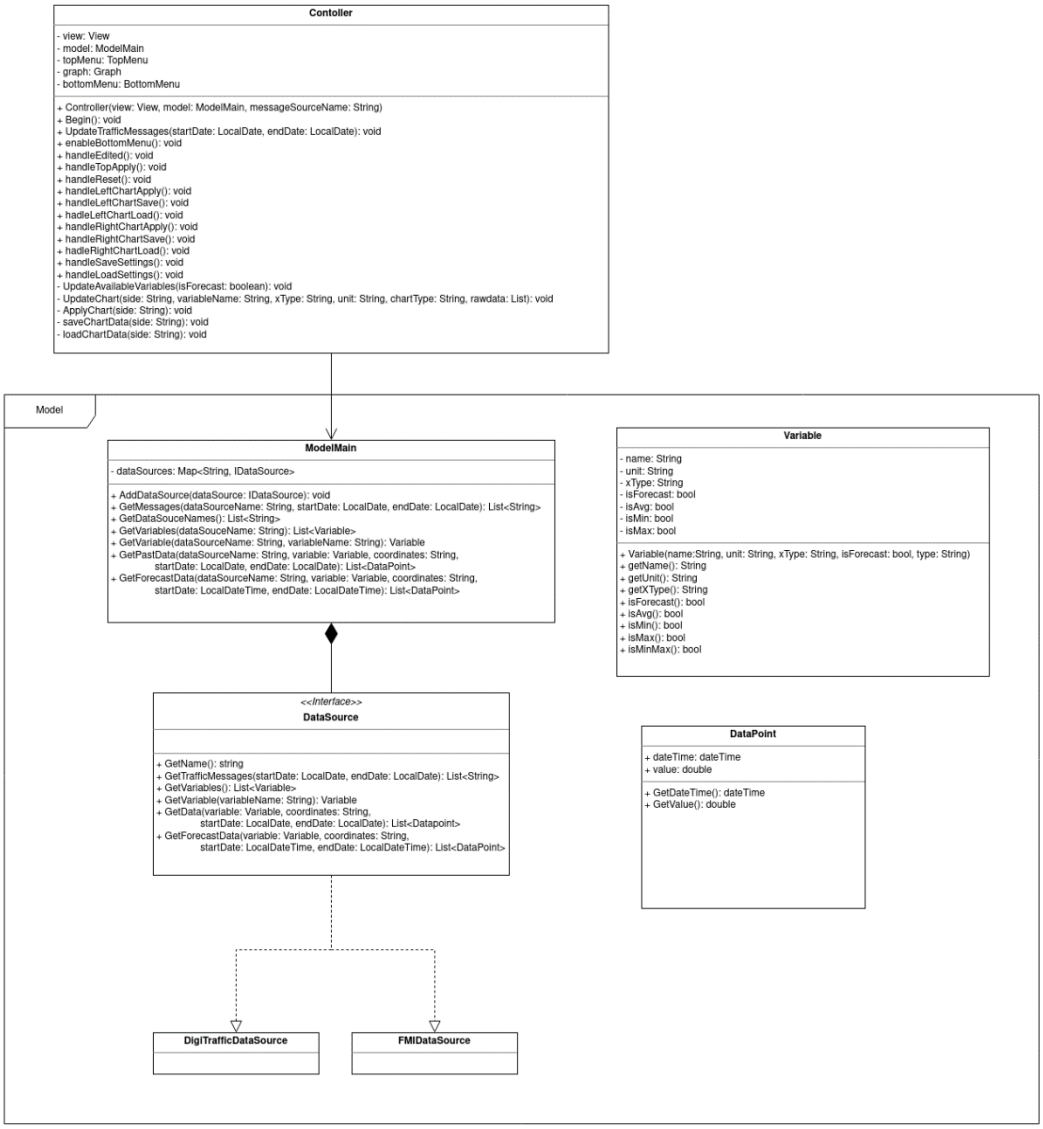
