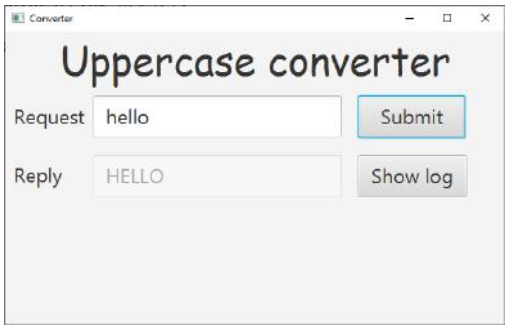
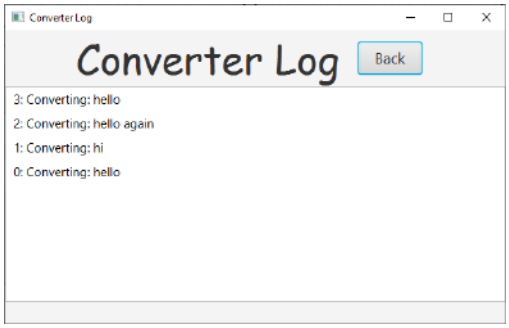
Exercise: MVVM - Uppercase and log

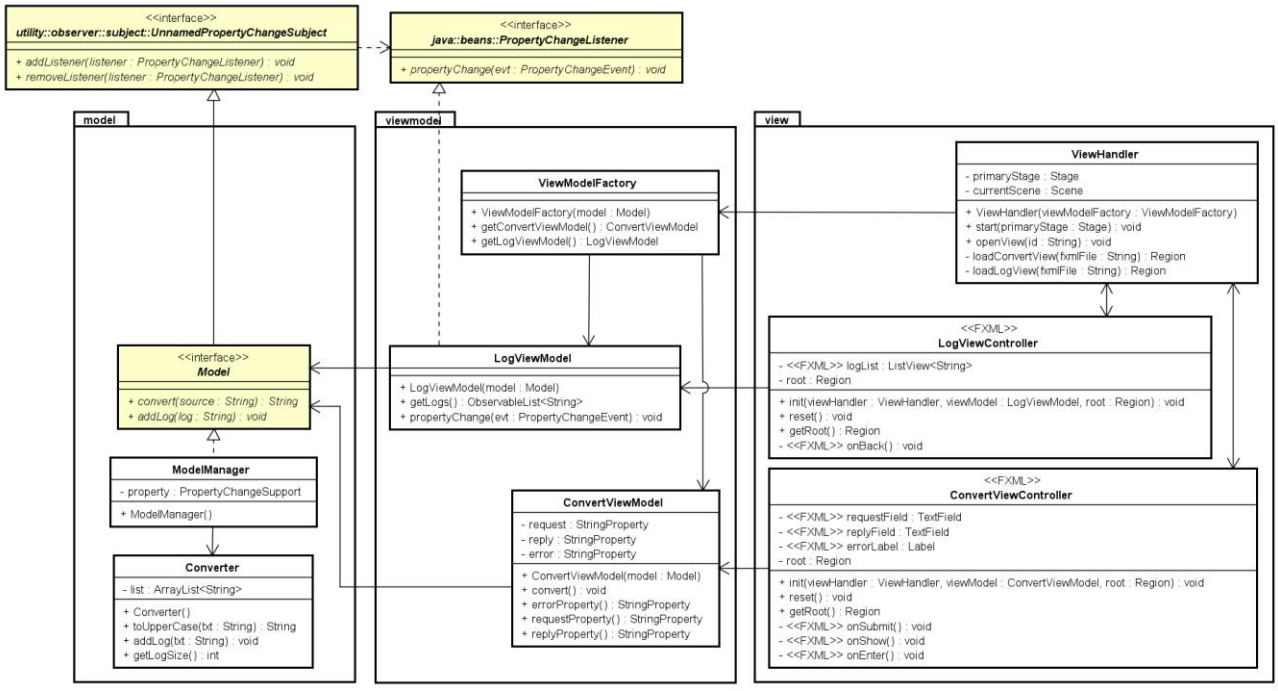
The purpose of this exercise is to create an MVVM application with two windows, one window with the ability to convert a string to uppercase and one window showing the log.





Note that a lot of the exercise can be taken from your solution to a previous exercise about a one window application converting to uppercase.

Implement the MVVM application exactly as shown in the class diagram below (description is given below and on the next page).



