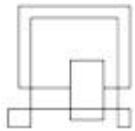
VIA University College



Semester Project 1 (SEP1) Single User System

Steffen Vissing Andersen (sva@via.dk)

Analysis: Interview →Requirements

Requirements

1. An employee should be able to reserve a vehicle for a customer.
2. An employee making a reservation should always enter customer name, phone number,, pick up time, return time, and the vehicle to rent.
3. For a family car, a user should be able to get a list of vehicle information with registration number, model, make, manufacturer year, year of registration, amount of kilometers and notes e.g. about any defects.
4. For a van and a truck, an employee should be able to get the same vehicle information as for family cars and additional the length, the load size and the driver's license category needed.
5. An employee should be able to get vehicle information part of making a reservation.
6. An employee should be able to delete a reservation

...

24. An employee should be able to add data about new vehicles, modify data for an existing vehicle and delete a record for a vehicle no longer available for rent

Analysis: Use Case Modelling

Requirements

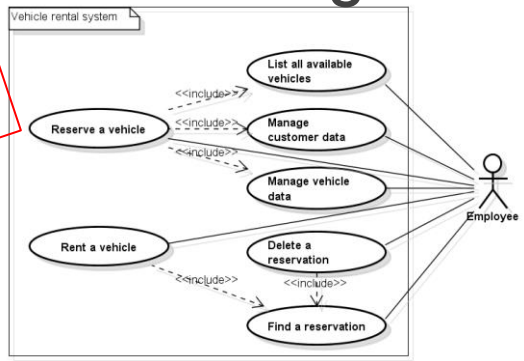
1. An employee should be able to reserve a vehicle for a customer.
2. An employee making a reservation should always enter customer name, phone number,, pick up time, return time, and the vehicle to rent.
3. For a family car, a user should be able to get a list of vehicle information with registration number, model, make, manufacturer year, year of registration, amount of kilometers and notes e.g. about any defects.
4. For a van and a truck, an employee should be able to get the same vehicle information as for family cars and additional the length, the load size and the driver's license category needed.
5. An employee should be able to get vehicle information part of making a reservation.
6. An employee should be able to delete a reservation

…

24. An employee should be able to add data about new vehicles, modify data for an existing vehicle and delete a record for a vehicle no longer available for rent

What?

Use Case diagram



How?

Use Case description #1

Use Case: Reserve a vehicle

….

1. Enter vehicle type (family car, van, truck or bus) and the two dates; pick-up date and return date
2. System returns a list of available vehicles of the given type in the given date interval (Use Case: List all available vehicles)
3. Select from the list the vehicle to reserve
4. System returns details about the vehicle (Use Case: Manage vehicle data)
5. If vehicle cannot be accepted by the customer then go to step 4 again
6. Verify the dates and vehicle to reserve
7. If dates are not correct then go to step 1
8. Enter name and phone number for the customer
9. System search for the customer by phone number and name (Use case: Manage customer data)

…

Analysis: UCD →Activity Diagrams

Use Case description #1

Use Case: Reserve a vehicle

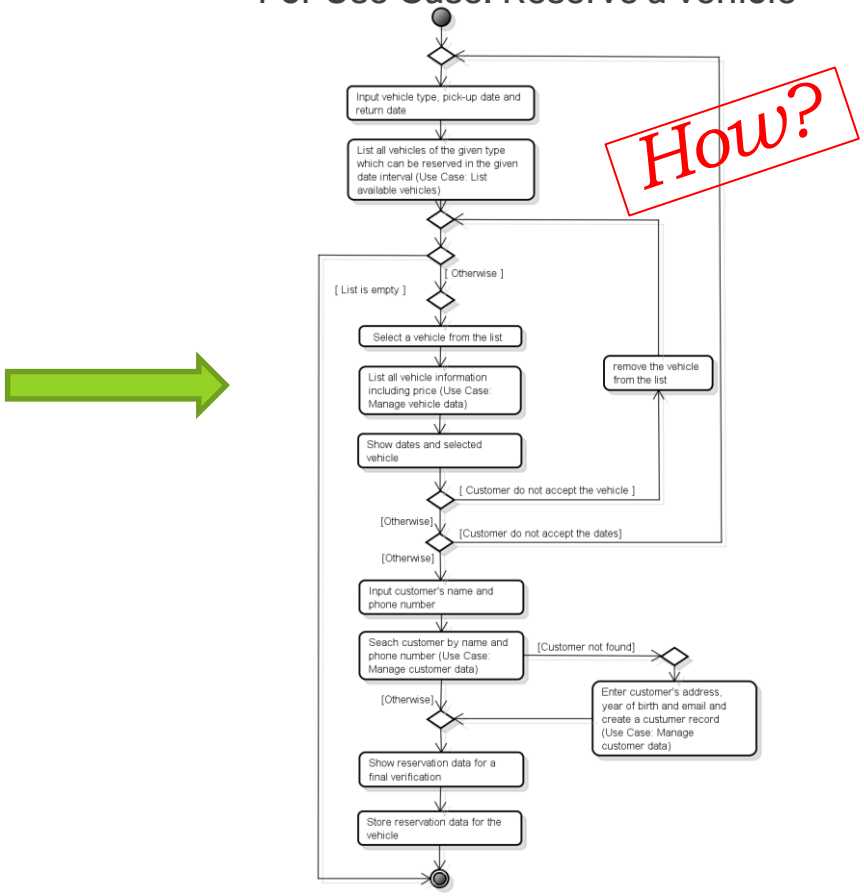
….

1. Enter vehicle type (family car, van, truck or bus) and the two dates; pick-up date and return date
2. System returns a list of available vehicles of the given type in the given date interval (Use Case: List all available vehicles)
3. Select from the list the vehicle to reserve
4. System returns details about the vehicle (Use Case: Manage vehicle data)
5. If vehicle cannot be accepted by the customer then go to step 4 again
6. Verify the dates and vehicle to reserve
7. If dates are not correct then go to step 1
8. Enter name and phone number for the customer
9. System search for the customer by phone number and name (Use case: Manage customer data)

