Object Oriented Languages – Story of development (1)

- First object oriented language in year 1967:
  - SIMULA-67: first object oriented language (OO)
  - Application area: simulation
  - No practical meaning
- 70’s:
  - SMALLTALK: till the mid 80’s the only OO language with practical application

It is no coincidence that the first OO programming language comes from the domain of modeling and simulation. The OO paradigm is very well suited for modeling real, physical states.
Since 1988 fast dispersal of OO programming languages
  - Eiffel: a completely OO language
  - Different OO extensions:
    - C -> C++
    - Pascal -> Object Pascal (Delphi)

Present state:
  - Due to its compatibility to C, C++ prevailed on a large base
  - Smalltalk is increasingly losing its importance
  - Java is established as a language for the development of web applications, but is not restricted only to this domain
Object oriented programming

- The object-oriented paradigm is based on the concept that the world is built of a heap of objects that communicate with each other by messages.

- An object is described by data, that characterizes its attributes, and by methods, that determine the access on data.

- The methods spectrum of an object determines how the objects react on messages it receives. Access on object data is possible through the object methods.
Advantages of object-oriented programming languages

- Reusability of existing program parts and programs by inheritance and class libraries.

- Avoidance of duplicates in code by reusability.

- Code maintenance is done by adjusting only small parts of the program.

- Abstract classes are the basis for consistent interfaces for other classes.

- Programming errors can be avoided due to the high, problem-oriented abstraction level.
Disadvantages of Object Oriented Programming Languages

- Long orientation time in object oriented thinking
- Knowledge about algorithm formulation is not enough.
- The inner structure of classes and libraries must also be known in order to be able to extend the programs in a correct way.
Recommended literature (1)

[Balz05] Helmut Balzert: „Lehrbuch Grundlagen der Informatik“, Spektrum Akademischer Verlag, 2005


Recommended literature (2)


[Stud05] Rudi Studer: „Grundlagen der Informatik 1, SS 2005“, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Uni-Karlsruhe, 2005


Outline

Lecture Content

1. Preface
2. Basics
3. Object orientation

3.1. Introduction and basic concepts
3.1.1. Introduction
3.1.2. Objects and classes
3.1.3. Attributes of object orientation

3.2. Object-oriented modelling with UML
3.2.1. Intro
3.3.2. Case Application models
3.2.3. Static models
3.2.4. Dynamic models
3.2.5. summary

3.3. Methods of the OO Analysis and Design
Definition 2.1: Object orientation (OO) serves as the description of problem classes, where the single components are modeled through objects. The description includes not only the characteristics (attribute), but also the corresponding operations (methods).

- Object orientation is not only used in programming, but also in:
  - Modeling and simulation of complex technical systems
  - Analysis and Software-Design
  - Data base technologies
  - Design of user interfaces
Goal of Object Orientation

• „Natural“ Modeling and Design
  - Abstraction concepts of OO are based on reality (through classes and objects)
  - Metaphors for relationships and behavior: association, inheritance, message exchange, polymorphism
  - In contrast to procedural programming, OO methods are integrated in the different phases of the development process: analysis, design, and realization

• Re-use of the source code
• Re-use of OO designs: design pattern
Basic principle of object orientation

- Mapping of parts of the real world into a program:
  - The real problem is illustrated through a program
  - The illustrated elements of the program are related to each other and can interact with each other
Terms (1)

• An **object** is a theoretic or real entity of the environment or in the software. Objects are characterized by [Schn98]:
  - **Identity**: specification, resp. instance of a class
  - **Data aspect**: attributes, resp. properties
  - **Activity aspect**: behavior, given by the methods of the object

• A class is an **entity**, by which the **properties** and the **behavior** of the objects is described. Classes can be in a **class hierarchy**. [Schn98]

• The **attribute** is a property of an entity provided with a name. Usually, its meaning is given by its name [Schn98].
• A **method** is a program fragment, where a received message is processed and the inner state of the addressee is modified, accordingly. Methods can be compared with the procedures of the procedural programming languages and are assigned to an object. [Schn98]

• A **message** is an entity, from which all **interactions between objects** are derived. It is composed of the **specification of the addressee** (an object), a **method** that can be processed by the addressee, as well as the required **parameters** [Schn98].

• **Inheritance**: propagation of properties (attributes, methods) to another class or to another object [Schn98].