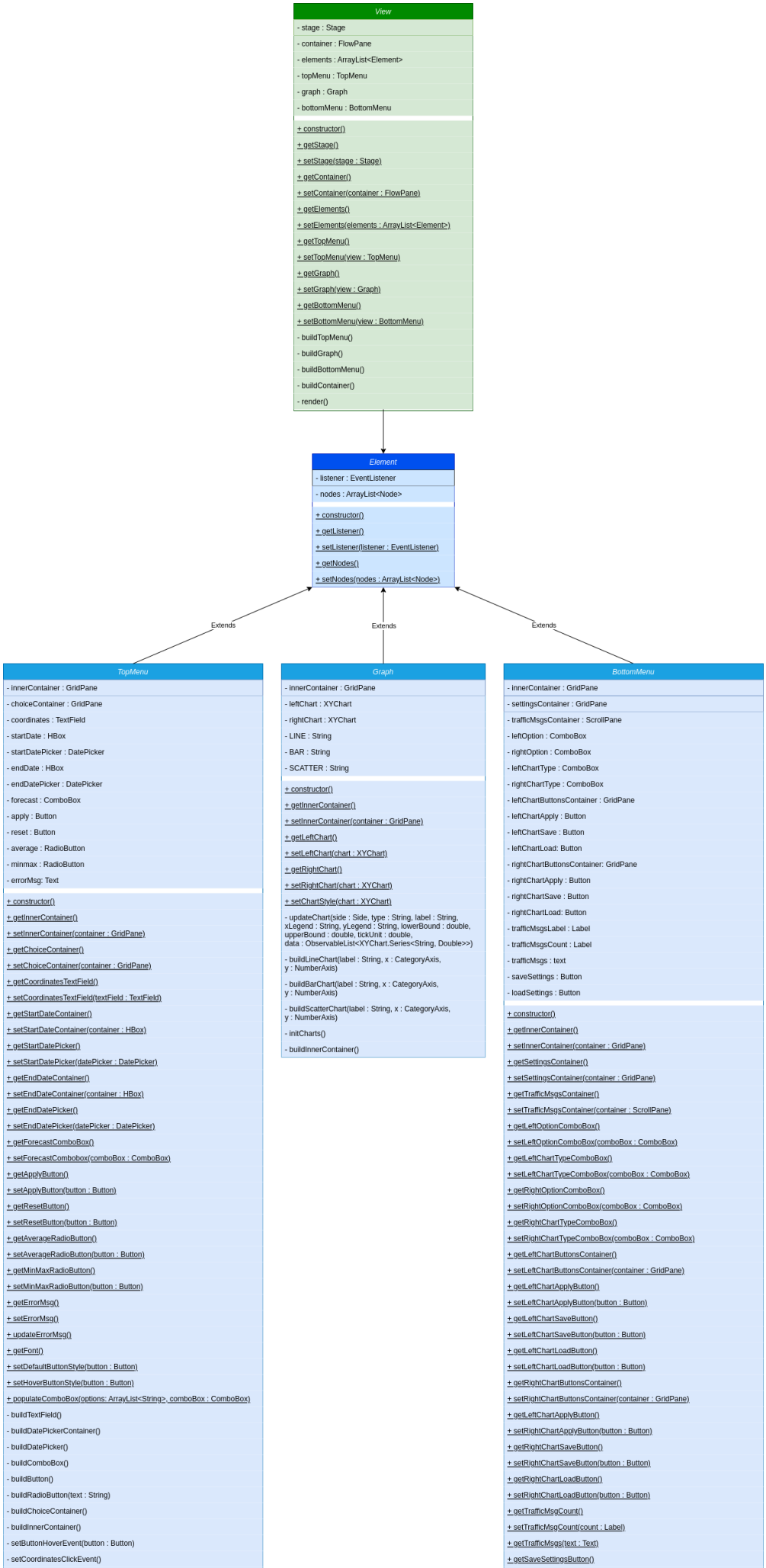
Design document