…

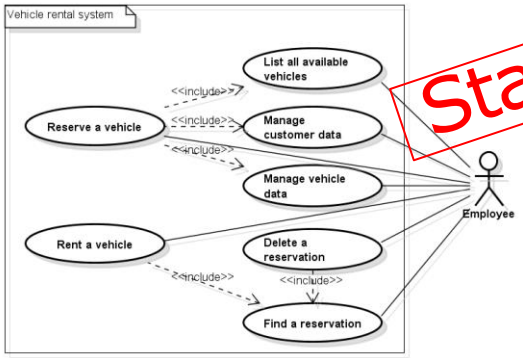
Activity diagram #1

For Use Case: Reserve a vehicle

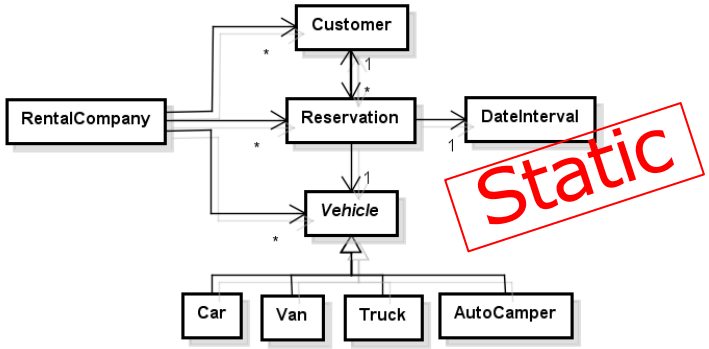


Diagrams

Use Case diagram

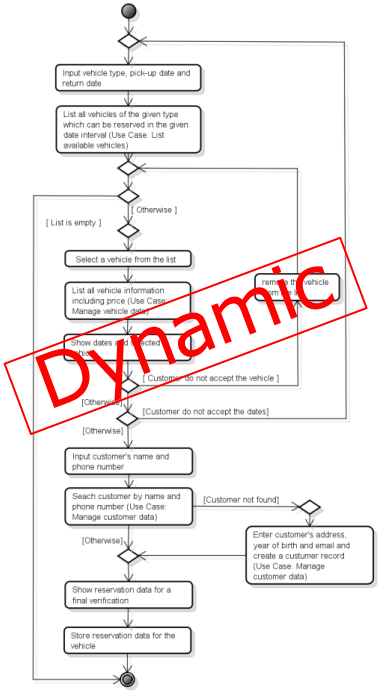


Class diagram

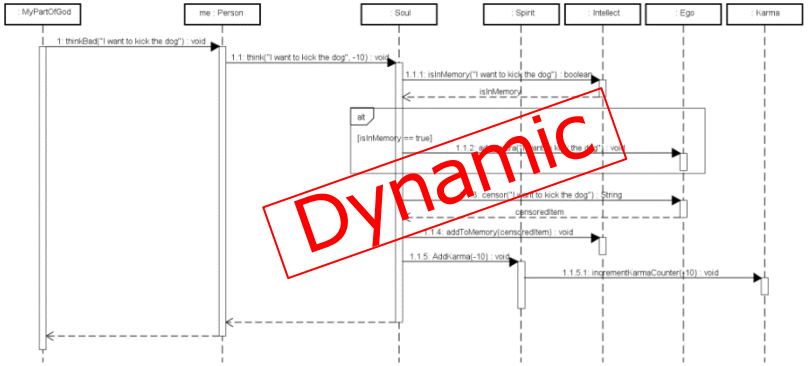


Activity diagram #1

For Use Case: Withdraw Cash

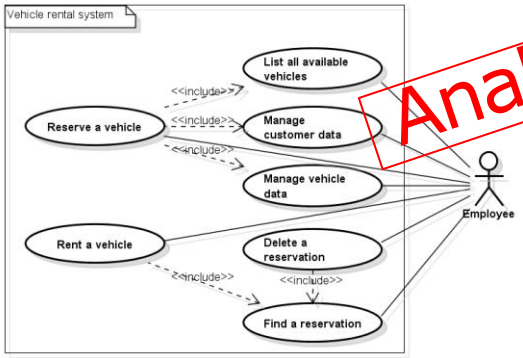


Sequence diagrams



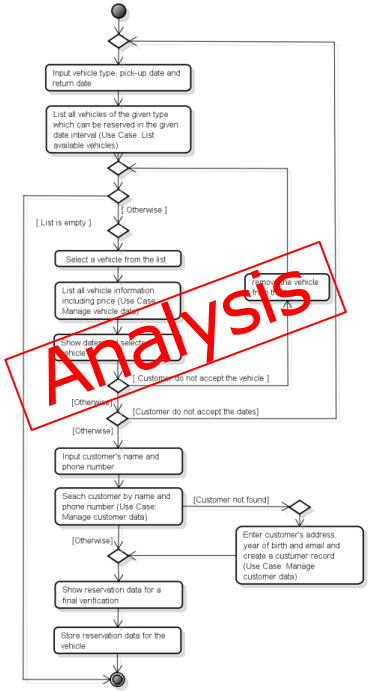
Diagrams

Use Case diagram

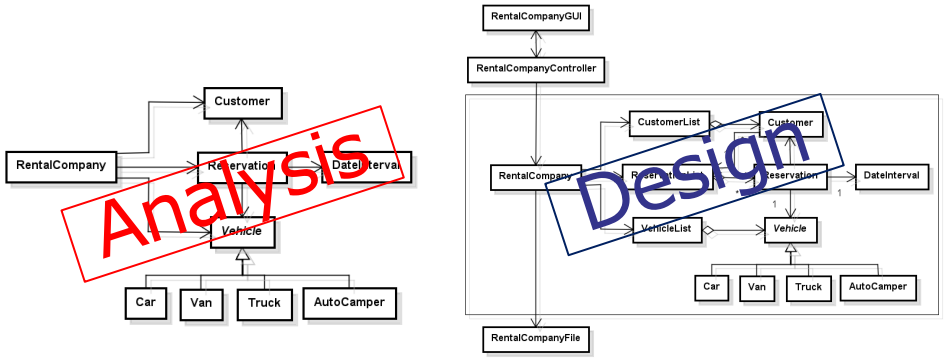


Activity diagram #1

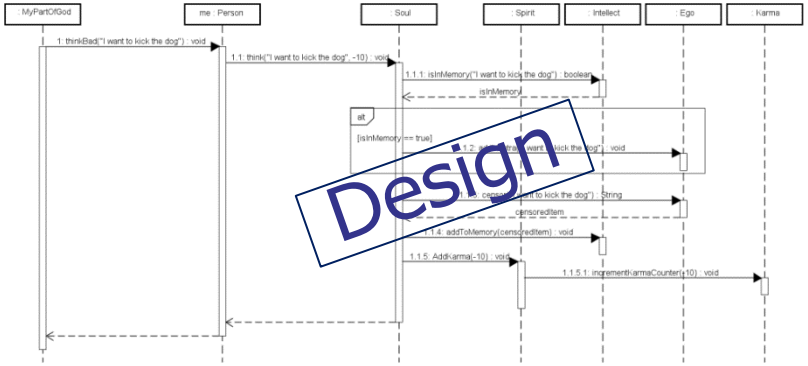
For Use Case: Withdraw Cash



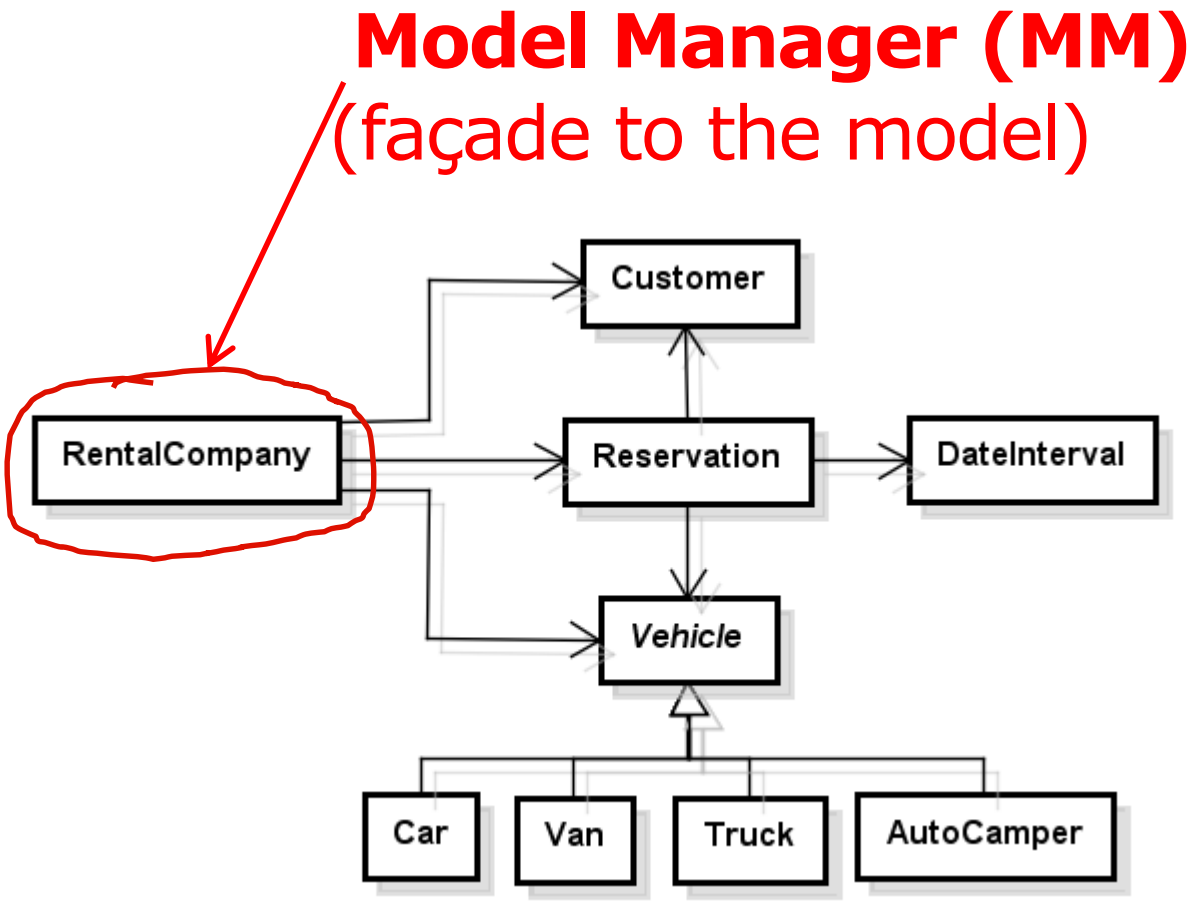
Class diagram



Sequence diagrams

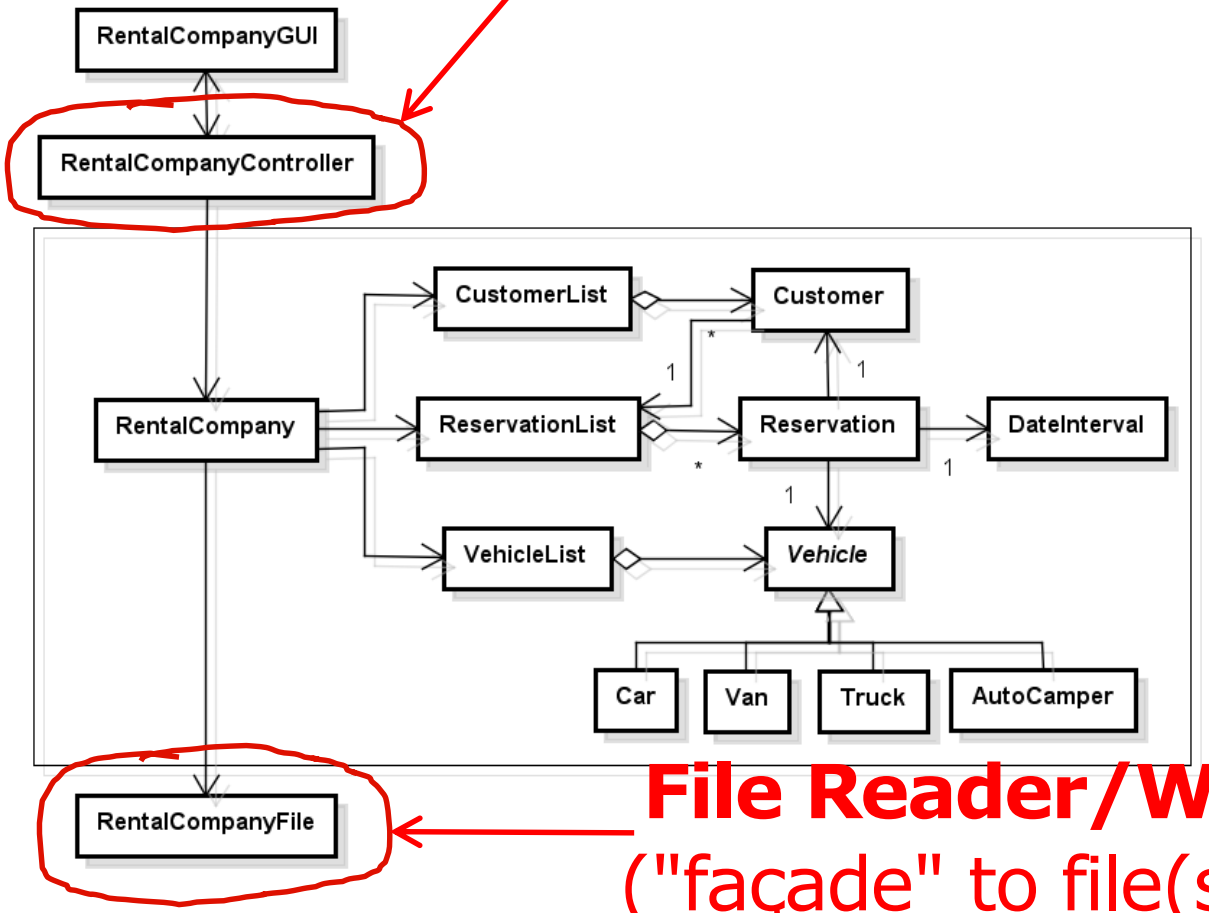


Model Classes (Analysis → Design)



