Modeling Dynamic Behavior

The following slides use material from:
Craig Larman; Applying UML and Patterns, 3rd Edition;
Prentice Hall
Two types of diagrams can be distinguished:

- UML Sequence Diagrams
- UML Communication Diagrams
Interaction diagrams are used to **visualize the interaction via messages between objects**; they are used for *dynamic object modeling*. 
Modeling the dynamic behavior is often more rewarding w.r.t. understanding the domain than modeling the static structure.
Four types of interaction diagrams are available.

- **Sequence diagrams** (which use a fence format.)
- Communication diagrams (which use a graph or network format)
- Timing diagrams (not discussed)
- Interaction overview diagrams (not further discussed)
Four types of interaction diagrams are available.

- **Sequence diagrams** (which use a fence format.)
- **Communication diagrams** (which use a graph or network format)
- Timing diagrams (not further discussed)
- Interaction overview diagrams (not further discussed)
Java Code for Interaction Diagrams

```java
public class A {
    private B myB = ...;

    public void do1() {
        myB.do2();
        myB.do3();
    }
}
```

Example Sequence Diagram
Java Code for Interaction Diagrams

Example Communication Diagram

```java
public class A {
    private B myB = ...;

    public void do1() {
        myB.do2();
        myB.do3();
    }
}
```
Java Code for Interaction Diagrams

```java
public class A {
    private B myB = ...;

    public void do1() {
        myB.do2();
        myB.do3();
    }
}
```
Lifeline box representing an unnamed instance of class Sale.
Common Notations for UML Interaction Diagrams

Lifeline box representing a named instance (s1) of Sale.

Java Code:
Sale s1 = ...;
Common Notations for UML Interaction Diagrams

Lifeline box representing the class Font, or more precisely, that Font is an instance of class Class - an instance of a metaclass.

Java Code:
```
Class<Font> fontClass = Font.class;
```
Common Notations for UML Interaction Diagrams

Lifeline box representing an instance of an ArrayList class, parameterized to hold Sale objects.

Java Code:
```java
ArrayList<Sale> sales = ...;
```
Common Notations for UML Interaction Diagrams

Lifeline box representing one instance of class Sale, selected from the sales ArrayList<Sale> collection.

Java Code:

```java
ArrayList<Sale> sales = ...;
Sale sale = sales.get(i);
```
Common Notations for UML Interaction Diagrams - Format for Interaction Messages

• “Commonly” Used Grammar:
  return = message(parameter:parameterType):returnType

• Parentheses are usually excluded if there are no parameters.

• Type information may be excluded if unimportant.

Examples

initialize(code)
initialize
d = getProductDescription (id)
d = getProductDescription (id : ItemId)
d = getProductDescription (id : ItemId) : ProductDescription

The same syntax is, e.g., used by the Scala programming language.
UML
Sequence Diagrams
Modeling (Synchronous) Messages

- a found message whose sender will not be specified

- typical synchronous message shown with a filled-arrow line

- execution specification bar indicates focus of control
  (optional)
  dt. Ausführungssequenz
Modeling (Synchronous) Messages

UML Superstructure

If the Message represents a CallAction, there will normally be a reply message from the called Lifeline back to the calling lifeline before the calling lifeline will proceed.
Self messages can be modeled using nested execution specification bars.
To show the return value of a message you can either use the message syntax (A) or use a message line at the end of an execution specification bar (B).

**Variant A**

```
:Register
theReport = report
:Sale
```

**Variant B**

```
:Register
report
theReport
:Sale
```

**Execution Specification Bar (dt. Ausführungssequenz)**

- Variant A
- Variant B
Object Instance Creation

The name `create` is an **UML idiom**; it is not required.

Newly created objects are placed at their creation "height".
Object Instance Destruction

The object destruction notation is also used to mark objects that are no longer usable.

Not strictly required by the UML.
Invoking Static Methods (Class Methods)

Beware, other notations are also used (e.g. underlined method names).
public class Register {
    public void report() {
        Locale[] locales = Calendar.getAvailableLocales();
    }
}
Diagram **frames** in UML sequence diagrams are used to support - among others - conditional and looping constructs.

**Example**

```plaintext
t = getTotal
st = getSubTotal
lineItems[i] : SalesLineItem
```
Diagram frames in UML sequence diagrams are used to support - among others - conditional and looping constructs. Frames have an operator and a guard.
How to model the iteration over a collection?

Modeling task: Calculate the total of a sale by summing up the sub totals for each sales line item.
Use a **UML loop frame** to iterate over a collection.

