# Exercise X8 – Model-View-Controller

## Find (and insert into a document) a general diagram of a Model-View-Controller design pattern

## Describe the overall purpose for the Model-View-Controller design pattern.

The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other.

## Describe in text each part of the Model-View-Controller. Hereby purpose and responsibility for each part of the pattern.

**Controller**

Controllers act as an interface between Model and View components to process all the logic and requests, manipulate data using the Model component and interact with the Views to show the final output.

**View**

Views are responsible for displaying all or a portion of the data to the user.

**Model**

Model corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other logic-related data.

## Give overall a few remarks to implementation, what to remember.

## Insert code examples (your own code) for each part of the Model-View-Controller pattern. Not full code only code fractions directly related to the pattern.

**Controller** (BusController)

private void loadList() {  
 busListview.getSelectionModel().setSelectionMode(SelectionMode.*MULTIPLE*);  
 ObservableList<Bus> items = DataHandler.*getInstance*().getObservableListOfBuses();  
 busListview.setItems(items);  
}  
  
*/\*\*  
 \* Deletes selected buses from list of buses.  
 \*/*public void deleteBus(ActionEvent actionEvent) throws FileNotFoundException, ParseException {  
 ObservableList<Bus> selected;  
 selected = busListview.getSelectionModel().getSelectedItems();  
 for (Bus aSelected : selected) {  
 DataHandler.*getInstance*().removeFromBuslist(aSelected);  
 }  
  
 loadList();  
 DataHandler.*getInstance*().save();  
}

}  
 }

BusController delegates work to DataHandler from mediator package.

**DataHandler**

public void removeFromBuslist(Bus bus) {  
 busList.removeBus(bus);  
}

public ObservableList getObservableListOfBuses() {  
 ObservableList<Bus> items = FXCollections.*observableArrayList*();  
 for (Bus bus : busList.getArrayBuses()) {  
 items.add(bus);  
 }  
 return items;  
}

DataHandler solves the tasks by accessing the model.

## Create and insert into the document a UML class diagram of the Model-View-Controller design pattern you implemented

