

Exercises for Object-Oriented Modeling

Marion Brandsteidl, Martina Seidl, Manuel Wimmer,
Christian Huemer, and Gerti Kappel

Business Informatics Group (BIG)
Institute of Software Technology and Interactive Systems,
Vienna Technical University
{brandsteidl|seidl|wimmer|huemer|kappel}@big.tuwien.ac.at

When attending our Object-Oriented Modeling course, the students have to prepare 6 different exercise sheets on the following UML2 diagrams: class, sequence, activity, state, and use case diagram. Each of the exercise sheets contains 6 different exercises; one of them covers the theoretical background and the others are various types of modeling tasks in order to “practically” apply the UML. Those exercises help getting an overview of the various UML concepts and are a good foundation for passing the tests where similar tasks have to be solved. The students must be able to present and explain their solutions to a teacher and the fellow students in groups of about 50 people.

In the following, the exercise sheet on the sequence diagram is shortly described. The first question on the theoretical background can be easily answered when the contents presented in the lecture have been well understood. The second exercise is still at a very abstract level where messages are exchanged which have no specific meaning at all. The students have to prove that they understand the relations between the lifelines and the mechanisms which control the order the messages are exchanged by (i) identifying all possible traces, (ii) introducing combined fragments to change the order or to make certain messages optional or repeat them more than one time. The third exercise gets a little more concrete: here the message exchange of a small Java program has to be depicted in a sequence diagram. One class is derived from class *Thread* in order to make it necessary to model an active object. In the fourth exercise, two design patterns have to be modeled from a textual description. In this context, sequence diagrams are usually applied the other way round: they are used to explain the patterns, but as the textual descriptions are unambiguous, we experienced that those are very good examples for learning the sequence diagram as well as providing a very clear application area with which the students will be confronted. In contrast, the specification of the 5th example is very vague, so the students have to identify the control flow by their own and different solution approaches are possible. Finally, the last exercise illustrates the connection between the class and the sequence diagram. Here a sequence diagram has to be derived from the methods of a given class diagram.

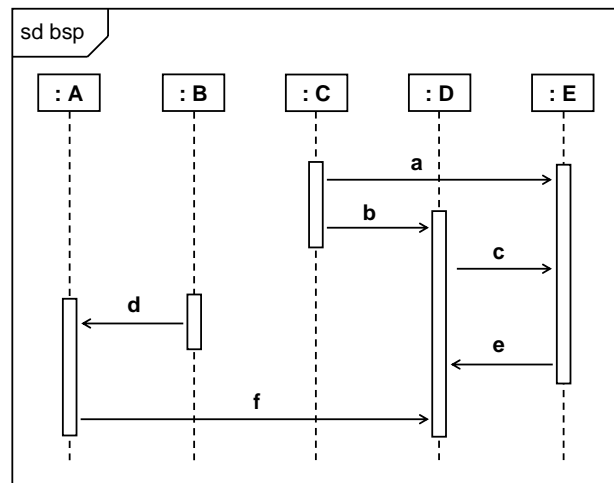
Exercise 1: Theoretical Background

Recapitulate the lecture on the sequence diagram and answer the following questions.

- Name the 4 different kinds of interaction diagrams and describe them briefly. In which situations are such diagrams employed?
- Describe the principle structure of a sequence diagram. Which elements does it contain?
- What is a state invariant? How can time constraints be declared?
- What is an active, what is a passive object? Describe the differences.
- Describe the differences between synchronous and asynchronous message passing.
- Which types of branches and loops may be used within a sequence diagram? Describe the according operators.
- Which operators are provided by the sequence diagram in order to express concurrency and to fix the order of the messages?
- Explain the operators of the group "Filtering and Assertions".

Exercise 2: Traces

You are given the following (simplified) sequence diagram with six messages (*a-f*).