• **A class library** is a collection of reusable software components that use the constructs of the object oriented programming paradigm. [Schn98]
Outline

Lecture Content

1. Preface

2. Basics

3. Object orientation
   3.1. Introduction and basic concepts
      3.1.1. Introduction
      3.1.2. Objects and classes
      3.1.3. Attributes of object orientation
   3.2. Object-oriented modelling with UML
      3.2.1. Intro
      3.3.2. Case Application models
      3.2.3. Static models
      3.2.4. Dynamic models
      3.2.5. summary
   3.3. Methods of the OO Analysis and Design
Objects

• An **object is an existing „thing“** from the problem area of the software (the user world)

• An object has a **defined behavior**
  - Behavior is comprised from a number of exactly defined operations to fulfill a task
  - One such operation is performed as soon as a message is received

• An object has an **inner condition**
  - The state of an object is private, that means, it is known only inside the object
  - The consequence of the operation of an object (by receiving a message) is dependent on the current state of the object

• An object has an **explicit identity**
  - The identity of an object is independent of its other characteristics
  - Multiple different objects with identical behavior and identical inner conditions can exist together in the same system
State – behavior – identity: example

3.1. Introduction and Basic Concepts
3.1.2 Objects and classes

Source: [Booc91]
Example: Object „Upper arm of a robot“ (1)

• Labeling of the object »Upper Arm of a Robot« by three attribute values (character)

• Manipulation of the angle using four methods (operations)

• **Attributes:**
  - Current angle
  - Maximal angle
  - Minimal angle

• **Operations:**
  - *Adjust grand position*, parameter: angle
  - *Report current position*, return value: angle
  - *Adjust new position*, parameter: angle
  - *Initialize min and max angle*, parameter: min and max angle
Example: Object „Upper arm of a robot“(2)

- **Behavior of the object „Upper Arm of a Robot“:**
  - Reaction to messages such as `report current position`, `initialize min angle`

- The condition of the upper arm is given through its characteristics (current, max, and minimum angle)

- The object „Upper Arm of a Robot“ exists independently from its characteristics
A class is the building plan of an object

- Classes correspond to data types
- In the class, the properties of the objects are defined, as well as their behavior. The object is a concrete specification of the class
  - Class: “the car has a color“, Object: “the color of the car is silver“
- There are several objects of a class, each object can be identified unambiguously.
• **A class** is a description of one or more similar objects. „Similar“ means that a class describes only one object of a certain type.
• A class describes the **construction**, the **processing possibilities** and the possible **behavior** of objects of this class.

• A class definition is comprised of:
  - Definition of the attributes of the class (local criteria)
  - Definition of the relationship to other classes
  - Definition of the operations that are possible for the objects of the class or on the class itself.
Classes have the following relationships to one another:

- **Association**: The objects in one class are associated with objects of another class. Often also the other way around - that means, that associations are frequently bi-directional
  
  - A **car** has an **engine**, an **engine** is part of the **car**

- **Usage**: The objects of a class use attributes or operations of another class to provide for their own attributes and operations
  
  - A **human being** is driving the **car**

- **Inheritance**: A class is a sub class of another class. It «inherits» all attributes and operations of this class, that means, all objects of the class possess them without having to be defined in the local class
  
  - A **car** inherits all the properties of the class **vehicle**.
Attributes can be objects of other classes.

In a class, this circumstance is described by a relation to another class.

An object is at this point a reference to another object.
Representation of classes in class diagrams

According to UML notation the class is represented in a rectangle and is comprised of:

- **Class name**: Naming of a class from the language area of application field. As the rule, a main word or an adjective (e.g., „vehicle“ or „civil vehicle“)

- **Attribute**: A data value (e.g., position, size, color,...). The objects of a class have the data value of this class

- **Operation**: A function (e.g., delay, delete, change size,..). Application to objects of a class
An **object attribute of a class** is the description of a **data element** that is available in each object of the class. The name of the object attribute is given in the **description of the class**.

An object carries an individual and unchangeable attribute value for each object attribute.