MVC

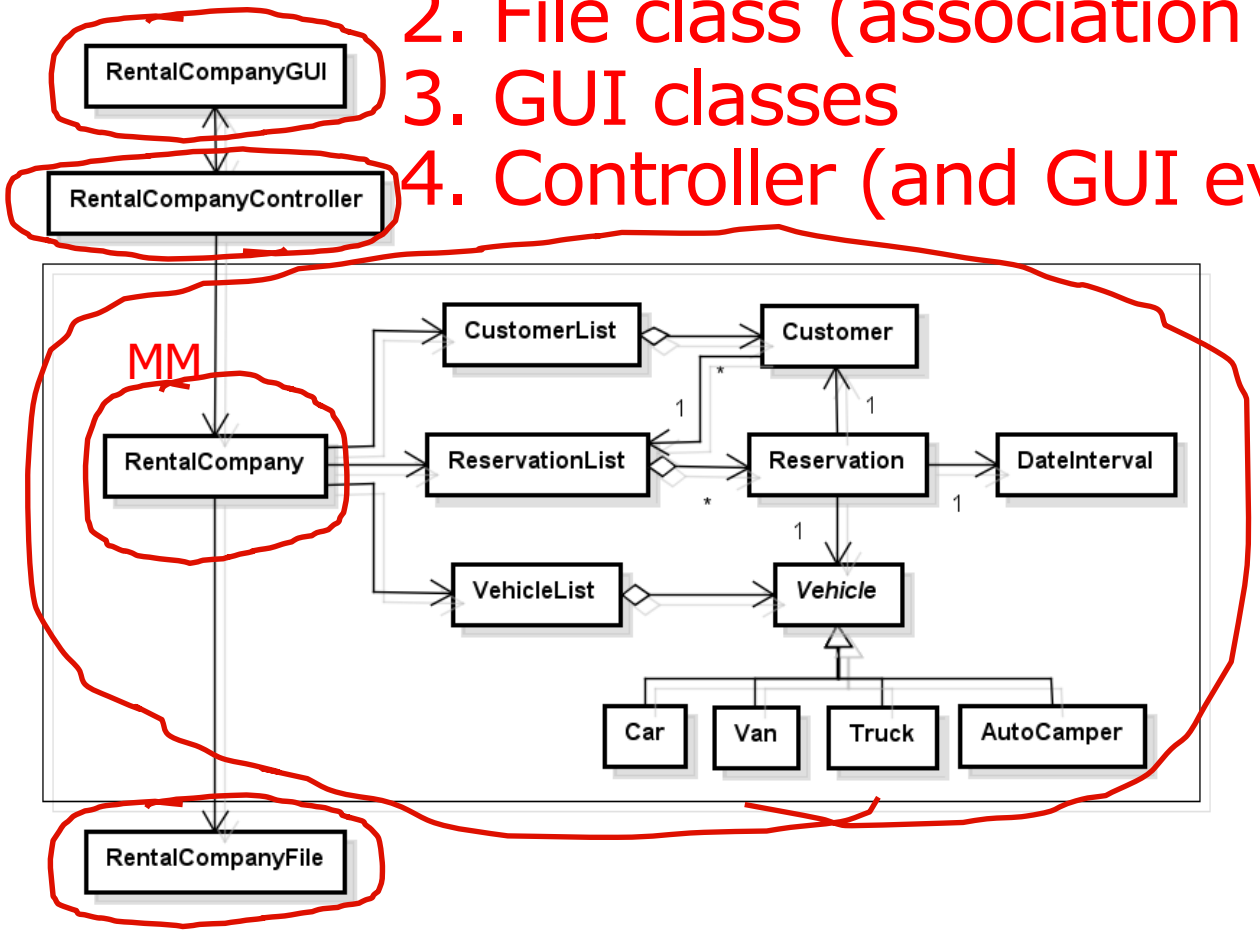
Controller
("façade" to the GUI)



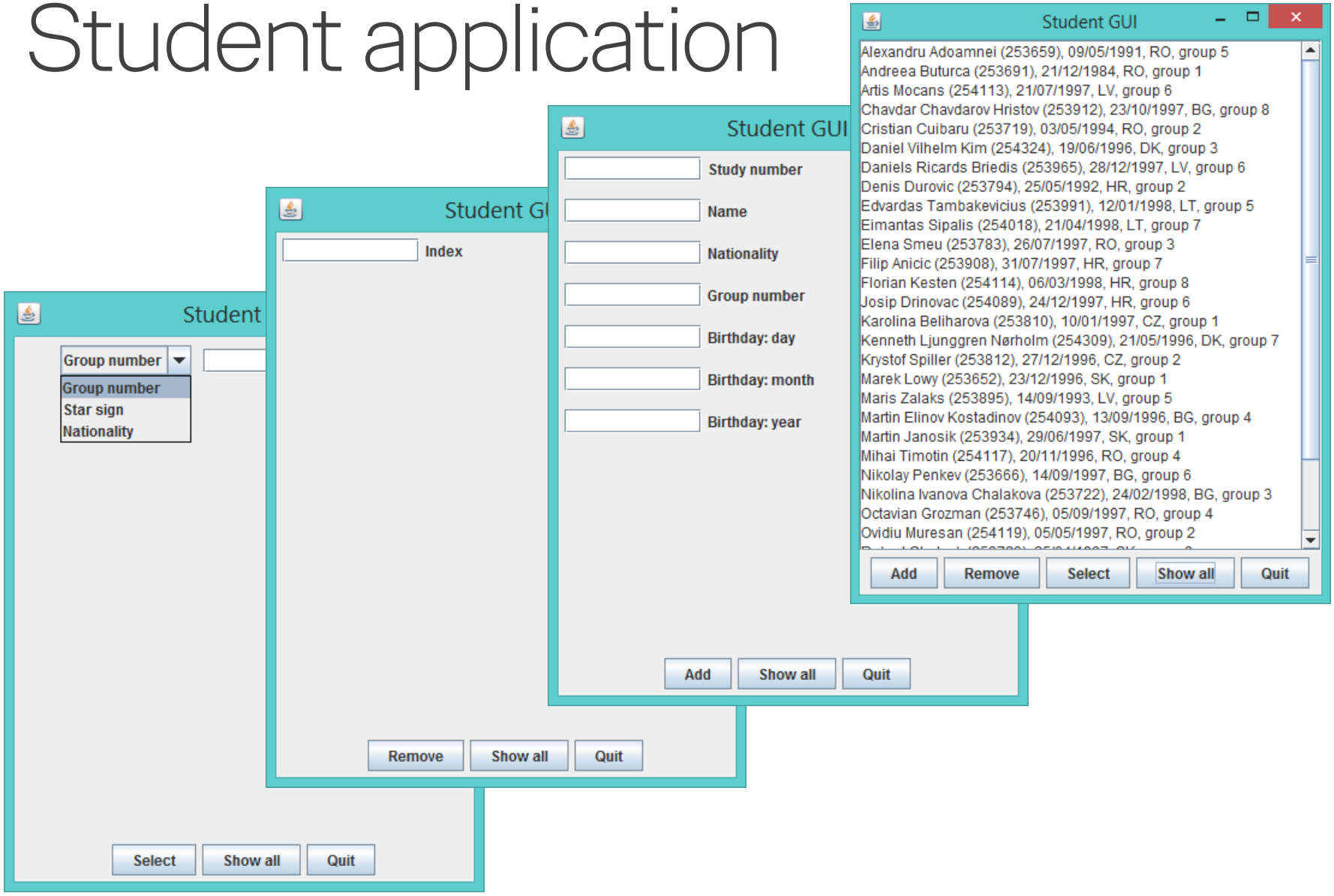
File Reader/Writer
("façade" to file(s))

Design class diagram

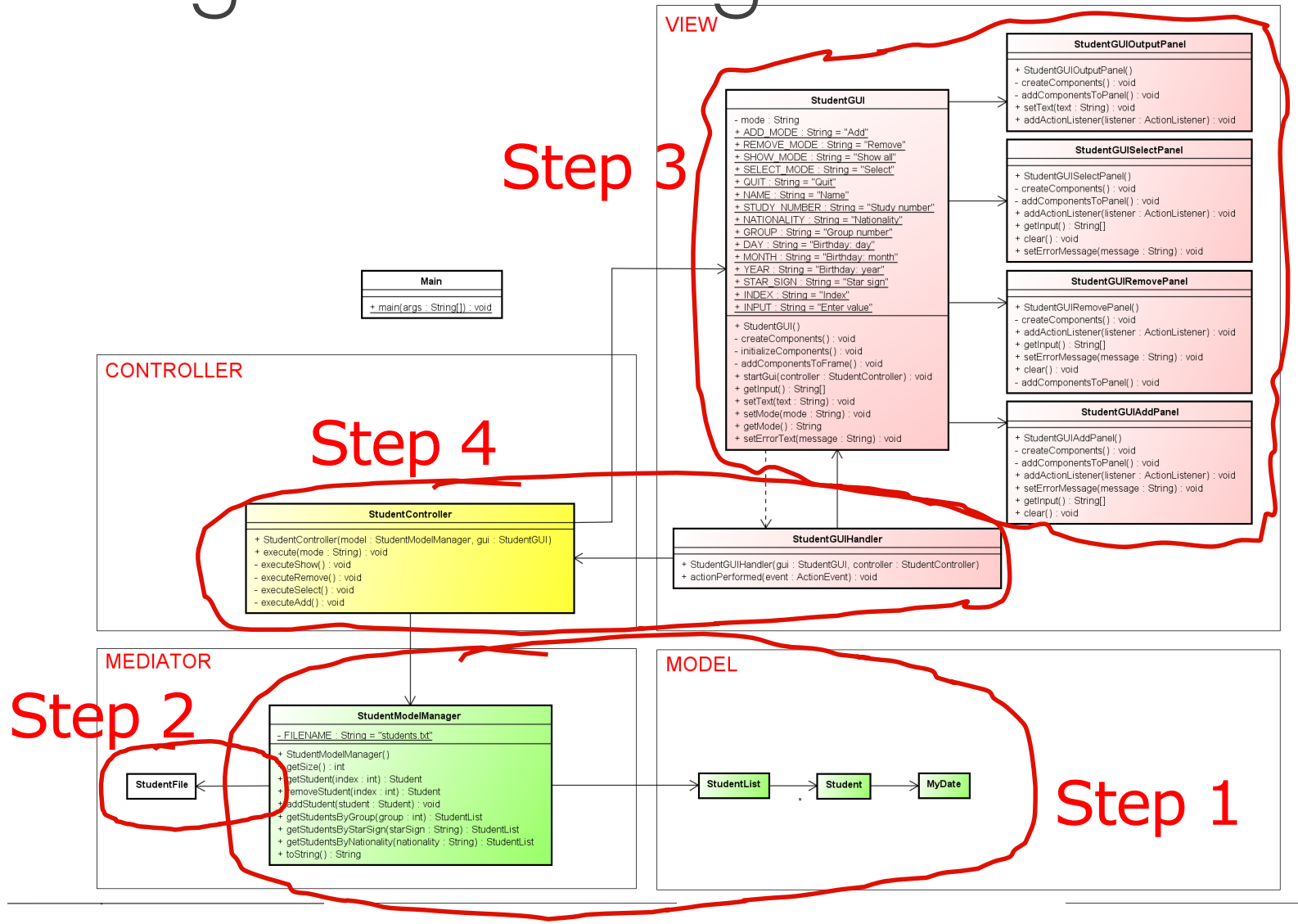
1. Model classes (incl. MM)
2. File class (association in MM)
3. GUI classes
4. Controller (and GUI events)