- List all possible interaction sequences (traces).
- Modify the given sequence diagram to make sure that the interaction messages have to occur in alphabetical order.
- Modify the given sequence diagram to make sure that message *d* occurs right after message *c*. List all now possible interaction sequences (traces).
- Modify the given sequence diagram to make sure that the following issues are also covered in the diagram:
 - Introduce a local attribute *x* (Integer).
 - a* and *b* are exchanged if $x=1$, otherwise *c* and *d* are exchanged.
 - c* and *d* are exchanged 14 times.

Exercise 3: Using the sequence diagram to visualize a program flow

You are given the Pseudo-code of the following classes. Visualize the interaction of the three classes in a sequence diagram – start with the *main*-Method.

Note: Keep in mind that class *A* extends class *Thread*. As soon as the method *start* is called, the code of the method *run* is executed parallel.

```
public class Main {
    public static void main(String args[]) {
        B b1 = new B(3);
        int i = b1.m1(3);
        A a = new A(3);

        a.start();

        i = b1.m1(3);
        b1.m2();
        b1.m3();
    }
}

public class A extends Thread {
    private int i;

    public A (int i) {
        this.i = i;
    }

    public void incI() {
        i++;
    }

    public void run () {
        B b = new B(i);

        incI();
        i = b.m1(i);
        b.m2();
        b.m3();
        incI();
    }
}

public class B {
    private int i;

    public B (int i) {
        this.i = i;
    }

    public int m1(int j) {
        return j--;
    }

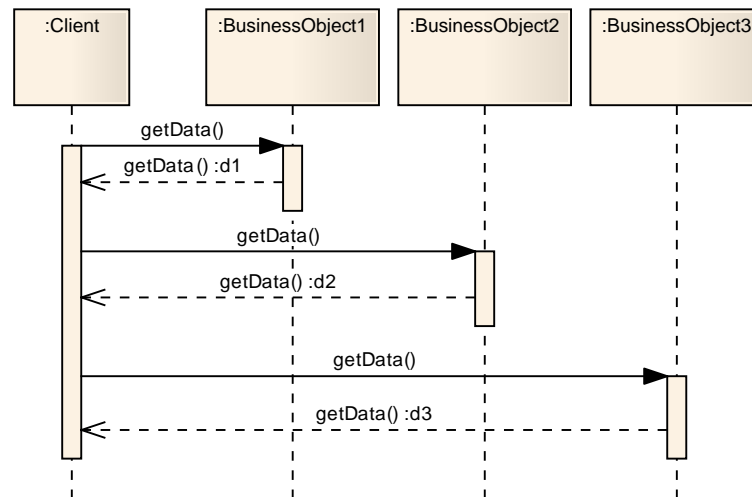
    public void m2() {
        ...
    }

    public void m3() {
        ...
    }
}
```

Exercise 4: Patterns

a) Transfer-Object-Assembler-Pattern

A client in a distributed environment needs different kinds of information provided by 3 business objects. This can be realized as follows:



Following this approach, each client needs knowledge about the business objects in order to obtain the desired data. So the client is strongly bound to the business objects, which is not desirable in general. To resolve those dependencies the Transfer-Object-Assembler-Pattern can be applied. The data of multiple business objects is collected in a transfer object, which is provided to the client. So the client obtains only the required data in an encapsulated manner. More concretely, the pattern is realized as follows:

The **Client** requests via `getData()` the required data from the **TransferObjectAssembler** which creates a **DataTransferObject** and fills it with the data obtained from 3 different **BusinessObjects**. The data of a **BusinessObject** can be also requested with `getData()`. The **DataTransferObject** offers (ev. overloaded) `addData`-methods, which allow to set the data. Finally the **TransferObjectAssembler** returns the **DataTransferObject** to the **Client**.

Model the Transfer-Object-Assembler-Pattern with a sequence diagram.

b) Lazy-Load-Pattern

You develop an application which has to provide huge amounts of data. Since the clients usually need only a very small part of the contents, and since the loading of the complete data is very cost intensive, you decide to implement the Lazy-Load-Pattern.

