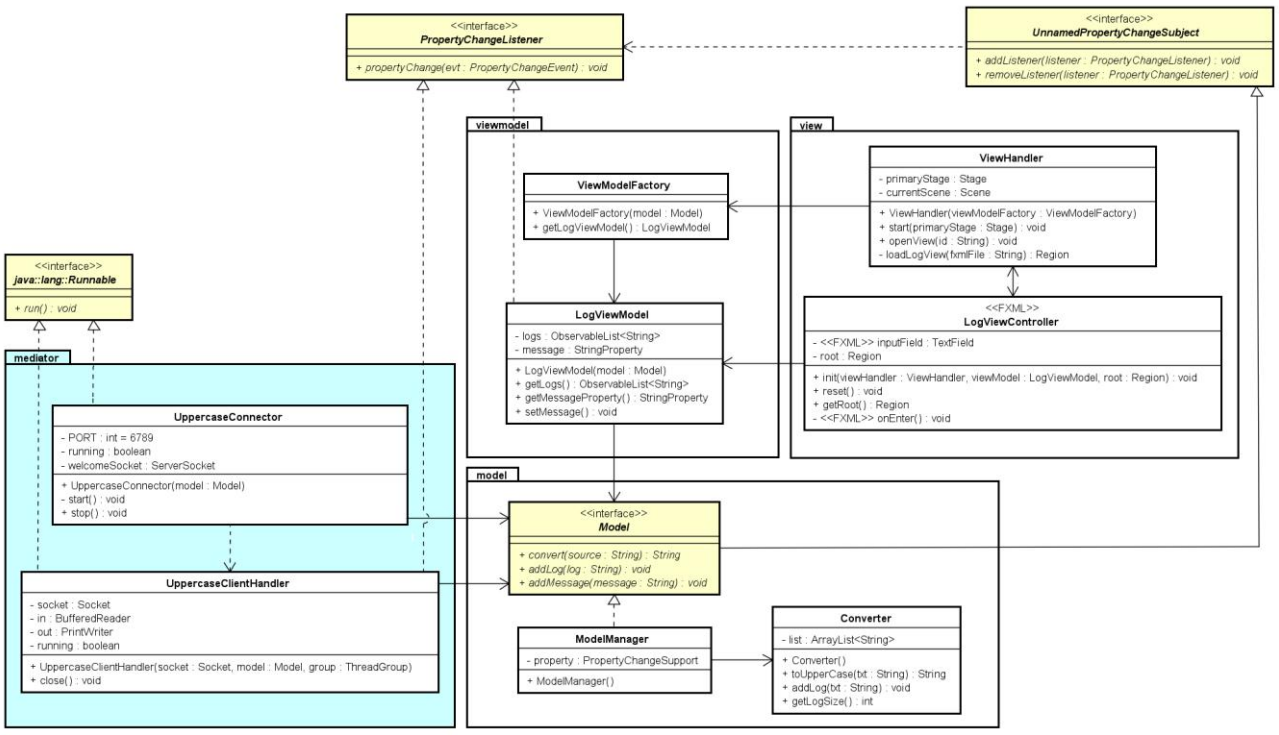
Exercise: MVVM-Uppercase-Server

A simple console server is given in the file `Sockets-Uppercase-Server.zip`. The server is a multithreaded TCP socket server on port `6789`, it accepts multiple incoming clients and repeatedly reads a `String` from a `BufferedReader` stream and reply with an upper case `String` in a `PrintWriter` stream.

The purpose for this exercise is to apply the MVVM design pattern to the server side application. See The Server window and the UML class diagram below.

Log 1+2: starting server and waiting for the first client
Log 3+4: two clients logged on
Log 5+6: each client converts a string to uppercase
Log 7: Server adds a message





Step 1: Create a new module in IntelliJ and copy your solution to the two window version of the MVVM-Uppercase exercise (the version where you have a window with the log).

Step 2: Delete all relations to the first window, i.e. delete `ConvertView.fxml`, `ConvertViewController`, and `ConvertViewModel`. Modify `ViewHandler` and `ViewModelFactory` accordingly, i.e. removing all relation to the deleted classes and starting up the log view instead of the missing convert view.

Modify the `LogView.fxml` removing the Back-button and adding a text field to add a message (e.g. in Scene Builder). Alternatively, use the FXML file shown at the end of this document.

Step 3: Create an extra method in the Model (`addMessage`) representing a message typed in the text field. Therefore, also update the ViewModel (with a StringProperty) and the ViewController (with the binding to the text field). The `addMessage` in the ModelManager simply fires an event (which will be seen in the ClientHandler (to be send to the client, i.e. a broadcast). It is ok to change the subject interface to a NamedPropertyChangeSubject such that you only get the event for broadcasts.

Step 4: Copy classes `UppercaseConnector` and `UppercaseClientHandler` from the upload and include in the same project (in package `mediator`)

Step 5: Modify method `start` in class `MyApplication` (in the default package)

- After creating Model, ViewModel and View and starting the View, insert statements to create a server thread and start the thread (`UppercaseConnector`)

Step 6: Run the server and test with the client you made in the previous exercise

LogView.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>

<VBox stylesheets="view/Layout.css" maxHeight="-Infinity" maxWidth="-Infinity"
      minHeight="-Infinity" minWidth="-Infinity" prefHeight="351.0"
      prefWidth="600.0" xmlns="http://javafx.com/javafx/10.0.1"
      xmlns:fx="http://javafx.com/fxml/1"
      fx:controller="view.LogViewController" userData="Uppercase log">
    <children>
        <HBox alignment="TOP_CENTER">
            <children>
                <Label text="Converter Log">
                    <font>
                        <Font name="Comic Sans MS" size="48.0"/>
                    </font>
                </Label>
            </children>
        </HBox>
        <TextField fx:id="inputField" onAction="#onEnter">
            <font><Font size="14.0"/></font>
        </TextField>
        <ListView fx:id="logList" editable="false" prefHeight="258.0"
                  prefWidth="554.0"/>
    </children>
</VBox>
```

Layout.css

```
.list-view:disabled {
    -fx-opacity: 1;
}

.list-cell {
    -fx-font-size:15.0;
    -fx-text-fill: black;
```

```
    -fx-control-inner-background: white ;
    -fx-control-inner-background-alt: derive(-fx-control-inner-background, 50%);
}

.list-cell:filled:selected:focused, .list-cell:filled:selected,
.list-cell:even, .list-cell:odd {
    -fx-background-color: white;
}
```