Programming Language Constructs as Basis for Software Architectures

UNIVERSITÄT SALZBURG

From individual parts to components

In the 50s:

- Machine/Assembler programs: bound to specific hardware

In the 60s-70s:

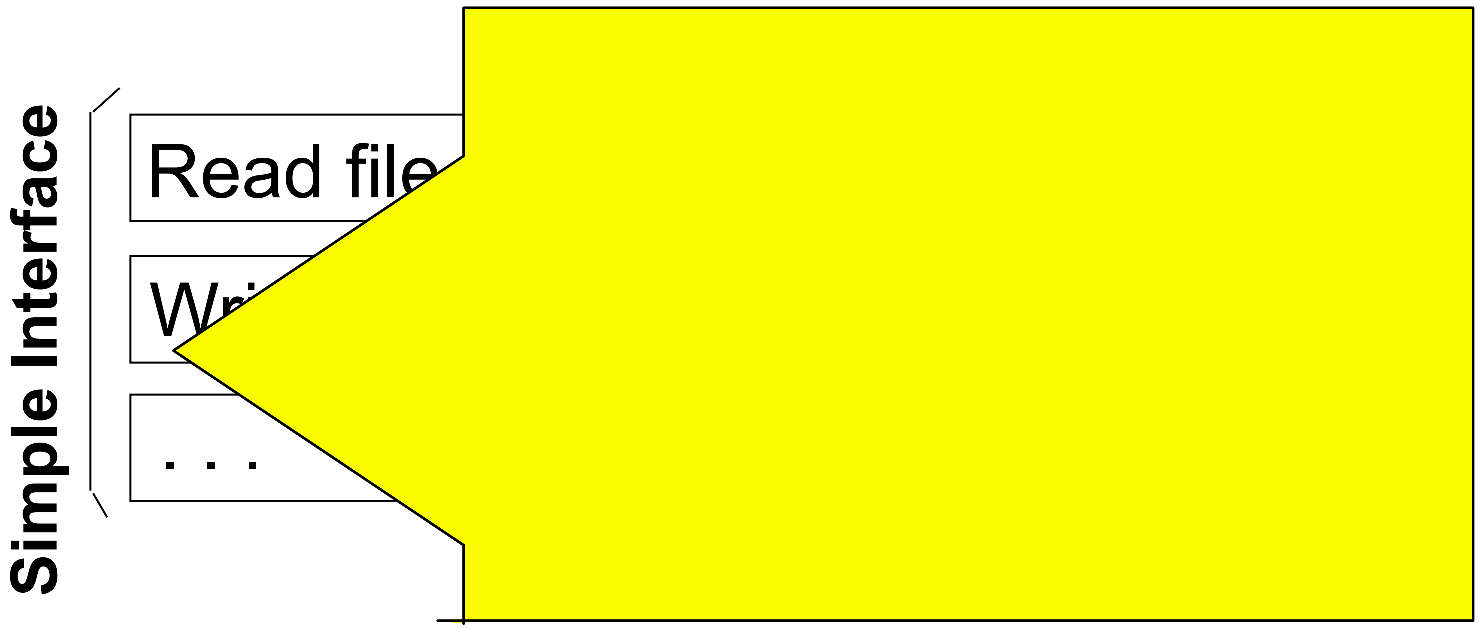
- Higher programming languages (such as Pascal, C)
- Instructions can be combined into functions/procedures
  - Individual parts

In the 80s and 90s:

- Functions/procedures are combined into Modules (Modula, Oberon, C++, Java, C#)
  - Software components

UNIVERSITÄT SALZBURG

Example: A File handler component



Hidden implementation details:

- Access to hard disc
- Splitting up file contents, etc.

Architecture-Patterns

↓

Software-Patterns

UNIVERSITÄT SALZBURG

The Timeless Way of Building

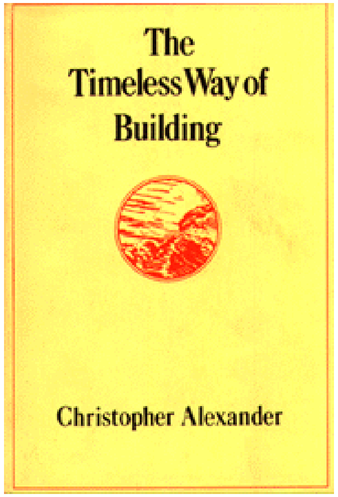
Christopher Alexander, Professor of Architecture, Univ. of California, Berkeley:

Books published in 1979:

The Timeless Way of Building

A Pattern Language (253 Patterns)