Model:

- Step 1 – just copy from your previous exercise about the one window uppercase application.
- Step2 – modify the Model to be a Subject in the Observer design pattern (i.e. let `Model` extend interface `UnnamedPropertySubject` and let `ModelManager` implement the two methods from the interface.
    - First: declare a `PropertyChangeSupport` object as an instance variable,
    - second: initialize it in the constructor creating a `PropertyChangeSupport` with itself (`this`) as argument to the constructor,

- third: delegate to this object implementing the `addListener` and `removeListener` methods from the interface,
- and last: fire an event in the `addLog` method.

ViewModel:

Modify class `ConvertViewModel`

- Add a void method `clear` to set the properties to `null` strings (or empty strings). Note that this method is not shown in the class diagram.

Create a class `LogViewModel`

- `Model` instance variable
- An `ObservableList<String>` instance variable: `logs`
- A constructor setting the `Model` instance variable to the parameter variable and initializing the `ObservableList` (i.e. `logs = FXCollections.observableArrayList();`)
- A get method for the `ObservableList` instance variable
- Observer pattern: Let the class implement interface `PropertyChangeListener` overriding method `propertyChange(PropertyChangeEvent evt)` with the following
  - Call `add` for the `ObservableList` instance variable with `evt.getNewValue()` converted to a `String`
  - It is not needed in this exercise, but it is a good practice to wrap the statement(s) into a `Platform.runLater` when the method potentially could be called from outside the JavaFX thread, i.e. the following

```
Platform.runLater(() -> {
  // statements updating properties bound to JavaFX components
});
```

Update class `ViewModelFactory`

- Instance variables for both of the viewModels (`LogViewModel` and `ConvertViewModel`)
- A constructor taking the `Model` and creating both viewModels objects
- Getters for both viewModels / instance variables

View:

Modify `ConvertView.fxml` such that you add another Button (with the text: `Show log`). Add an onAction method to the Button and call this method `onShow`.

Modify class `ConvertViewController`

- An @FXML annotated method `onShow` calling a method in the `ViewHandler` to open the Log window (method `openView` with the string `"log"` as argument)
- Modify method `reset` to call the method `clear` in the ViewModel

FXML file `LogView.fxml` is given in appendix. Copy this to the view package.

Create a class `LogViewController`

- An @FXML annotated instance variable `logList` of type `ListView<String>`
- Instance variables for `root` (type `Region`), `viewModel` (`LogViewModel`) and `ViewHandler`

- An `init` method setting the non-FXML instance variables, and perform a 'binding' to the viewModels Observable List the following way
  `logList.setItems(viewModel.getLogs());`
- Getter for `root`
- An @FXML annotated method `onBack` calling a method in the `ViewHandler` to open the convert window again, i.e. `openView("convert")`

Modify class `ViewHandler`

- Instance variables for `Stage`, `Scene`, `ConvertViewController` , `LogViewController` and `ViewModelFactory`
- Modify the private method `loadConvertView` such that you only load the FXML file, get the controller and call `init`, if instance variable of type `ConvertViewController` is `null` and if not (in the else-part) you call `reset`.
  Note: You have to call the method in the `ViewModelFactory` to get the `ConvertViewModel` needed in the `init` method
- Create a private method `loadLogView` such that you load the FXML file, get the controller (initialise `LogView` instance variable) and call `init` - but only if instance variable of type `LogViewController` is `null`. Structural identical to the other private load-method.
- Update method `openView` with an extra case `"log"` in the switch, to open the log view window.
- Optional: To style the output, you may copy (into the view folder), the CSS files `Layout.css` shown below – and add the following to the root node in `LogView.fxml`:
  `stylesheets="view/Layout.css"`

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;
}
```

Run the application.

Appendix: FXML file: LogView.fxml

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

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

<VBox 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="Converter Log">
    <children>
        <HBox alignment="TOP_CENTER">
            <children>
                <Label prefHeight="67.0" prefWidth="322.0" text="Converter Log">
                    <font>
                        <Font name="Comic Sans MS" size="48.0"/>
                    </font>
                </Label>
                <Pane prefHeight="67.0" prefWidth="109.0">
                    <children>
                        <Button layoutX="16.0" layoutY="14.0"
                                mnemonicParsing="false" onAction="#onBack"
                                prefHeight="39.0" prefWidth="77.0" text="Back">
                            <font>
                                <Font size="18.0"/>
                            </font>
                        </Button>
                    </children>
                </Pane>
            </children>
        </HBox>
        <ListView fx:id="logList" editable="false" prefHeight="258.0"
                  prefWidth="554.0"/>
    </children>
</VBox>
```