**Notation:**

- **Class**
  - `Attribute_1`
  - `Attribute_n`

- **Object : Class**
  - `...`

**Example:**

<table>
<thead>
<tr>
<th>Team meeting</th>
</tr>
</thead>
<tbody>
<tr>
<td>title: String</td>
</tr>
<tr>
<td>beginning: Date</td>
</tr>
<tr>
<td>duration: int</td>
</tr>
</tbody>
</table>
Object diagram with examples for object attributes

### Team meeting

- **title**: String
- **beginning**: Date
- **duration**: int

### ar12: team meeting

- **title**: "12. Department discussion"
- **beginning**: 5-10-98 10:00
- **duration**: 120 min

### ar13: team meeting

- **title**: "13. Department discussion"
- **beginning**: 5-17-98 10:00
- **duration**: 120 min

### ar14: team meeting

- **title**: "exhibition preparation"
- **beginning**: 5-12-98 9:00
- **duration**: 180 min

\[ O \rightarrow C \]

means "O is an instance of C"
The classes define the static structure of the system. The dynamic structure (the structure during runtime of the program) reflects the static structure. The relations between classes mould the possible relations between their objects. An object can have a relation to another object only if the classes of these two objects have defined these relations.
Outline

Lecture Content

1. Preface

2. Basics

3. Object orientation
   3.1. Introduction and basic concepts
      3.1.1. Introduction
      3.1.2. Objects and classes
      3.1.3. Attributes of object orientation
   3.2. Object-oriented modelling with UML
      3.2.1. Intro
      3.2.2. Case Application models
      3.2.3. Static models
      3.2.4. Dynamic models
      3.2.5. summary
   3.3. Methods of the OO Analysis and Design
Properties of object orientation (1)

• **Enclosure:**

  Data enclosure allows the separation of the internal implementation of objects to external access. Access is granted only via an predefined interface for making it independent of implementation details.

• **Instantiation:**

  Instantiation means the creation of an object of a specified class, that represents its (abstract) description. This instance of the class is a concrete occurrence of that class.

• **Inheritance:**

  Classes can be derived from other classes. Each class inherits the attributes and methods from the upper class.

• **Polymorphism:**

  The concept of polymorphism means, that properties or methods of a class can be referenced by objects, without knowing the concrete specification of the class inside those respective objects.
Properties of object orientation (2)

- Enclosure
- Instantiation of classes
- Inheritance
- Polymorphism

3.1. Introduction and Basic Concepts

3.1.3 Attributes of object orientation

Spring-Mass-System

The springs are interchangeable

Ideal spring

This object oriented model of a spring-mass-system will explain the features of object orientation

Instead of ideal (undamped) springs, the spring-mass-system can have a damped spring
Properties of object orientation

• **Enclosure:**
  
  Data enclosure allows the *separation of the internal implementation of objects* to external access. Access is granted only via a predefined interface for making it independent of implementation details.

• **Instantiation:**
  
  Instantiation means the *creation of an object of a specified class*, that represents its (abstract) description. This instance of the class is a concrete occurrence of that class.

• **Inheritance:**
  
  Classes can be derived from other classes. Each class inherits the attributes and methods from the upper class.

• **Polymorphism:**
  
  The concept of polymorphism means, that *properties or methods of a class can be referenced by objects, without knowing the concrete specification of the class inside those respective objects.*
In object oriented programming language, a class is defined through the **summary of a heap of data and methods** that operate on it.

The data is represented through a set of variables, that have been newly created for each instantiated class (noted as *attribute*, member variables, instance variables or *instance criteria*).

The operations (methods) are available in executable programming code only once. However, they can operate at each call on the data of a certain object.

---

**A message** is an entity, on which all interactions between objects are based. A message is build of the indication of the respective **object** and an **operation**, that can be performed by that object, as well as the required **data**. [Schn98].
• The **attributes** represent the **state of an object**. They can **differ in each instance of a class** and **change during their lifetime**

• Objects can hide their attributes, making invisible to the outside world these attributes, as well as their values

• Objects encapsulate their data and allow data access only via predefined interfaces

• The **methods** represent the **behavior of the object**

• The **behavior of an object** of a class is determined by the **definition of its methods in the class** and is dependent on the corresponding program code and the current state of the object

• This **summarization of methods and variables in a class is known as enclosure**
The object „spring“ encapsulates its attributes: spring flanges, spring constant, spring length as well as the spring equation. The object contains and administrates independently the knowledge on its physical behavior.