Java-group

Software Design, Fall 2022

UMLs


Contoller
- view: View
- model: ModelMain
- topMenu: TopMenu
- graph: Graph
- bottomMenu: BottomMenu
+ Controller(view: View, model: ModelMain, messageSourceName: String)
+ Begin(): void
+ UpdateTrafficMessages(startDate: LocalDate, endDate: LocalDate): void
+ enableBottomMenu(): void
+ handleEdited(): void
+ handleTopApply(): void
+ handleReset(): void
+ handleLeftChartApply(): void
+ handleLeftChartSave(): void
+ hadleLeftChartLoad(): void
+ handleRightChartApply(): void
+ handleRightChartSave(): void
+ hadleRightChartLoad(): void
+ handleSaveSettings(): void
+ handleLoadSettings(): void
- UpdateAvailableVariables(isForecast: boolean): void
- UpdateChart(side: String, variableName: String, xType: String, unit: String, chartType: String, rawdata: List): void
- ApplyChart(side: String): void
- saveChartData(side: String): void
- loadChartData(side: String): void
Model
ModelMain
- dataSources: Map<String, IDataSource>
+ AddDataSource(dataSource: IDataSource): void
+ GetMessages(dataSourceName: String, startDate: LocalDate, endDate: LocalDate): List<String>
+ GetDataSouceNames(): List<String>
+ GetVariables(dataSouceName: String): List<Variable>
+ GetVariable(dataSourceName: String, variableName: String): Variable
+ GetPastData(dataSourceName: String, variable: Variable, coordinates: String, startDate: LocalDate, endDate: LocalDate): List<DataPoint>
+ GetForecastData(dataSourceName: String, variable: Variable, coordinates: String, startDate: LocalDateTime, endDate: LocalDateTime): List<DataPoint>
Variable
- name: String
- unit: String
- xType: String
- isForecast: bool
- isAvg: bool
- isMin: bool
- isMax: bool
+ Variable(name:String, unit: String, xType: String, isForecast: bool, type: String)
+ getName(): String
+ getUnit(): String
+ getXType(): String
+ isForecast(): bool
+ isAvg(): bool
+ isMin(): bool
+ isMax(): bool
+ isMinMax(): bool
<<Interface>>
DataSource
+ GetName(): string
+ GetTrafficMessages(startDate: LocalDate, endDate: LocalDate): List<String>
+ GetVariables(): List<Variable>
+ GetVariable(variableName: String): Variable
+ GetData(variable: Variable, coordinates: String, startDate: LocalDate, endDate: LocalDate): List<Datapoint>
+ GetForecastData(variable: Variable, coordinates: String, startDate: LocalDateTime, endDate: LocalDateTime): List<DataPoint>
DataPoint
+ dateTime: dateTime
+ value: double
+ GetDateTime(): dateTime
+ GetValue(): double
DigiTrafficDataSource
FMIDataSource