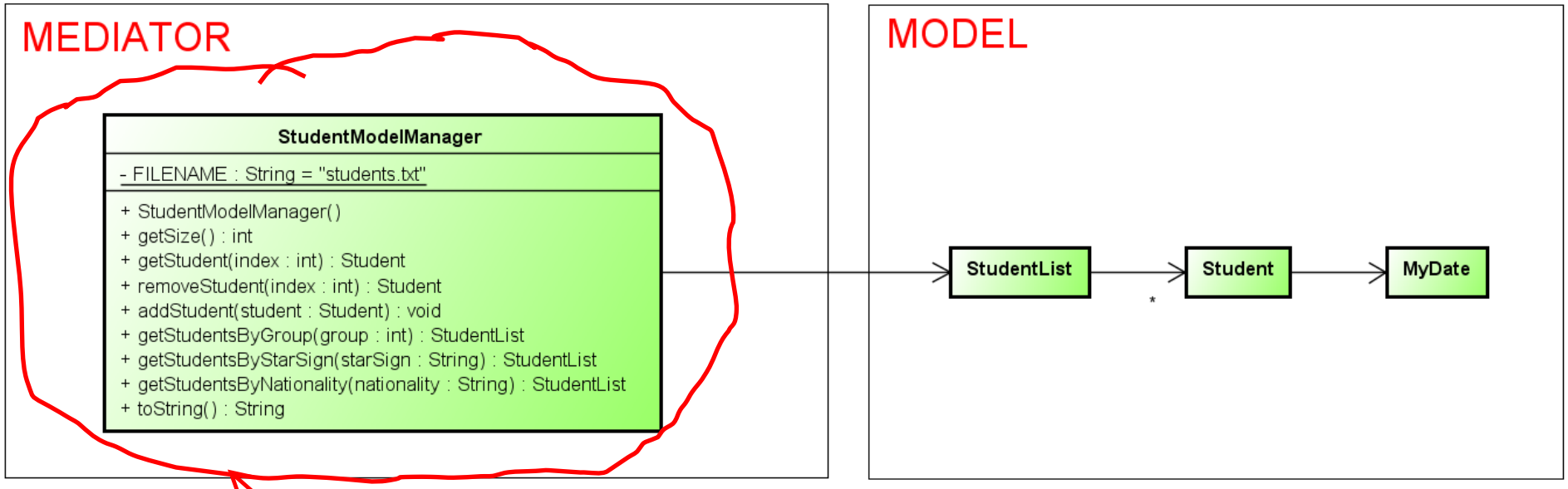
Student application



Design: Class diagram



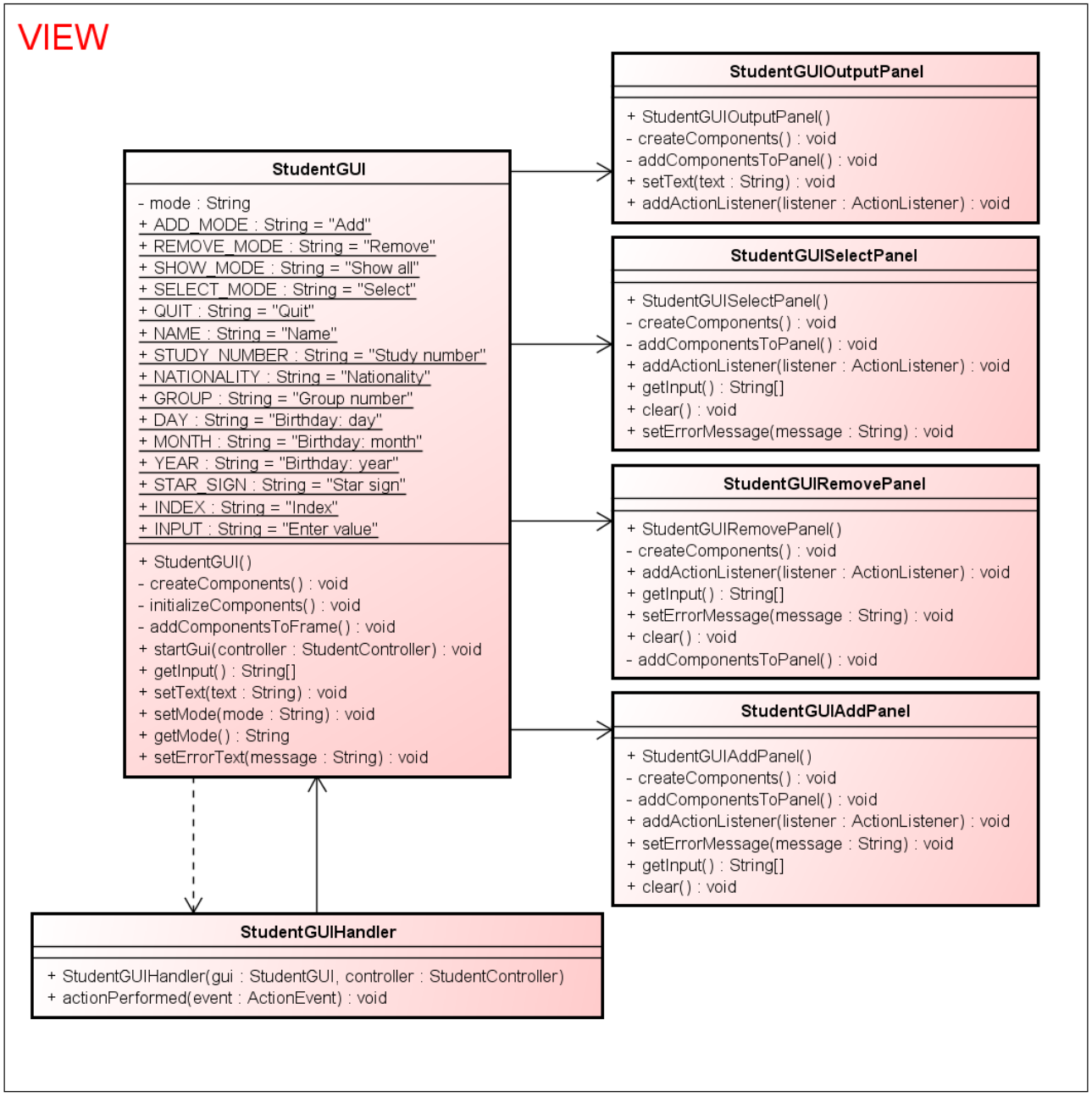
Model



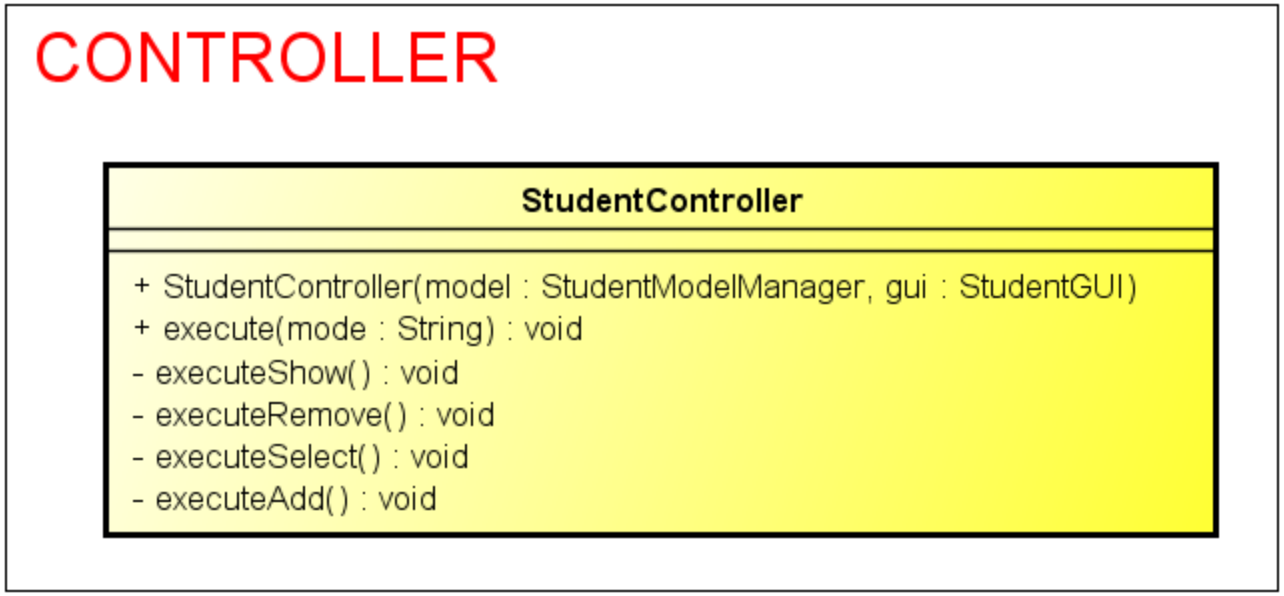
- Model manager keeps the state of the model
- Constructor gets all students from a file
- All other methods gets from and sets to the model (StudentList)

(Critical updates: if model has been changed then also update file)

View


VIEW
StudentGUI
- mode : String
+ ADD_MODE : String = "Add"
+ REMOVE_MODE : String = "Remove"
+ SHOW_MODE : String = "Show all"
+ SELECT_MODE : String = "Select"
+ QUIT : String = "Quit"
+ NAME : String = "Name"
+ STUDY_NUMBER : String = "Study number"
+ NATIONALITY : String = "Nationality"
+ GROUP : String = "Group number"
+ DAY : String = "Birthday: day"
+ MONTH : String = "Birthday: month"
+ YEAR : String = "Birthday: year"
+ STAR_SIGN : String = "Star sign"
+ INDEX : String = "Index"
+ INPUT : String = "Enter value"
+ StudentGUI()
- createComponents() : void
- initializeComponents() : void
- addComponentsToFrame() : void
+ startGui(controller : StudentController) : void
+ getInput() : String[]
+ setText(text : String) : void
+ setMode(mode : String) : void
+ getMode() : String
+ setErrorText(message : String) : void
StudentGUIOutputPanel
+ StudentGUIOutputPanel()
- createComponents() : void
- addComponentsToPanel() : void
+ setText(text : String) : void
+ addActionListener(listener : ActionListener) : void
StudentGUISelectPanel
+ StudentGUISelectPanel()
- createComponents() : void
- addComponentsToPanel() : void
+ addActionListener(listener : ActionListener) : void
+ getInput() : String[]
+ clear() : void
+ setErrorMessage(message : String) : void
StudentGUIRemovePanel
+ StudentGUIRemovePanel()
- createComponents() : void
+ addActionListener(listener : ActionListener) : void
+ getInput() : String[]
+ setErrorMessage(message : String) : void
+ clear() : void
- addComponentsToPanel() : void
StudentGUIAddPanel
+ StudentGUIAddPanel()
- createComponents() : void
- addComponentsToPanel() : void
+ addActionListener(listener : ActionListener) : void
+ setErrorMessage(message : String) : void
+ getInput() : String[]
+ clear() : void
StudentGUIHandler
+ StudentGUIHandler(gui : StudentGUI, controller : StudentController)
+ actionPerformed(event : ActionEvent) : void

Controller



- Taking actions from the View (here method execute)
- Call methods in the View (get additional data and show result)
- Call methods in the Model (get and set data)

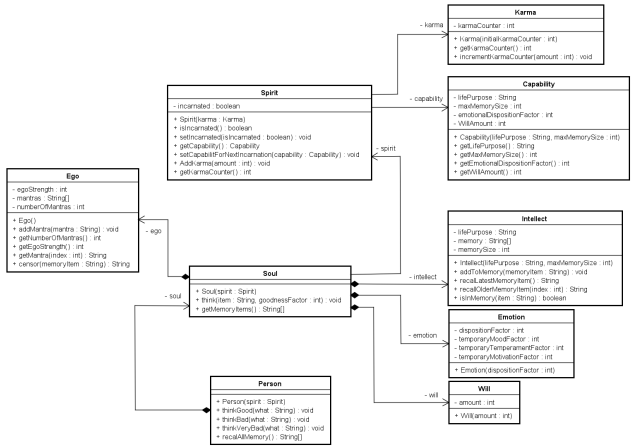
Main

```
public class Main
{
   public static void main(String[] args)
   {
      StudentModelManager model = new StudentModelManager();
      StudentGUI gui = new StudentGUI();
      StudentController controller =
                new StudentController(model, gui);
      gui.startGui(controller);
   }

}
```