- The modeling- and simulation (M&S) system has no direct access to the attributes of the object (example: the spring equation)
- The system can determine the physical behavior of the object only by sending it an message (example: “compute equation”)
- The message is send by calling the respective method, example `Spring.ComputeEquation()`.

```model Spring
    Flange a „left flange“ ;
    Flange b „right flange“ ;
    parameter Real c „spring constant“ ;
    parameter Length l „spring length“ ;
    equation
        b.f - a.f = c * ( (b.s - a.s) - l ) ;
end Spring;
```
• The **package-concept**:
  - All classes in Java are organized in packages. Packages can be **nested up to any depth** (ex. `java.awt.image`).
  - The package-structure is shown in the filesystem through the **folder structure** (ex. `/java/awt/image`).
  - A package has his **own namespace** and the classes in a package can be accessible either only **locally** (from the package – **private** class) or from **outside** the package (**public** class).
  - Classes from another package must be **imported explicitly** (ex., `import java.awt.Button`). **All classes** from the package `java.lang` are imported by default.
Properties of object orientation

• **Enclosure:**
  Data enclosure allows the separation of the internal implementation of objects to external access. Access is granted only via an predefined interface for making it independent of implementation details.

• **Instantiation:**
  Instantiation means the creation of an object of a specified class, that represents its (abstract) description. This instance of the class is a concrete occurrence of that class.

• **Inheritance:**
  Classes can be derived from other classes. Each class inherits the attributes and method from the upper class.

• **Polymorphism:**
  The concept of polymorphism means, that properties or methods of a class can be referenced by objects, without knowing the concrete specification of the class inside those respective objects.
3.1. Introduction and Basic Concepts

3.1.3 Attributes of object orientation

- Creation of an object (an instance) of a class
- This instance is a concrete specimen of that class
• The model of the spring-mass-system is created by instantiating (creating objects) of the given classes Flange, Spring and Mass,
• These objects are interconnected to a complete system
• Each object can be initialized with the corresponding parameters, ex. the spring with its constant c=k

```
model SpringMassSystem
  parameter Mass m;
  parameter Real k;
  Flange F;
  Spring S (c = k);
  SlidingMass M (mass = m);
  equation
    connect (F.a, S.a);
    connect (S.b, M.a);
end SpringMassSystem;
```
Properties of object orientation

- **Enclosure:**
  Data enclosure allows the *separation of the internal implementation of objects* to external access. Access is granted only via an predefined interface for making it independent of implementation details.

- **Instantiation:**
  Instantiation means the *creation of an object of a specified class*, that represents its (abstract) description. This instance of the class is a concrete occurrence of that class.

- **Inheritance:**
  Classes can be derived from other classes. Each class inherits the attributes and methods from the upper class.

- **Polymorphism:**
  The concept of polymorphism means, that properties or methods of a class can be referenced by objects, *without knowing the concrete specification of the class inside those respective objects.*
• Through **inheritance**, a class **will not be completely newly defined**, but will be **derived from other classes**. In this case, the class **inherits all characteristics of this class and adds other characteristics if desired**.

• Inheritances **can have multiple levels** - that means, a derived class can be the basis class for other classes. In this way, multi level hierarchies are created. The hierarchies represent the taxonomy and the structure of the modeled problem area.
**Problem:** in the M&S system we do not want the restriction of objects from masses (ideal springs). Our model shall be extendable by *new elements*, like damped springs or damper. We want to *reuse as much as possible of the existing code* in order to spare time and costs and reduce the probability of errors.

**Solution:** *through inheritance, we isolate the attributes and the behavior of that is common to several objects into an upper class.* Common features of springs and dumpers are:

1. Both have 2 connecting points (flanges).
2. Both are elastic elements: their length is modified by applying a force
   - Delimitation: there are also other elements with 2 connecting points, that are non-elastic, like the beam. Therefore, common feature 1 and common feature 2 should be placed on different hierarchical levels of the inheritance tree.
Inheritance (3)

3.1. Introduction and Basic Concepts
3.1.3 Attributes of object orientation

```plaintext
import TwoFlange
import Compliant
import Spring

TwoFlange

Flange a; Flange b;
end TwoFlange;

Compliant [extends TwoFlange]

Force f; Distance s;
equation
  s = b.s - a.s; 0 = a.f + b.f; f = b.f;
end Compliant;

Spring [extends Compliant]

parameter Real c; parameter Length l;
equation
  f = c * (s - l);
end Spring;
```
Inheritance (4)

- **TwoFlange** is the class that contains 2 flanges a and b. The class Flange can be considered as given.

- The class **Compliant** inherits from the class TwoFlange. This means, that a compliant part (Compliant) is an element with 2 connecting points (TwoFlange).

- The class **Compliant** defines the behavior of a compliant part by an equation, that defines the dependence between the force $f$ and the distance $s$ between the two flanges a and b.