Modeling task: Calculate the total of a sale by summing up the sub totals for each sales line item.
Use a **UML loop frame** to iterate over a collection.

Modeling task: Calculate the total of a sale by summing up the sub totals for each sales line item.
public class Sale {

    private List<SalesLineItem> lineItems = new ArrayList<>();

    public Money getTotal() {
        Money t = new Money();
        Money st = null;
        for (SalesLineItem lineItem : lineItems) {
            st = lineItem.getSubtotal();
            t.add(st);
        }
        return t;
    }
}

Modeling task: Calculate the total of a sale by summing up the sub totals for each sales line item.
How to model the sending of a message only if a guard condition matches?

Modeling task: Get the sum of all sales that happened today after 18:00 o’clock.
Use a **UML opt frame** to model the sending of a message if the guard condition matches.

**Modeling task:** Get the sum of all sales that happened today after 18:00 o’clock.
How to model mutually exclusive alternatives?

Modeling task: A register should be able to handle credit card payments and cash payments.
Use the **UML alt frame** to model between 2 and n mutually exclusive alternatives.

Modeling task: A register should be able to handle credit card payments and cash payments.
Diagram frames in UML sequence diagrams are used to support - among others - conditional and looping constructs.

In general, sequence diagrams are only well suited to show simple looping and conditional behavior.

Activity diagrams and code may be alternatives.

Modeling task: A register should be able to handle credit card payments and cash payments.
UML Communication Diagrams
Links and Messages in Communication Diagrams

- **A link** is a connection path between two objects (it is an instance of an association)
  A link indicates that some form of navigation and visibility between the objects is possible.

- **Each message** between objects is represented with a message expression and a small arrow indicating the direction of the message
  Sequence numbers are added to show the sequential order of messages in the current thread of control; the starting message is often not numbered.
• Modeling self messages

```
1: verify →
```

```
makePayment ↓
```

```
:Register
```
Create message, with optional initializing parameters. This will normally be interpreted as a constructor call.

If an unobvious creation message name is used, the message may be stereotyped for clarity.
Message Number Sequencing
The initial message is not numbered to make the numbering easier to comprehend.

The diagram illustrates the sequence of messages between objects A, B, C, and D. The messages are numbered as follows:

1. First message: i_msg → A
2. Second message: j_msg → B
3. Third message: k_msg ↓ B
4. Fourth message: l_msg ↑ B
5. Fifth message: m_msg ↓ B
6. Sixth message: n_msg → C

The numbering helps to understand the flow and sequence of messages in the communication diagram.
The message is only sent if the condition evaluates to true. The condition is written in square brackets. In case of modeling mutually exclusive message conditional path letters are prepended.
Messages to Class Objects

makePayment ↓

1: locs = getAvailableLocales →

e.g. to format the date as either 3/2/2009 or 2.3.2009
"Real-world complex example"
UML Communication vs. UML Sequence Diagrams
# Strengths and Weaknesses Interaction Diagrams

<table>
<thead>
<tr>
<th>Type</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Diagram</td>
<td>✓ clearly shows sequence or time ordering of messages</td>
<td>– forced to extend to the right when adding new objects; consumes horizontal space</td>
</tr>
<tr>
<td></td>
<td>✓ large set of detailed notation options</td>
<td></td>
</tr>
<tr>
<td>Communication Diagram</td>
<td>✓ space economical - flexibility to add new objects in two dimensions</td>
<td>– more difficult to see sequence of messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– fewer notational options</td>
</tr>
</tbody>
</table>
## Strengths and Weaknesses Interaction Diagrams

<table>
<thead>
<tr>
<th>Type</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Diagram</td>
<td>✓ clearly shows sequence or time ordering of messages</td>
<td>– forced to extend to the right when adding new objects; consumes horizontal space</td>
</tr>
<tr>
<td>Communication Diagram</td>
<td>✓ space economical - flexibility to add new objects in two dimensions</td>
<td>– more difficult to see sequence of messages; fewer notational options</td>
</tr>
</tbody>
</table>
The goal of this lecture is to enable you to systematically carry out small(er) software projects that produce quality software.

- Modeling the dynamic behavior is often more rewarding than modeling the static structure w.r.t. understanding a domain.
- Modeling the dynamic behavior is often particularly useful if the control-flow is more involved; but only draw the part that is relevant to understand the problem at hand.
- The UML is often used informally - this is OK if everyone interprets the diagrams in the same way.
The goal of this lecture is to enable you to systematically carry out small(er) commercial or open-source projects.