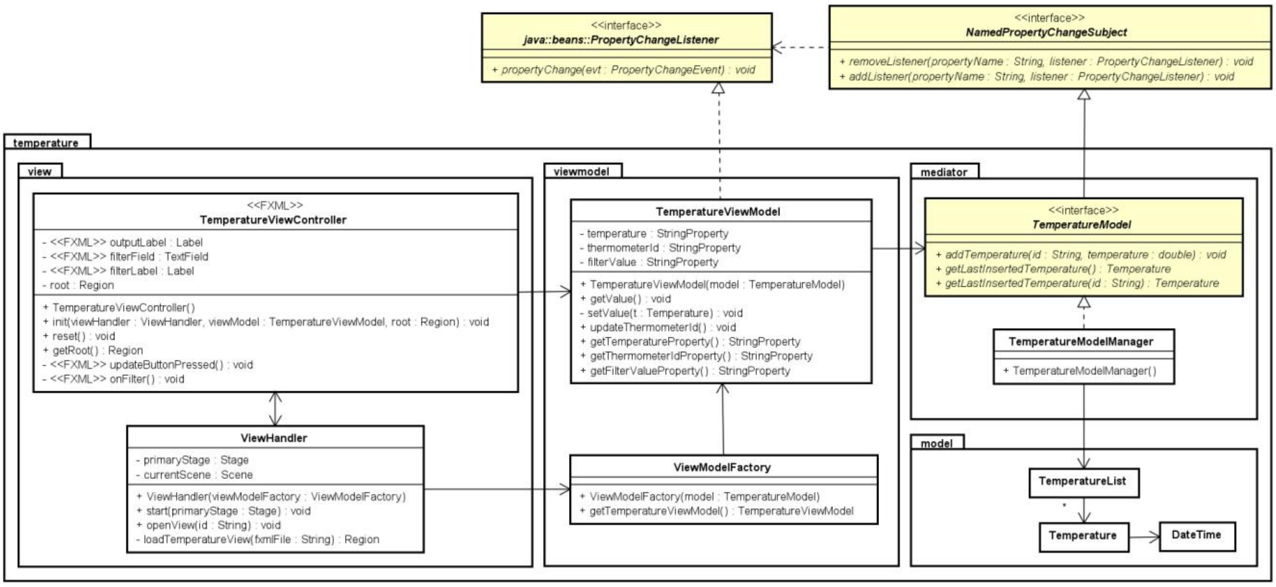
A JavaFX temperature presenter (the Thermometer exercise) using the MVVM pattern

In this exercise, you are going to transform your solution to the temperature presenter (*the exercise presenting temperatures generated from external Thermometer threads)*, such that this follows the MVVM pattern.

Your solution may reflect the following class diagram but you are free to define other attributes and methods. (Note that the Thermometer updating the model is not shown)



<<interface>>
java::beans::PropertyChangeListener
+ propertyChange(evt : PropertyChangeEvent) : void
<<interface>>
NamedPropertyChangeSubject
+ removeListener(propertyName : String, listener : PropertyChangeListener) : void
+ addListener(propertyName : String, listener : PropertyChangeListener) : void
temperature
view
<<FXML>>
TemperatureViewController
- <<FXML>> outputLabel : Label
- <<FXML>> filterField : TextField
- <<FXML>> filterLabel : Label
- root : Region
+ TemperatureViewController()
+ init(viewHandler : ViewHandler, viewModel : TemperatureViewModel, root : Region) : void
+ reset() : void
+ getRoot() : Region
- <<FXML>> updateButtonPressed() : void
- <<FXML>> onFilter() : void
ViewHandler
- primaryStage : Stage
- currentScene : Scene
+ ViewHandler(viewModelFactory : ViewModelFactory)
+ start(primaryStage : Stage) : void
+ openView(id : String) : void
- loadTemperatureView(fxmlFile : String) : Region
viewmodel
TemperatureViewModel
- temperature : StringProperty
- thermometerId : StringProperty
- filterValue : StringProperty
+ TemperatureViewModel(model : TemperatureModel)
+ getValue() : void
- setValue(t : Temperature) : void
+ updateThermometerId() : void
+ getTemperatureProperty() : StringProperty
+ getThermometerIdProperty() : StringProperty
+ getFilterValueProperty() : StringProperty
ViewModelFactory
+ ViewModelFactory(model : TemperatureModel)
+ getTemperatureViewModel() : TemperatureViewModel
mediator
<<interface>>
TemperatureModel
+ addTemperature(id : String, temperature : double) : void
+ getLastInsertedTemperature() : Temperature
+ getLastInsertedTemperature(id : String) : Temperature
TemperatureModelManager
+ TemperatureModelManager()
model
TemperatureList
*
Temperature
DateTime

Step 1: Split up class `TemperatureViewController` into two parts, a view and a viewModel (classes `TemperatureViewController` and `TemperatureViewModel` in the class diagram above)

Figure out how to implement class `TemperatureViewModel` with properties and functionality for the view (see class diagram). After the spilt-up, the class `TemperatureViewController` only present data and define the binding to the viewModel properties, and could end up like the following:

```
public class TemperatureViewController
{
  @FXML private Label outputLabel;
  @FXML private TextField filterField;
  @FXML private Label filterLabel;
  private ViewHandler viewHandler;
  private TemperatureViewModel viewModel;
  private Region root;

  public TemperatureViewController()
  {
  }

  public void init(ViewHandler viewHandler,
                   TemperatureViewModel viewModel, Region root)
  {
    this.viewHandler = viewHandler;
    this.viewModel = viewModel;
    this.root = root;

    // TODO: Statements to bind to viewModel properties
  }

  public void reset()
  {
    // empty
  }

  public Region getRoot()
  {
    return root;
  }

  @FXML private void updateButtonPressed()
  {
    viewModel.getValue();
  }

  @FXML private void onFilter()
  {
    viewModel.updateThermometerId();
  }
}
```

Step 2: Create class `ViewModelFactory` and update `ViewHandler` to have access to `ViewModelFactory` and not to the model.

Step 3: Update class `MyApplication` to reflect the changes – and run the `main` method.