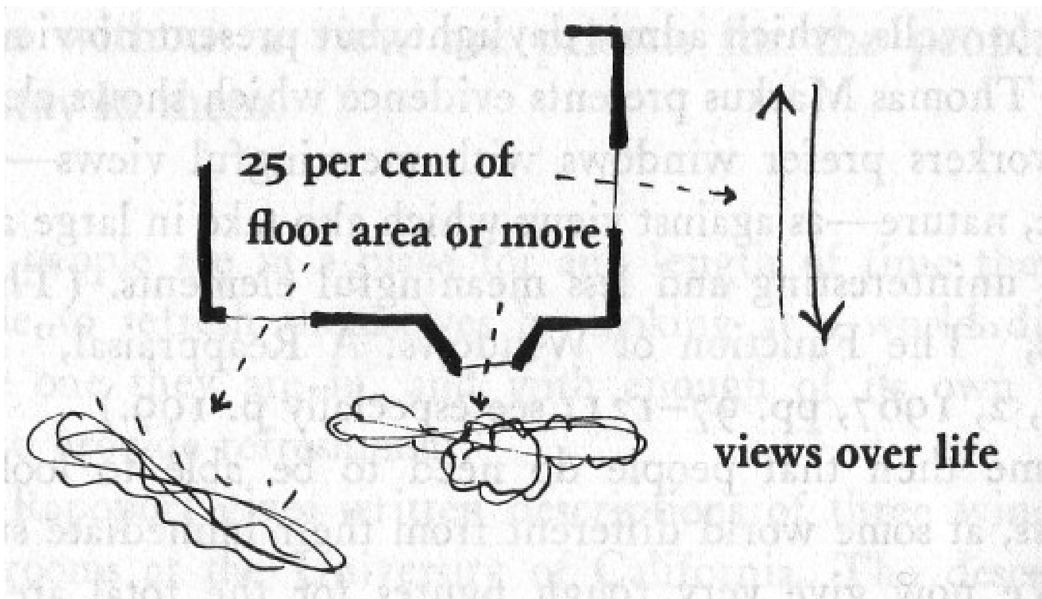
Quality without a name



Discovered by the Software community in 1991

UNIVERSITÄT SALZBURG

Example: Windows Overlooking Life



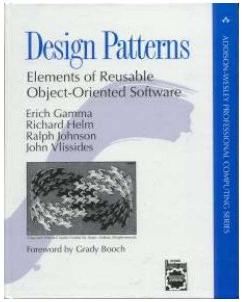
Examples of Software Patterns

UNIVERSITÄT SALZBURG

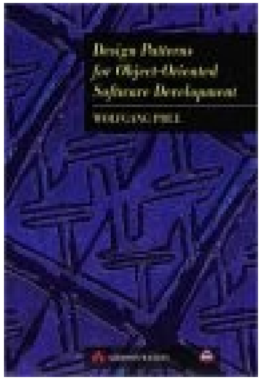
How can software PlugIn-Architectures be created?

Described in Architecture manuals (1995):

- E. Gamma, R. Helm, R. Johnson, J. Vlissides:
  Design Patterns: Elements of Reusable Software



- W. Pree:
  Design Patterns for Object-Oriented Software Development



UNIVERSITÄT SALZBURG

What are PlugIn-Architectures?

- Modern cooking machine: plugging in various tools makes it a mixer, a mincer, a blender

- New automobile models resemble older ones in their core: chassis, transmission, engine pallet.

UNIVERSITÄT SALZBURG

Software Examples

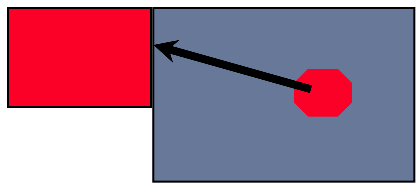
- Dedicated software:
  - Hotel reservation system
  - Car rental system
  - Ski rental system
  - Motorcycle rental system
  - etc.
- PlugIn-Architecture:
  - Reservation system
    (of rental property)

UNIVERSITÄT SALZBURG

Dedicated Software

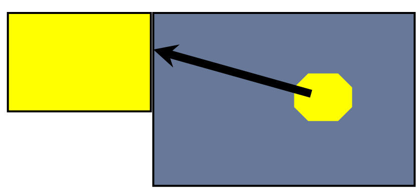
Dependence between components is hard-coded