View
- stage : Stage
- container : FlowPane
- elements : ArrayList<Element>
- topMenu : TopMenu
- graph : Graph
- bottomMenu : BottomMenu
+ constructor()
+ getStage()
+ setStage(stage : Stage)
+ getContainer()
+ setContainer(container : FlowPane)
+ getElements()
+ setElements(elements : ArrayList<Element>)
+ getTopMenu()
+ setTopMenu(view : TopMenu)
+ getGraph()
+ setGraph(view : Graph)
+ getBottomMenu()
+ setBottomMenu(view : BottomMenu)
- buildTopMenu()
- buildGraph()
- buildBottomMenu()
- buildContainer()
- render()
Element
- listener : EventListener
- nodes : ArrayList<Node>
+ constructor()
+ getListener()
+ setListener(listener : EventListener)
+ getNodes()
+ setNodes(nodes : ArrayList<Node>)
Extends
Extends
Extends
TopMenu
- innerContainer : GridPane
- choiceContainer : GridPane
- coordinates : TextField
- startDate : HBox
- startDatePicker : DatePicker
- endDate : HBox
- endDatePicker : DatePicker
- forecast : ComboBox
- apply : Button
- reset : Button
- average : RadioButton
- minmax : RadioButton
- errorMsg: Text
+ constructor()
+ getInnerContainer()
+ setInnerContainer(container : GridPane)
+ getChoiceContainer()
+ setChoiceContainer(container : GridPane)
+ getCoordinatesTextField()
+ setCoordinatesTextField(textField : TextField)
+ getStartDateContainer()
+ setStartDateContainer(container : HBox)
+ getStartDatePicker()
+ setStartDatePicker(datePicker : DatePicker)
+ getEndDateContainer()
+ setEndDateContainer(container : HBox)
+ getEndDatePicker()
+ setEndDatePicker(datePicker : DatePicker)
+ getForecastComboBox()
+ setForecastCombobox(comboBox : ComboBox)
+ getApplyButton()
+ setApplyButton(button : Button)
+ getResetButton()
+ setResetButton(button : Button)
+ getAverageRadioButton()
+ setAverageRadioButton(button : Button)
+ getMinMaxRadioButton()
+ setMinMaxRadioButton(button : Button)
+ getErrorMsg()
+ setErrorMsg()
+ updateErrorMsg()
+ getFont()
+ setDefaultButtonStyle(button : Button)
+ setHoverButtonStyle(button : Button)
+ populateComboBox(options: ArrayList<String>, comboBox : ComboBox)
- buildTextField()
- buildDatePickerContainer()
- buildDatePicker()
- buildComboBox()
- buildButton()
- buildRadioButton(text : String)
- buildChoiceContainer()
- buildInnerContainer()
- setButtonHoverEvent(button : Button)
- setCoordinatesClickEvent()
Graph
- innerContainer : GridPane
- leftChart : XYChart
- rightChart : XYChart
- LINE : String
- BAR : String
- SCATTER : String
+ constructor()
+ getInnerContainer()
+ setInnerContainer(container : GridPane)
+ getLeftChart()
+ setLeftChart(chart : XYChart)
+ getRightChart()
+ setRightChart(chart : XYChart)
+ setChartStyle(chart : XYChart)
- updateChart(side : Side, type : String, label : String, xLegend : String, yLegend : String, lowerBound : double, upperBound : double, tickUnit : double, data : ObservableList<XYChart.Series<String, Double>>)
- buildLineChart(label : String, x : CategoryAxis, y : NumberAxis)
- buildBarChart(label : String, x : CategoryAxis, y : NumberAxis)
- buildScatterChart(label : String, x : CategoryAxis, y : NumberAxis)
- initCharts()
- buildInnerContainer()
BottomMenu
- innerContainer : GridPane
- settingsContainer : GridPane
- trafficMsgsContainer : ScrollPane
- leftOption : ComboBox
- rightOption : ComboBox
- leftChartType : ComboBox
- rightChartType : ComboBox
- leftChartButtonsContainer : GridPane
- leftChartApply : Button
- leftChartSave : Button
- leftChartLoad: Button
- rightChartButtonsContainer: GridPane
- rightChartApply : Button
- rightChartSave : Button
- rightChartLoad: Button
- trafficMsgsLabel : Label
- trafficMsgsCount : Label
- trafficMsgs : text
- saveSettings : Button
- loadSettings : Button
+ constructor()
+ getInnerContainer()
+ setInnerContainer(container : GridPane)
+ getSettingsContainer()
+ setSettingsContainer(container : GridPane)
+ getTrafficMsgsContainer()
+ setTrafficMsgsContainer(container : ScrollPane)
+ getLeftOptionComboBox()
+ setLeftOptionComboBox(comboBox : ComboBox)
+ getLeftChartTypeComboBox()
+ setLeftChartTypeComboBox(comboBox : ComboBox)
+ getRightOptionComboBox()
+ setRightOptionComboBox(comboBox : ComboBox)
+ getRightChartTypeComboBox()
+ setRightChartTypeComboBox(comboBox : ComboBox)
+ getLeftChartButtonsContainer()
+ setLeftChartButtonsContainer(container : GridPane)
+ getLeftChartApplyButton()
+ setLeftChartApplyButton(button : Button)
+ getLeftChartSaveButton()
+ setLeftChartSaveButton(button : Button)
+ getLeftChartLoadButton()
+ setLeftChartLoadButton(button : Button)
+ getRightChartButtonsContainer()
+ setRightChartButtonsContainer(container : GridPane)
+ getLeftChartApplyButton()
+ setRightChartApplyButton(button : Button)
+ getRightChartSaveButton()
+ setRightChartSaveButton(button : Button)
+ getRightChartLoadButton()
+ setRightChartLoadButton(button : Button)
+ getTrafficMsgCount()
+ setTrafficMsgCount(count : Label)
+ getTrafficMsgs(text : Text)
+ getSaveSettingsButton()