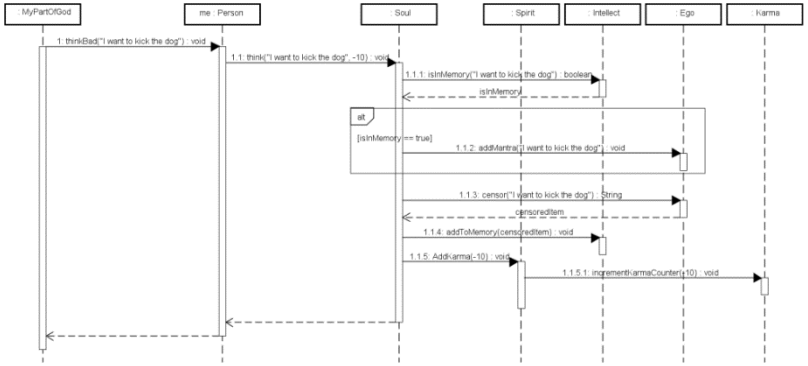
Sequence diagrams

Design and implementation

Class diagram



Sequence diagrams



Source code

```
public class Soul
{
   private Spirit spirit;
   private Intellect intellect;
   private Emotion emotion;
   private Will will;
   private Ego ego;

   public Soul(Spirit spirit)
   {
      this.spirit = spirit;
      Capability capability = spirit.getCapability();
      this.intellect =
         new Intellect(capability.getLifePurpose(),
                       capability.getMaxMemorySize());
      this.emotion = new
          Emotion(capability.getEmotionDispositionFactor());
      this.will = new Will(capability.getWillAmount());
      this.ego = new Ego();
   }

   public void think(String item, int goodnessFactor)
   {
      boolean inMemory = intellect.isInMemory(item);
      if (inMemory)
      {
         ego.addMantra(item);
      }
      String censoredItem = ego.censor(item);
      intellect.addToMemory(censoredItem);
      spirit.addKarma(goodnessFactor);
   }
   // …and more
}
```

Sequence Diagrams

- A sequence diagram shows how objects are working together (collaborate with each other) to realize an Use Case
  - The source is the Analysis Class diagram
  - Shows "snapshots" of the running system
  - Add the time dimension to the sequence of messages send between objects
  - Shows the dynamic interaction between objects
  - The messages send between objects

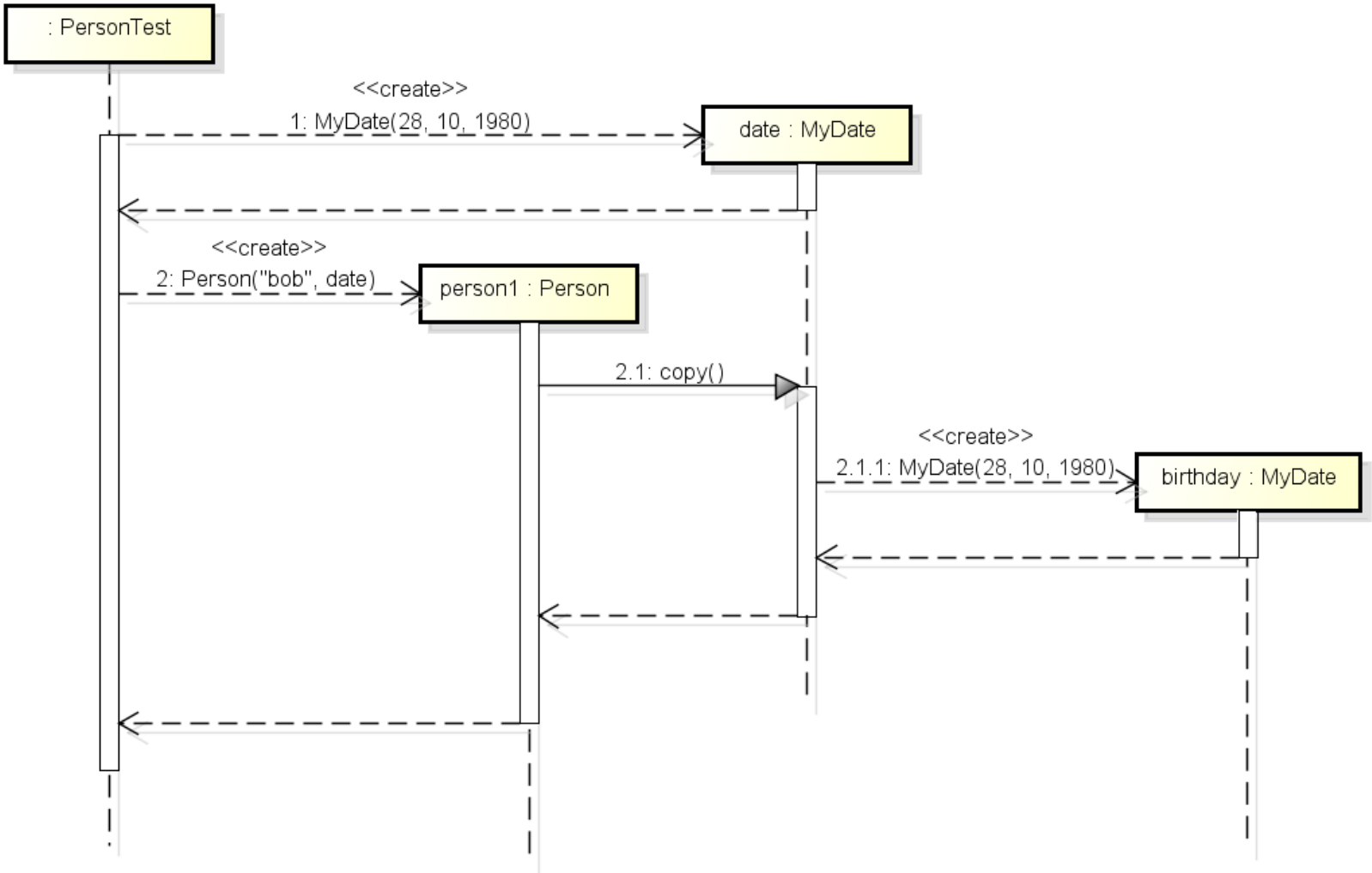
A composition example

```
public class Person
{
  private String name;
  private MyDate birthday;

  public Person(String name, MyDate date)
  {
    this.name = name;
    this.birthday = date.copy();
  }
  public getBirthday()
  {
    return birthday.copy();
  }
  // ...
}
```

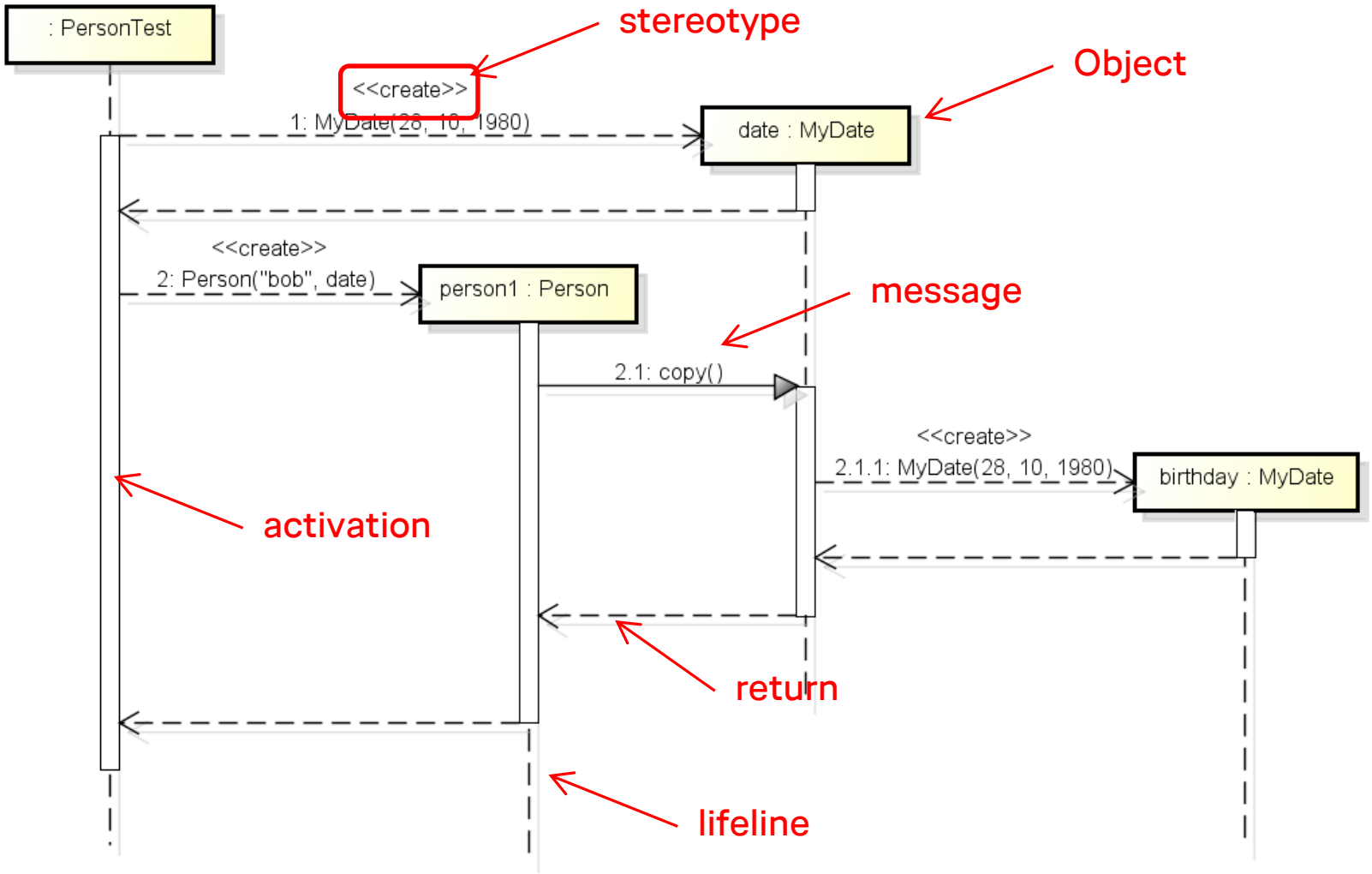
A composition example

```
public class PersonTest
{
  public static void main(String[] args)
  {
    MyDate date = new MyDate(28, 10, 1980);
    Person person1 = new Person("Bob", date);

    // ...

    date = person1.getBirthday();

    // ...
  }
}
```