The **Client** requests a **LazyLoadObject** from the **Supplier** via `getObject(a,b,c)`. The **Supplier** creates a **LazyLoadObject** and initializes it with the parameters **a** und **b** provided by the **Client**. If **c** is set to `true`, then some data is loaded from the **DataSource** via `initLoad()`. Then the **Supplier** returns the **LazyLoadObject** to the **Client**.

Now the **Client** needs the data "X". Therefore it calls `getX()` of the **LazyLoadObject**. If the data is already loaded by the **LazyLoadObject** they are returned immediately, otherwise the necessary data is loaded from the **DataSource**. Finally the data "X" is returned to the **Client**.

Describe this situation with a sequence diagram.

Exercise 5: Cash-Machine

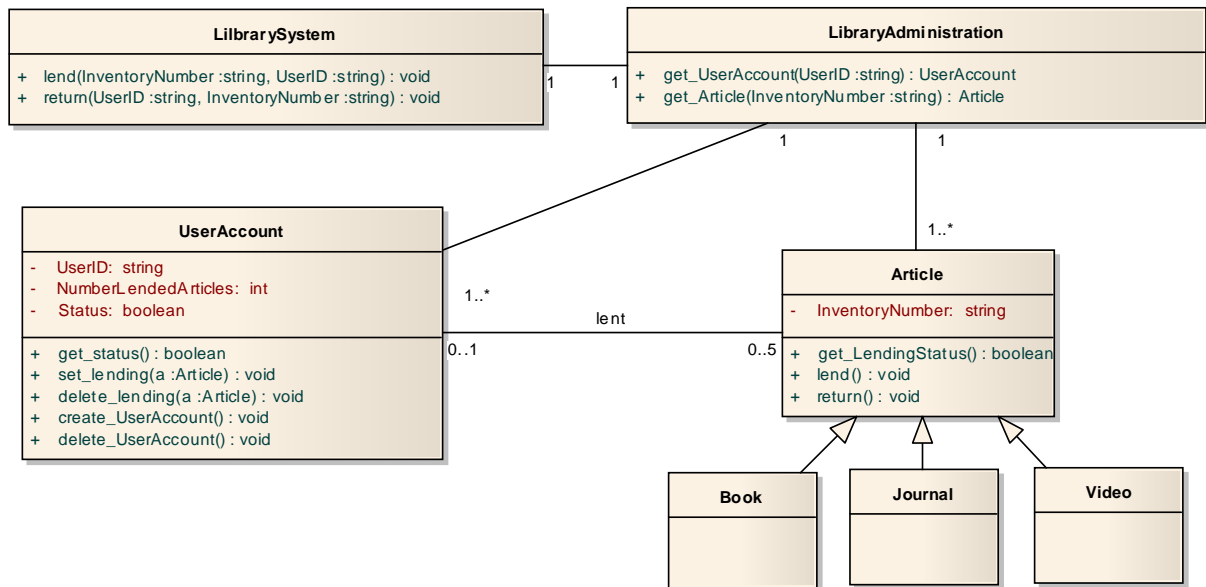
Model the usage of an ATM to withdraw money with a sequence diagram. The usage includes the input of the card, the input of the code, the choice of the amount of money to be withdrawn, and the output of the money. The code may not be entered more than three times. Other error situations need not to be considered. The ATM consists of the following components:

- screen
- keyboard
- card reader
- money output device
- bank server

Exercise 6: Library

The exercise is about a library where you can borrow books, journals, and videos. The articles have to be found, lent, and returned. A user is identified via a user ID, each article is identified via an inventory number. Each user can lend a maximum of five articles at a time. As soon as a user has lent five articles, his/her status is **blocked**. Only if he/she returns an article, the user gets **active** again.

You are given the following (incomplete) class diagram:



Complete the following sequence diagram to illustrate the successful lending of a book.