- The class **Spring** inherits from the class Compliant. This means, that a Spring is a Compliant (compliant part). In addition, Spring defines the parameter specific to a spring: spring constant $c$ and spring length $l$. The equation describes the behavior of the spring.

```
partial model TwoFlange
    Flange a; Flange b;
end TwoFlange;

partial model Compliant extends TwoFlange;
    Force f; Distance s;
    equation
        s = b.s - a.s; 0 = a.f + b.f; f = b.f;
end Compliant;

model Spring extends Compliant;
    parameter Real c; parameter Length l;
    equation
        f = c * (s - l);
end Spring;
```
Inheritance relation

Source: [Booc91]
Properties of object orientation

- **Enclosure:**
  Data enclosure allows the separation of the internal implementation of objects to external access. Access is granted only via a predefined interface for making it independent of implementation details.

- **Instantiation:**
  Instantiation means the creation of an object of a specified class, that represents its (abstract) description. This instance of the class is a concrete occurrence of that class.

- **Inheritance:**
  Classes can be derived from other classes. Each class inherits the attributes and methods from the upper class.

- **Polymorphism:**
  The concept of polymorphism means, that properties or methods of a class can be referenced by objects, without knowing the concrete specification of the class inside those respective objects.
• **Polymorphism** directly translated means „many sidedness“ and designates first the ability of variables of type object of a class to gather together objects of different classes. This doesn’t happen uncontrolled, but rather is limited to object variables of type X or all objects in the class X or a further derived class.

• There are many forms of **polymorphism**:
  - Generic polymorphism
  - Parametric polymorphism
  - Inheritance polymorphism
Polymorphism (2)

- As for this lecture, we will limit ourselves to **inheritance polymorphism**
- A message can initiate different behavior (different operations), depending on which subclass of the upper class the object belongs to
Polymorphism (3)

- The upper class (vehicle) inherits its properties and methods:
  - The subclasses (bus, ship, plane) contain all the method `start()`, but they interpret it in different ways
- One demands from a vehicle to start, but each concrete vehicle has a different way of doing it.
Problem: in the M&S system we can model and simulate only an ideal spring, because we only have defined the class Spring. For precise studies, we need a model that allows us to represent without an extra effort the behavior of different types of springs, both ideal and real (damped) springs.

Solution: the inheritance hierarchy provides the possibility to consider common attributes and behavior of compliant elements. Both the damped spring and the ideal spring are compliant part, such that a class DampedSpring can be derived from the class Compliant. In the spring-mass-model, a Compliant-object is used instead of a Spring-object. This object of an upper class can contain both Spring and DampedSpring objects, that have different behavior according to polymorphism.
Polymorphism (5)

- Additionally to the class \textit{Spring}, that describes the behavior of an ideal spring, we define the class \textit{DampedSpring}.

- \textit{DampedSpring} is derived from \textit{Compliant}, too. This means, that a damped spring can receive and interpret the message that can be processed by an compliant element, for example „Compute length modification“

```plaintext
model DampedString extends Compliant;
  parameter Real c "spring constant";
  parameter Real d "damping";
  parameter Length l "length of unstretched spring";
  Velocity v "relative velocity between flanges b and a";

  equation
    v = der(s);
    f = c * (s - l) + d * v;
end DampedString;
```

3.1. Introduction and Basic Concepts

3.1.3 Attributes of object orientation
An ideal or damped spring can be introduced to the system as a compliant part object. The M&S system decides during runtime, if the method for the ideal spring or for the damped spring should be called.
3.1.3 Attributes of object orientation

Beispiel: Industrie-Roboter (DLR, Dynasim, KUKA)

1000 nichttriviale algebraische Gleichungen, 80 Zustände.
mit “Mixed-Mode Integration”:
schneller als Echtzeit auf 650 Mhz PC.
Beispiel: Hardware-in-the-Loop Simulation von Automatikgetrieben
(verschiedene KFZ Hersteller)

Steurgerät
(Hardware)

gewünschter Druck
Steering element

+ Fahrer + Motor + Wandler +
1D Fahrzeugdynamik

Courtesy ZF Friedrichshafen
Application example (3): large, detailed, vehicle model

- 60 joints, 70 bodies
- 25000 non-trivial algebraic equations
- 320 states

Motor (Combustion)

Drive chain (automatic drive)

3D-Mechanik

Hydraulik


[HiKa99] M. Hitz, G. Kappel: „UML@Work“, dpunkt.verlag, 1999