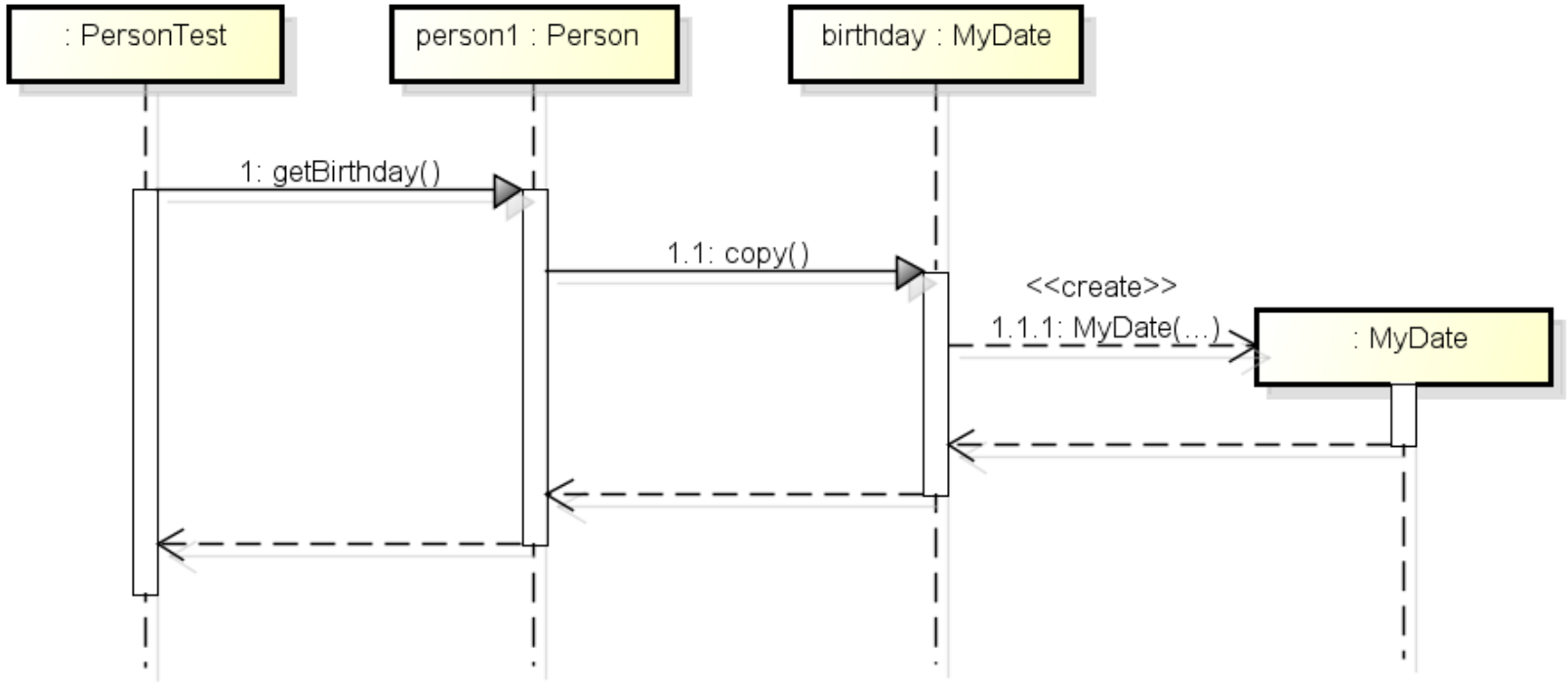
Ex: Composition – constructor



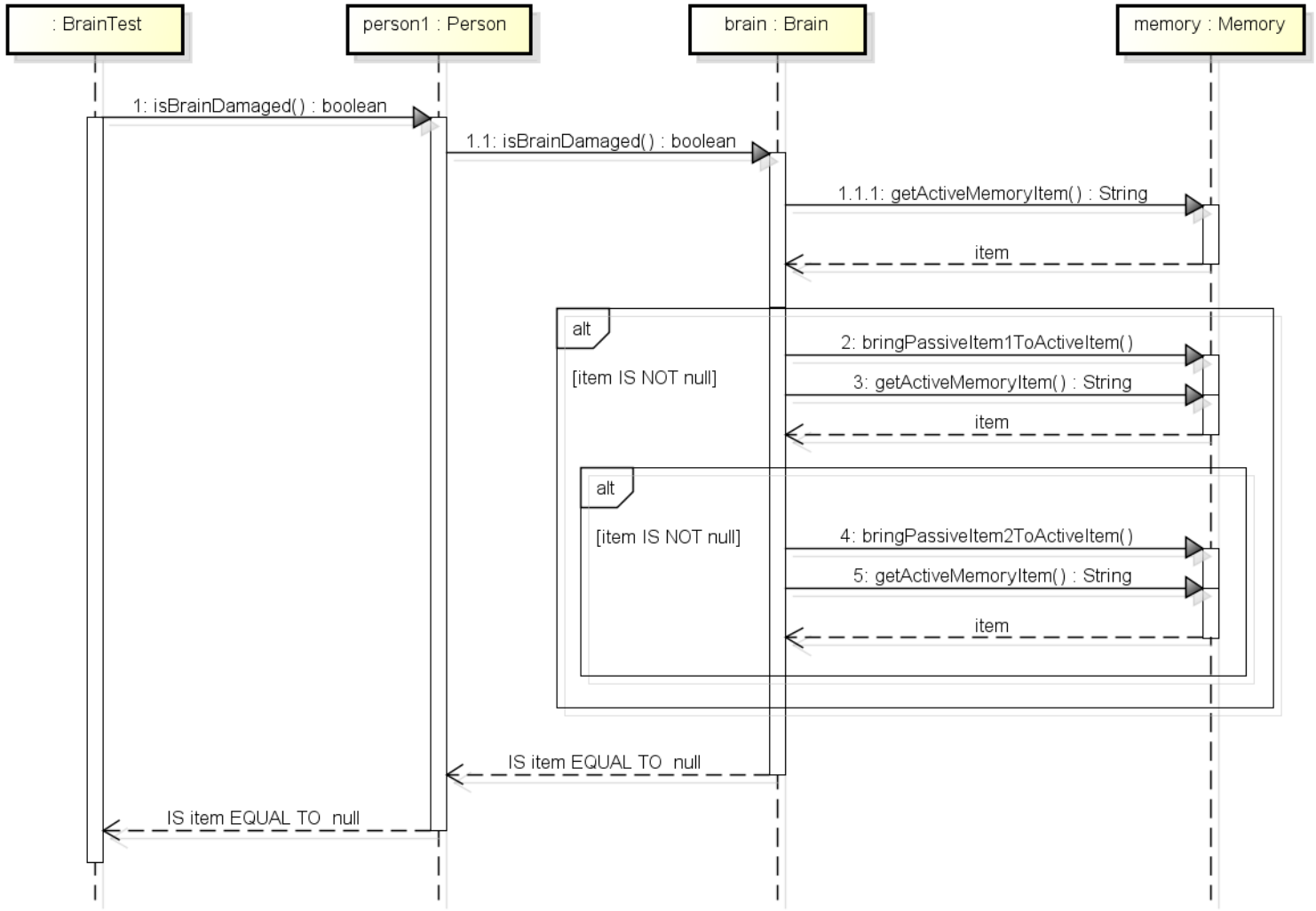
Ex: Composition – constructor



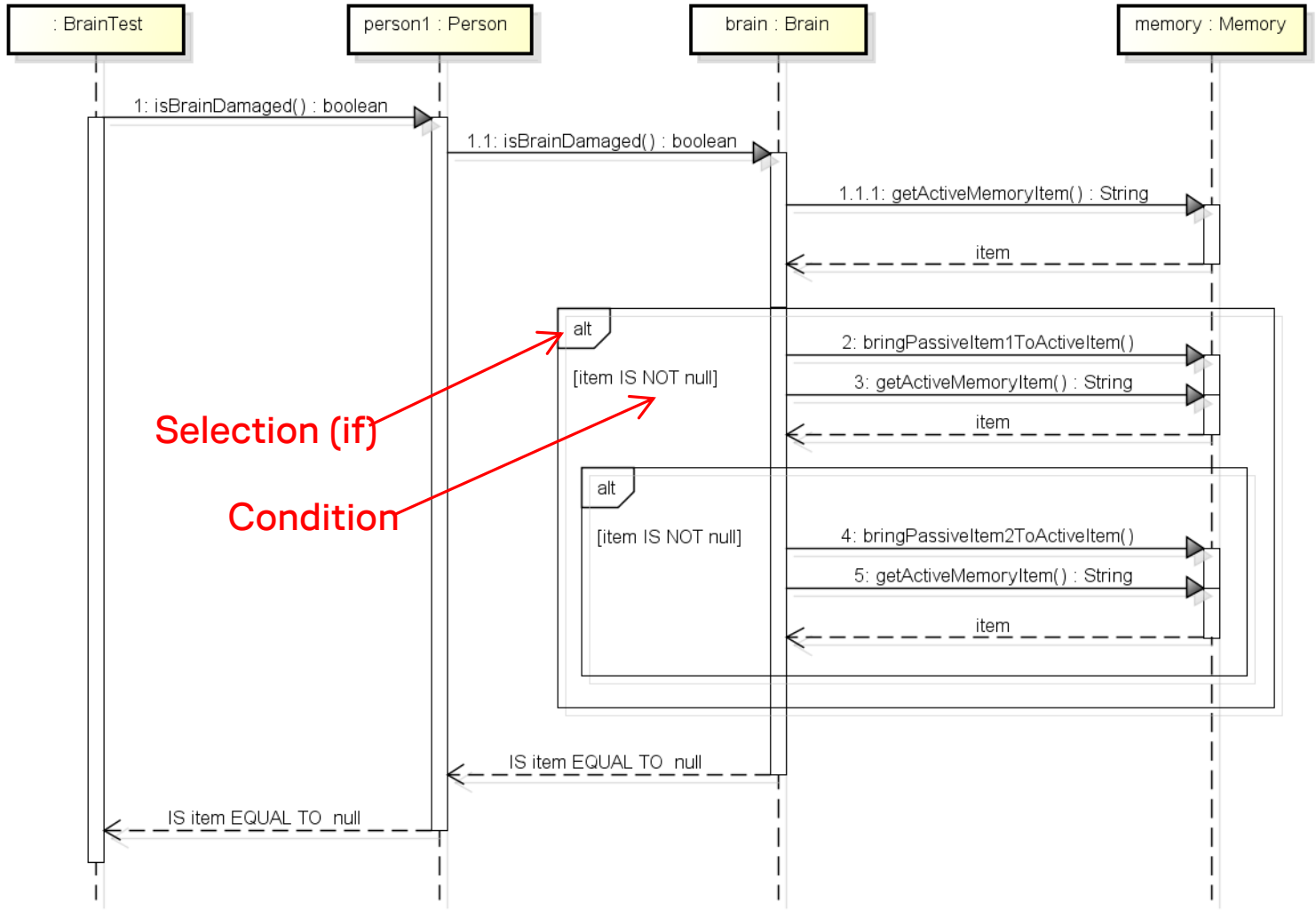
Ex: Composition – getBirthday()



