Inf2-SEPP 2024-25

Tutorial 3 (Week 5)

UML Class and Sequence Diagrams

Study this tutorial sheet and make notes of your answers BEFORE the tutorial.

1 Introduction

In this tutorial you will practice creating UML Class and Sequence diagrams.

2 Creating Class Diagrams

2.1 For a diary system

Consider the OO design of a diary system for university students.

Each diary maintains a list of events. Each event has start and end times and a description. Events in a diary are ordered by their start times. The system allows for there being various kinds of events: labs, lectures and tutorials. For lectures, the system records the name of the lecturer and the title of the lecture. For tutorials, the system records the name of the tutor and the number of students in the tutorial. Use of the diary is optional for the students. By using the diary, students can add a new event, delete an event and check a diary for clashing events. The system allows for the possibility of the same event being added to multiple diaries.

Using the noun identification technique of Lecture 9 part 2, draw a UML class diagram showing the classes and their relationships, as appropriate presenting each class either as a separate box or as the type of an attribute of another class. Include attributes and operations suggested by the above description in class boxes. Also include suitable multiplicities. Show types of attributes, but there is no need to show types for operations.

2.2 For a food box system

Following the system description that was provided to you in Tutorial 2 instructions Task 2, the use case diagram solution to Task 2 a) from Tutorial 2 solutions and the noun identi-

fication technique of Lecture 9 part 2, draw a UML class diagram for the food box system. In doing so:

- Imagine that this system does not have a user interface (for now), and that system activity is through message bursts: an actor instance sending a message to a system object (i.e. calling one of its operations), and the system sending back a response through the operation's return.
- Don't include getters and setters (What are these?)
- Include operation parameter types and return value types
- Use the type `Collection<T>` for collections of objects of type T
- Include association multiplicities
- Include any other notation which you may feel improves the understandability of your class diagram
- Think about the design principles presented to you in Lecture 9 part 1, and be ready to justify your design choices using them.

Hints:

- We are designing a solution for our system, the food box system, here, and we are not interested in how external systems work. You will need to use interfaces to any external systems. Revise interfaces from your Java Inf1-OP course if needed.
- In the use case diagram, we distinguished between a human and a system actor for the supermarket. While we can draw an interface for the external system, how would you manage the registration details of the supermarket which are saved by the supermarket staff?

2.3 (Advanced) For a lift System

Create a Class diagram for the Lift system described in Tutorial 2 instructions Task 3. Add sufficient operations to your classes that you can trace through the message sequences involved in realising the Lift use-cases. Annotate associations between classes with multiplicities. Think about the design principles presented to you in Lecture 9 part 1, and be ready to justify your design choices using them. Can you identify any opportunities for using inheritance?

Hints: The Lift system has a number of physical interfaces to the outside world: to the various buttons, displays showing the lift position, the lift motor for raising and lowering the lift, for example. The Class diagram for the Lift system would be rather impoverished if these interfaces were not explicitly recognised somehow in the diagram. In the *library* example discussed in lecture, some classes like Copy or LibraryMember are *avatars* or *proxies* for real world entities, holding just that information the system needs about each entity. Here, we can do something similar, introducing classes for each electrical/physical interface. We could introduce CallButton and LiftMotor classes, for example. The difference is that we then add

specific operations to these classes for modeling interface events: for a class modeling an input interface, we add an operation whose calls model input events. A CallButton class might have a press() operation that we imagine invoked on an CallButton object representing a particular button, whenever the lift user presses that button. For a class modeling an output interface, we add an operation whose calls model output events being generated. A LiftMotor class might have a startDown() operation whose calls on a LiftMotor object model the system issuing commands to the lift motor to start the lift moving down.

3 Creating Sequence Diagrams

3.1 Warm-up exercise

Draw a UML sequence diagram for the invocation of the `placeOrder` method on some `OrderManager` object as it is shown in the code below. Be sure to show calls of every method and constructor. When a method returns some interesting value, add a reply message with some appropriate annotation to your diagram. To keep things simple, represent a single object of class `OrderPart` which is named `part`.

```
public class OrderManager{
    private ArrayList<Order> orders = new ArrayList<Order>();
    static final int MAX_NUM_ORDERS = 100;

    private void addOrder (Order o) {...}

    public void placeOrder (OrderPart[] parts){
        Order o = new Order();
        for (OrderPart part : parts){//iterates with part over the parts array
            boolean available = part.isAvailable();
            if (available){
                part.reduceStock();
                o.addPart(part);
            }
        }
        addOrder(o);
    }
}
```

3.2 (Advanced) For the lift system

Draw a UML sequence diagram for the 'Call lift' use case identified and described in the Tutorial 2 solutions.

Handling real time

One issue to consider here is the real-time nature of the system. For example, a LiftDoors class might have an close() operation for closing the doors. Does the close() operation return immediately, even though it may take a couple of seconds for the doors to close? It would be very bad if the

close() operation did return immediately and then the system immediately after this invoked an operation to start the lift moving: we could have the lift moving while the doors are still partially open. Two alternatives to think about are

1. Have the close() operation wait for the doors to fully close before returning
2. have the close() operation return immediately, but then have the doors generate a doorsFullyClosed() event when the doors are indeed fully closed. The Doors class would then include a doorsFullyClosed() for modeling such events.

You might try one approach for the doors, another for the lift motion control system.

With the second alternative, the doors or the lift motion control system effectively become actors, and you could describe them as such in your diagram. Alternatively, you could use the Sequence diagram message notation for the events generated by these actors where the message arrow head points to the object receiving the message and the arrow tail is a filled-in circle. This avoids cluttering the diagram with e.g. lifelines for both a Doors actor and a Doors object. Such messages are called *found messages* in UML.

Cristina Adriana Alexandru. 5th Feb 2025.