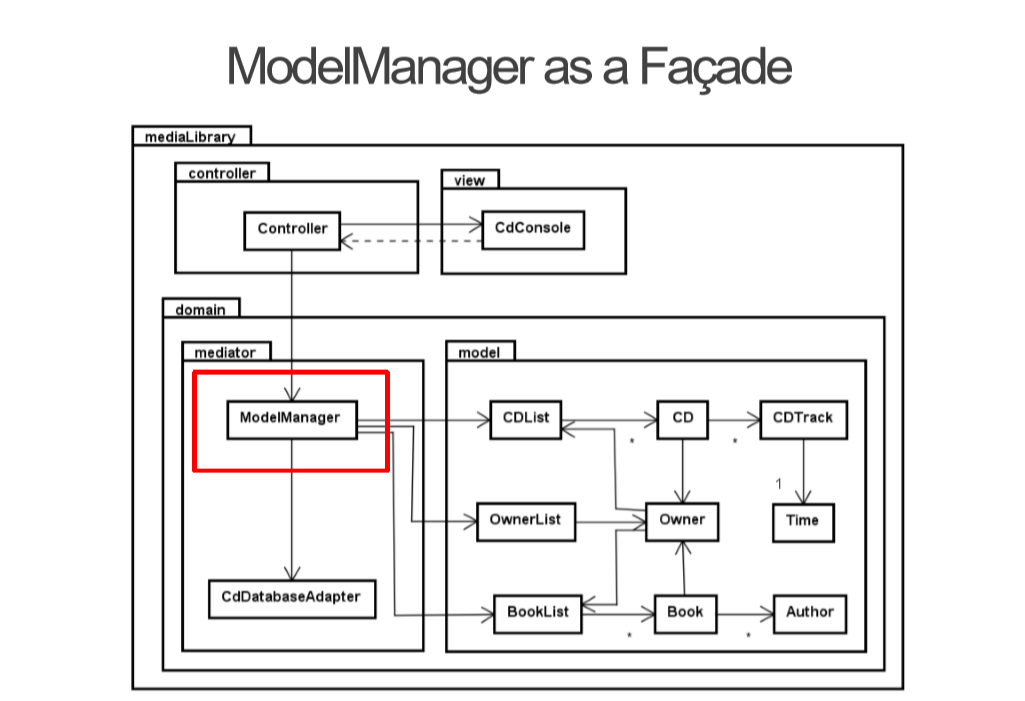
**Façade**

**An example of UML class diagram for façade design pattern:**

****

**The overall purpose for the Façade design pattern:**

* Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

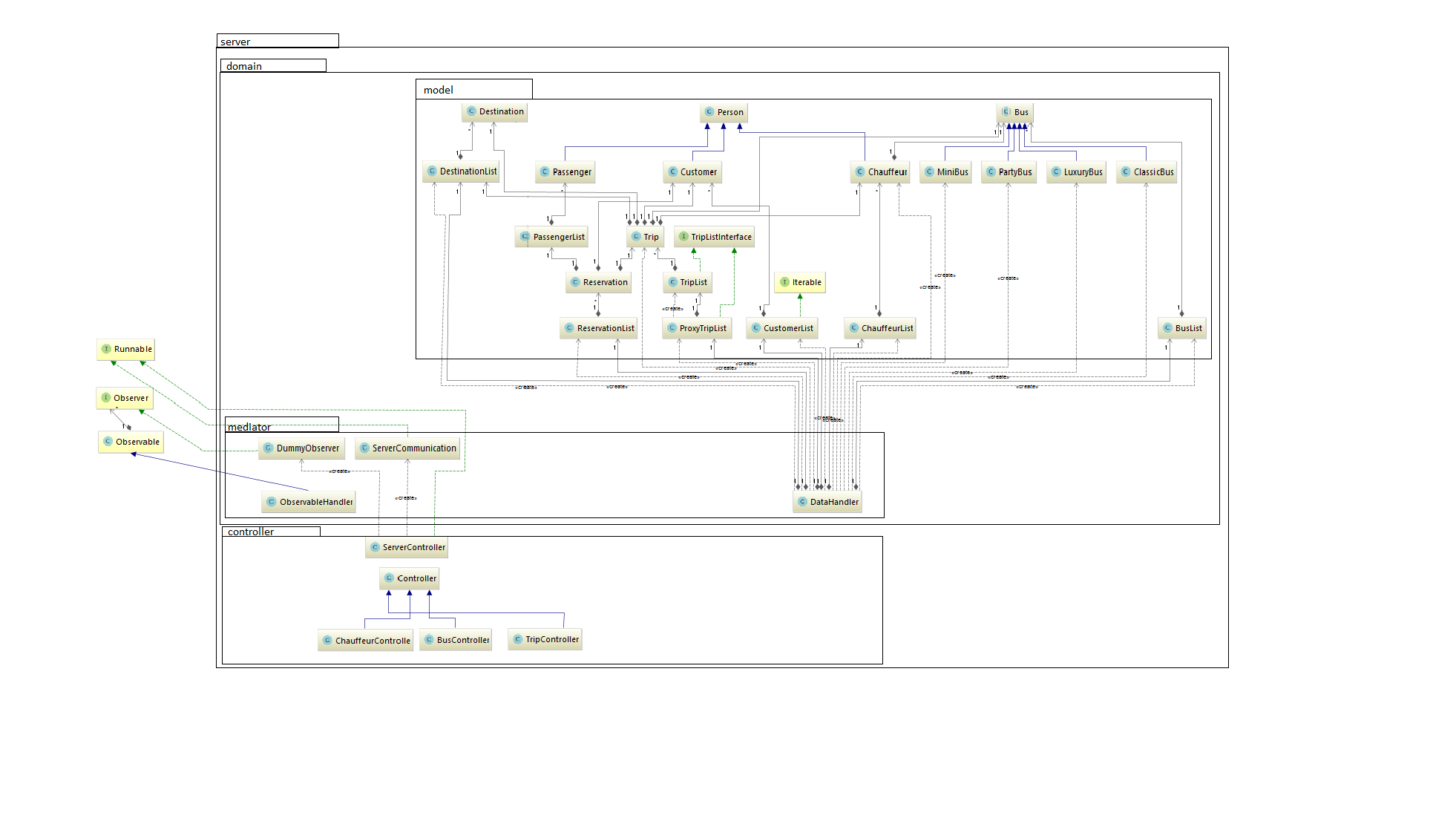
**Description of example UML diagram:**

* The ModelManager is the façade for model having model state (and additional actions).
* The intent of façade is to provide an interface with simplified methods required by client which delegate calls to methods of existing system classes.
* Facade shows how to make a single object represent an entire subsystem.

**Usage of façade design pattern:**

* When client wants to access data from the model, the model manager method is called and model manager does the job of accessing model data.
* Façade design pattern does not require to have interface for façade class.

**Our implementation:**



* In our system, the data is controlled through one class called DataHandler. The user is able to access all the data through this class. It doesn’t use an interface like it is shown on the original diagram. When the client wants to access the list of tours from his computer, The DataHandler class allows the user to get their list of information in one call by creating an object of type DataHandler.

**Code example:**

* Method load from data handler loads data from file and stores them inside

public void load() {  
 String filename = "mainData.bin";  
 ObjectInputStream in = null;  
 try {  
 File file = new File(filename);  
 FileInputStream fis = new FileInputStream(file);  
 in = new ObjectInputStream(fis);  
 trips = (ProxyTripList) in.readObject();  
 busList = (BusList) in.readObject();  
 chauffeurList = (ChauffeurList) in.readObject();  
 customerList = (CustomerList) in.readObject();  
 reservationList = (ReservationList) in.readObject();  
 destinationList = (DestinationList) in.readObject();  
 } catch (ClassCastException | IOException | ClassNotFoundException e) {  
 testCreate();  
 } finally {  
 try {  
 if (in != null) {  
 in.close();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

* User can call getter methods to get model data

public ProxyTripList getTrips() {  
 return trips;  
}

* Or controllers call methods to get ObservableLists<> in order to display data in GUI

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