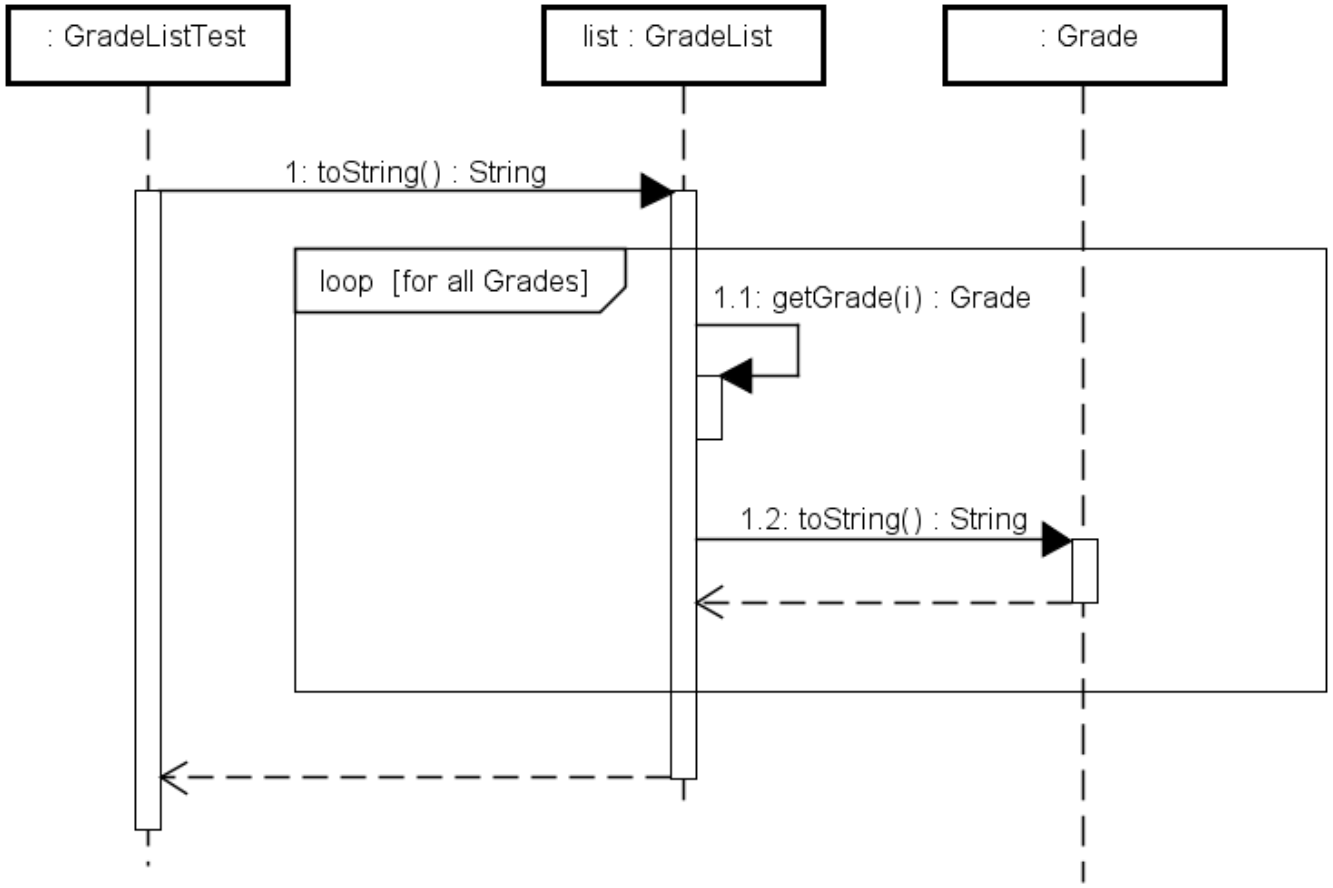
A selection example – nested if



A selection example – nested if



A loop example



A loop example

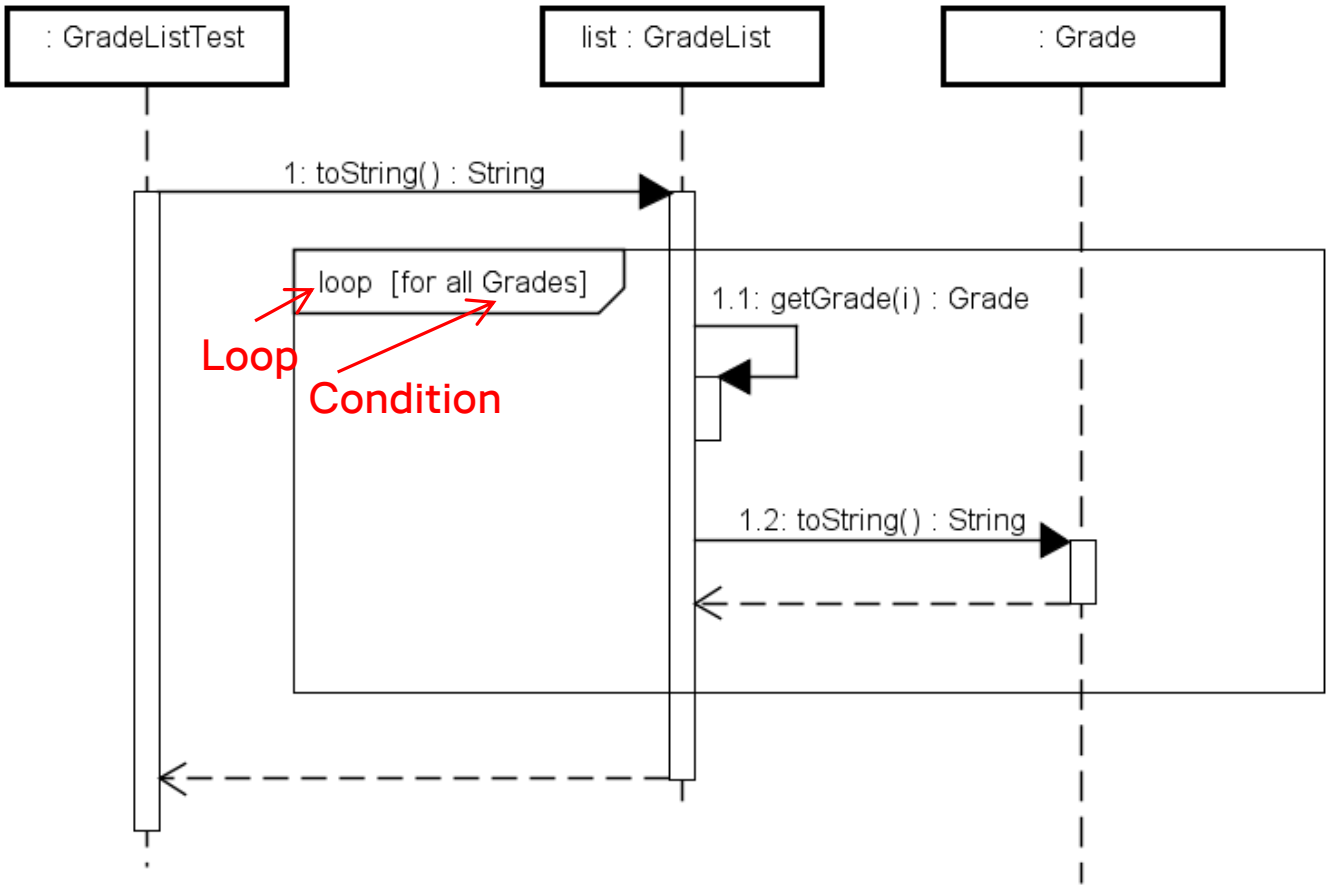
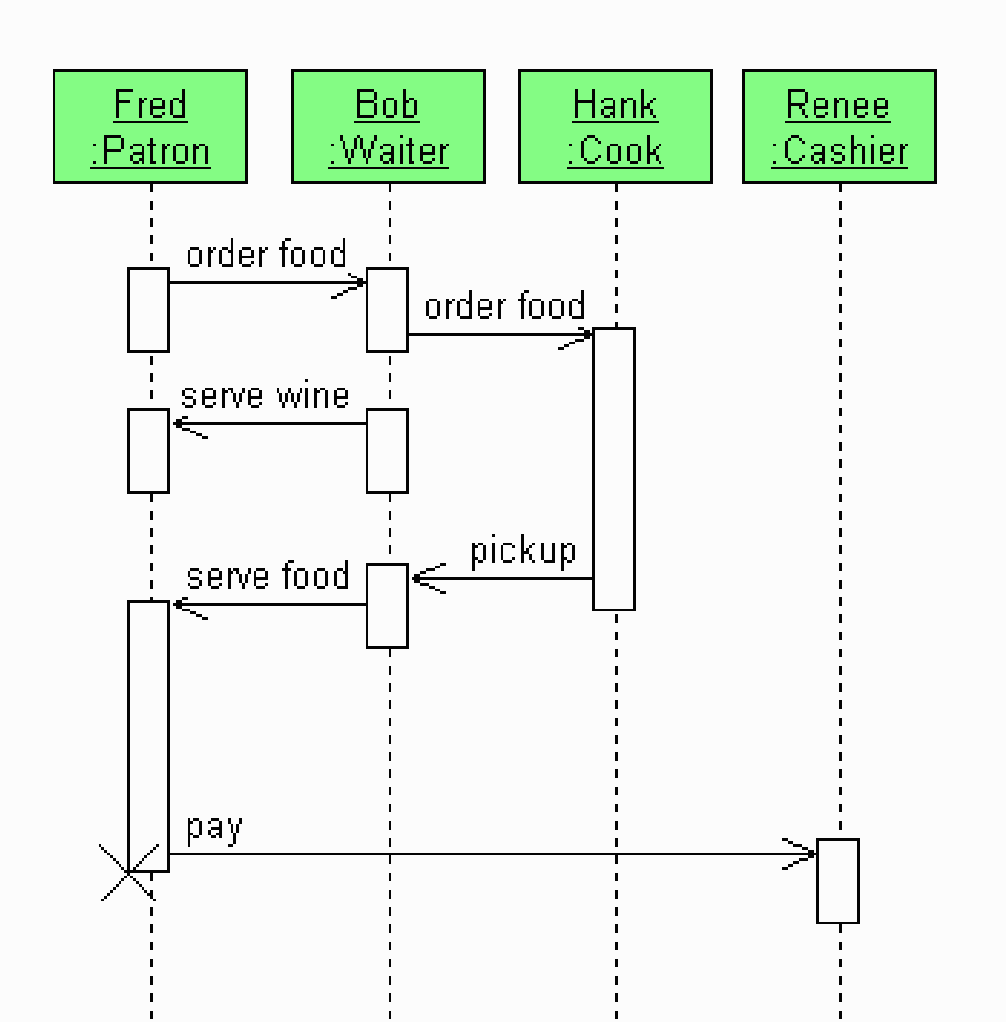
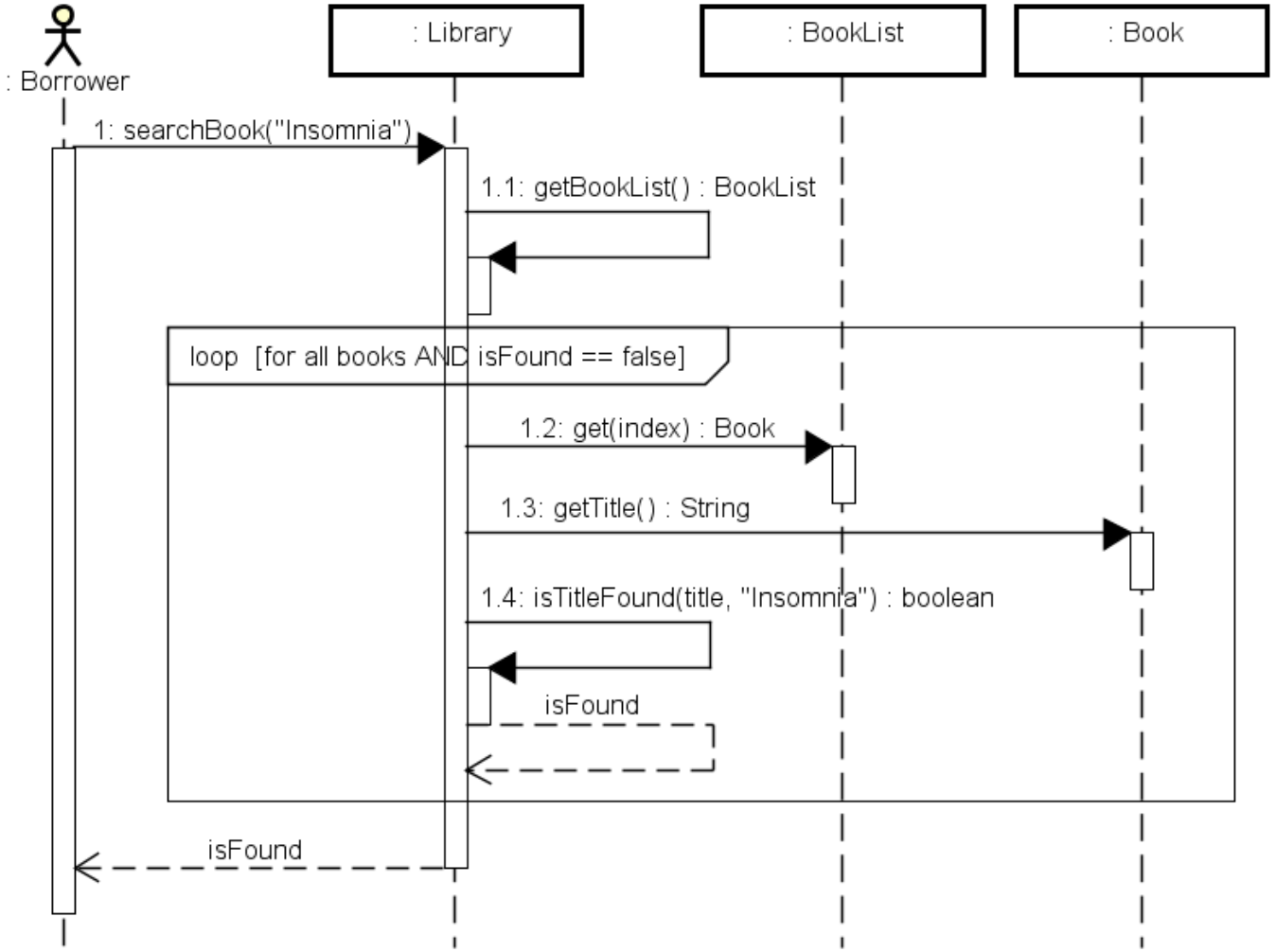