Description

The general structure of the program will follow MVC pattern as having separate parts for data handling and visuals makes sense based on the program requirements. Following the MVC pattern also facilitates sensible ways to divide work between the group members without too much interdependency during production.

The model will contain ModelMain class that contains available data source objects and provides interface for requesting data from the model. The data sources are done by data source specific implementations of a DataSource interface class. The DataSource interface provides methods for requesting specific data from a data source. The model also contains Variable and DataPoint classes for storing information.

The controller part of the program will be made of a single Controller class which facilitates communication between the Model and the View. The controller acts as listener to the View and whenever methods that require new data are called in the View, the View notifies the Controller, and the Controller reacts by calling the necessary methods in the View for additional data. If data from the Model is required, Controller passes it to the View.

The classes used in the Model and the View can be seen in the UMLs.

Decisions

The program's UI will be made using JavaFX as it is the only UI tool for Java that several group members are familiar with.

The Controller implements EventListener interface that contains all the methods for handling View related actions. When a user performs an action in the UI, the View notifies the Controller that then calls functions from the Model to fetch the required data.

However, actions that only require a minor change in the UI without any additional information, are done within the View. These sorts of actions are, for example, hover effects on buttons.

Other 3rd party libraries used

org.json [https://mvnrepository.com/artifact/org.json/json] – used for handling JSON data from DigiTraffic

General descriptions of components

Class	Description
ModelMain.java	Class for modeling the data
IDataSource.java	Interface class for implementing data sources
DataPoint.java	Class for a single datapoint used in sources
Variable.java	Class for storing variables used in data sources
DigiTrafficSource / FMIDataSource.java	Classes for fetching data from the APIs
Controller.java	Class handles user inputs, calls data sources and view accordingly
View / Menu / Graph	Classes set and render requested data

Self-evaluation

Mid-term evaluation

We have been able to design and implement the original prototype quite well. The only major difference was switching from one graph to two graphs due to some difficulties in combining two graphs into one. We also decided to provide traffic messages as text instead of just numbers (amount). Other than that, the current UI is quite similar to the original prototype.

The workload for each group member has been appropriate. All of us have been able to concentrate on tasks that have been to our liking.

Final evaluation

We were able to stick to our initial design quite well. Some minor changes were made to the Model class to better suit the data we were handling in the project. The Variable class also required considerable extension to facilitate some of the more detailed parts of the more advanced functionalities of the program. Due to time constraints and having only three group members, some of the implementations were a bit rough around the edges.

Project assignment instructions left a lot of room for interpretation, which made understanding specific requirements and how they should be implemented in practice sometimes difficult.