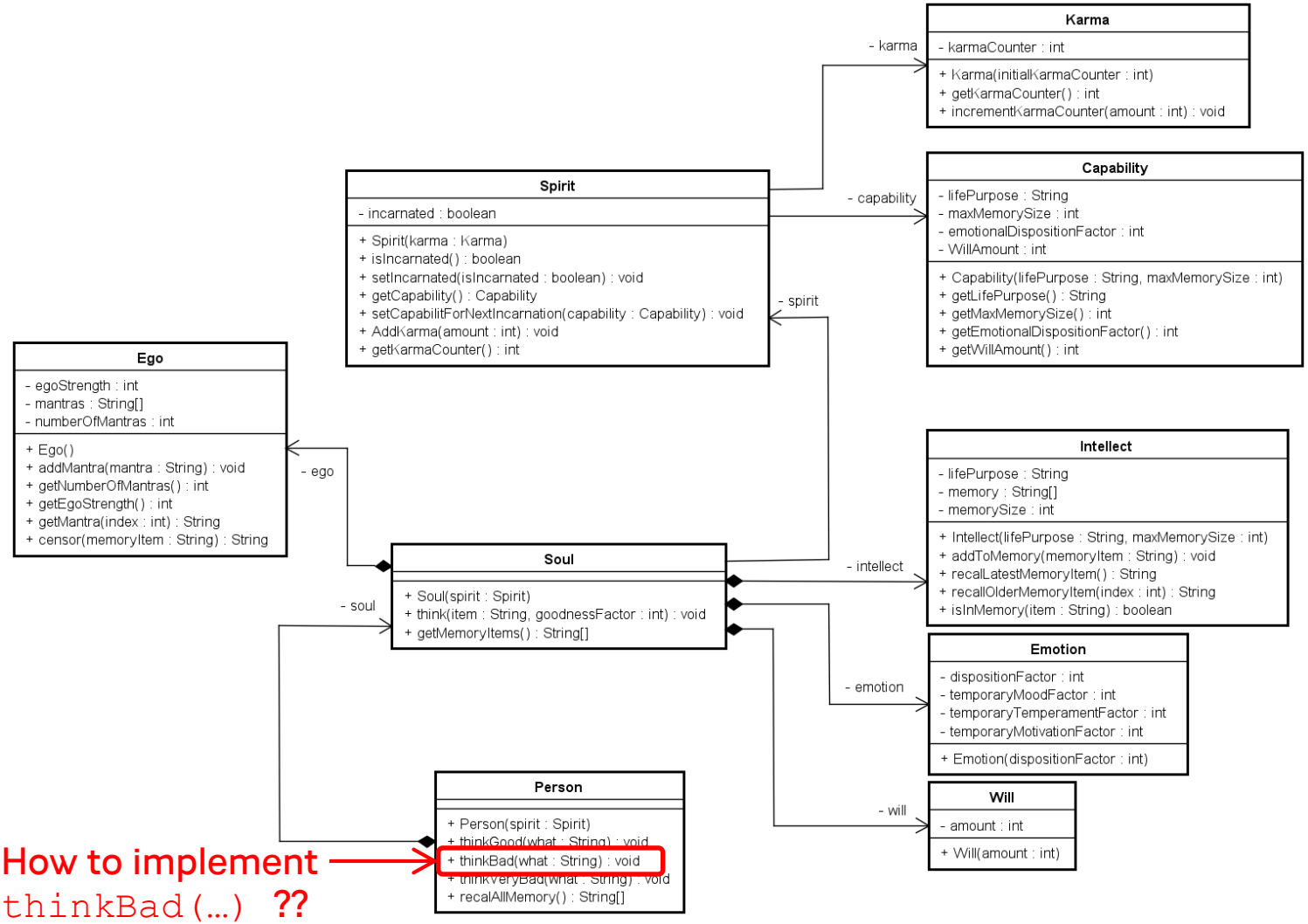
Diagram for a business flow



Sequence diagram for a method

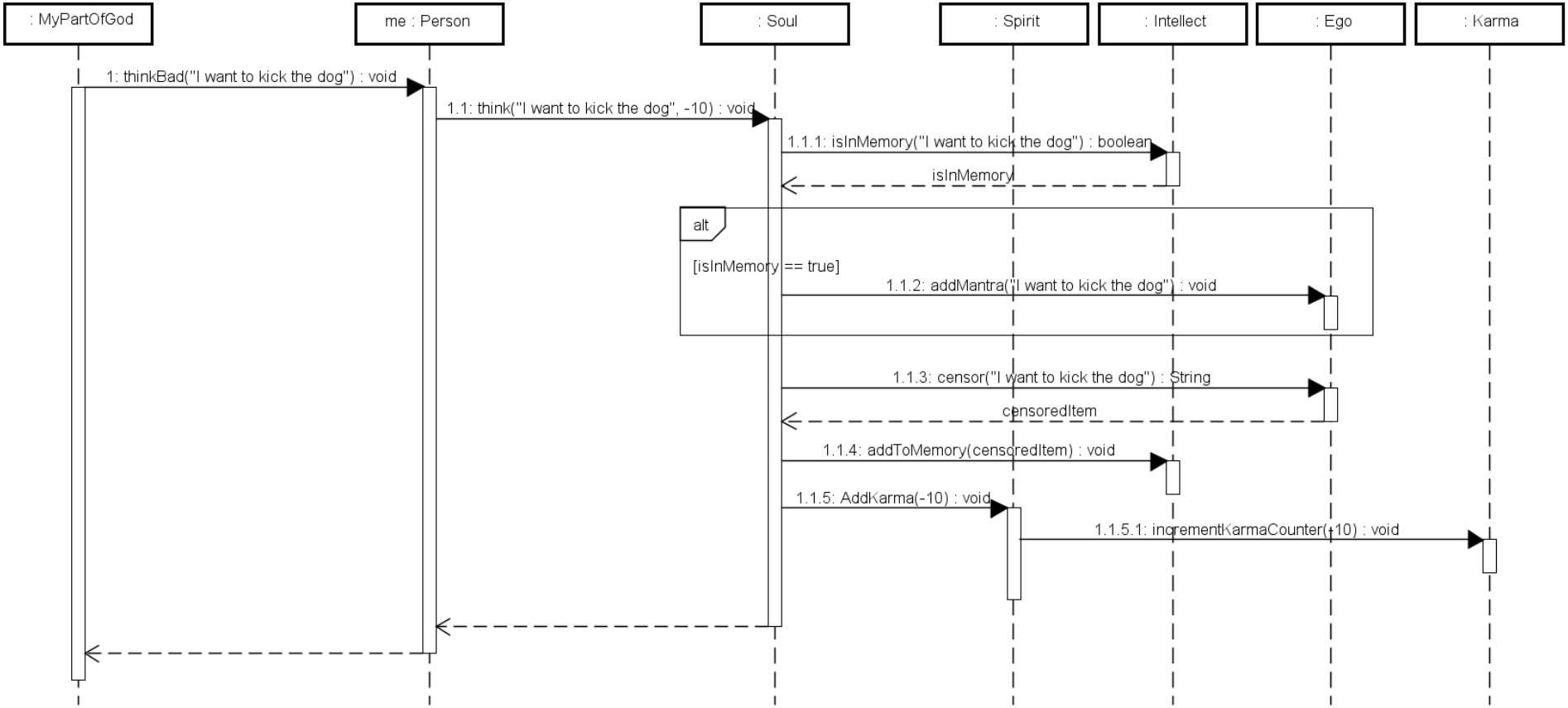


Class diagram: Person-Soul-Spirit



How to implement `thinkBad(…)` ??

Method in Person: thinkBad(..)



What to do now?

- Draw at least one sequence diagram (for a somewhat complicated method)
- Make sure you implement it this way, if not then change the diagram such that diagram and code matches.
- Insert the sequence diagram and companion text into your design section in the project report, if relevant. Alternatively, in appendix with a proper reference in the report.