Hotel room



Coupling with another component requires changes

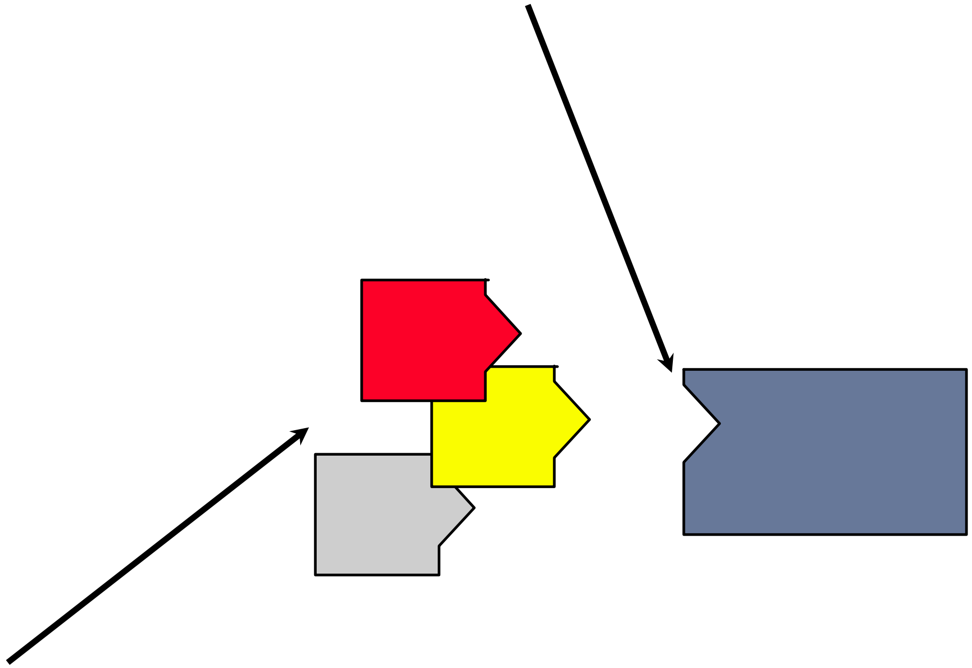
Car



UNIVERSITÄT SALZBURG

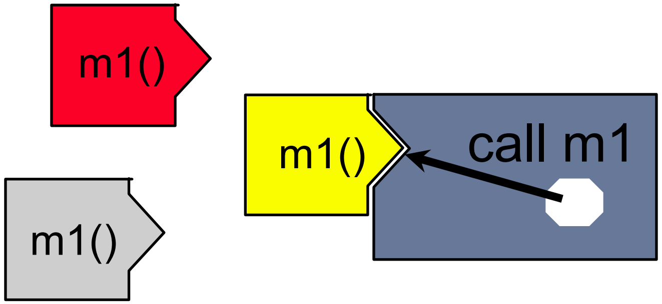
Pattern: PlugIn-Architectures require the definition of „Plugs“

Plug „Rental property“



Plug-compatible
Components

The so-called dynamic binding of calls makes changes in the source code unnecessary



The Rental property „Plug“

Defines general, abstract characteristics:

- isFree(Period)
- reserve(Period)
- estimatedPric
- etc.

Software Techniques – Quo vadis?

UNIVERSITÄT SALZBURG

- Cost intensive maintenance of software, which is 20 - 30 years old
- Engineering approaches will be established at least in sub - domains such as safety critical systems

UNIVERSITÄT SALZBURG

- The simple, mechanical view is hardly scalable
- Biological systems model → Internet growth by a factor of 100 million
- Development process:
  - Analisys → Design → Implementation

Basics of Object-Oriented Modeling

Analysis and Design with UML

Software Engineering
Winter 2011/2012

Dr. Stefan Resmerita

UNIVERSITÄT SALZBURG

Tools for OO Analysis and Design

UNIVERSITÄT SALZBURG

OO expectations

- Improved modularity
- Improved reusability
  - Potential for reusable software architectures

    (= generic complex components) has not been fully investigated so far

- Support for OO modeling is important

What can be expected from OOAD Tools (I)

Great designs come from great designers, not from great tools.

Tools help bad designers create ghastly designs much more quickly.

Grady Booch (1994)

What can be expected from OOAD Tools (II)

- OOAD tools can perform:
  - Providing and editing diagrams based on various OO notations
  - Checking of consistency and constraints
    - Does an object have the called method?
    - Are the invariants (e.g. single instance, etc.) satisfied?
    - ...
  - Completness evaluation
    - Are all the Methods/Classes used?
    - ...

UNIVERSITÄT SALZBURG

Conventional (SA/SD) versus OO tools (I)

The main differences regard two aspects:

- (1) Software Architecture
  - Conventional tools are based on a separation between data and functions
  - OO tools are based on the grouping of data and functions into meaningful „closed“ objects

UNIVERSITÄT SALZBURG

Conventional (SA/SD) versus OO tools (II)

- (2) Semantic possibilities

  Relationships in the conventional ER
  - One-to-one (1:1) – has_a, is_a
  - One-to-many (1:m) – owns, contains, is_contained_in
  - Many-to-many (m:n) – consists_of

UNIVERSITÄT SALZBURG

Conventional (SA/SD) versus OO tools (III)

OO modeling has more comprehensive means of expression

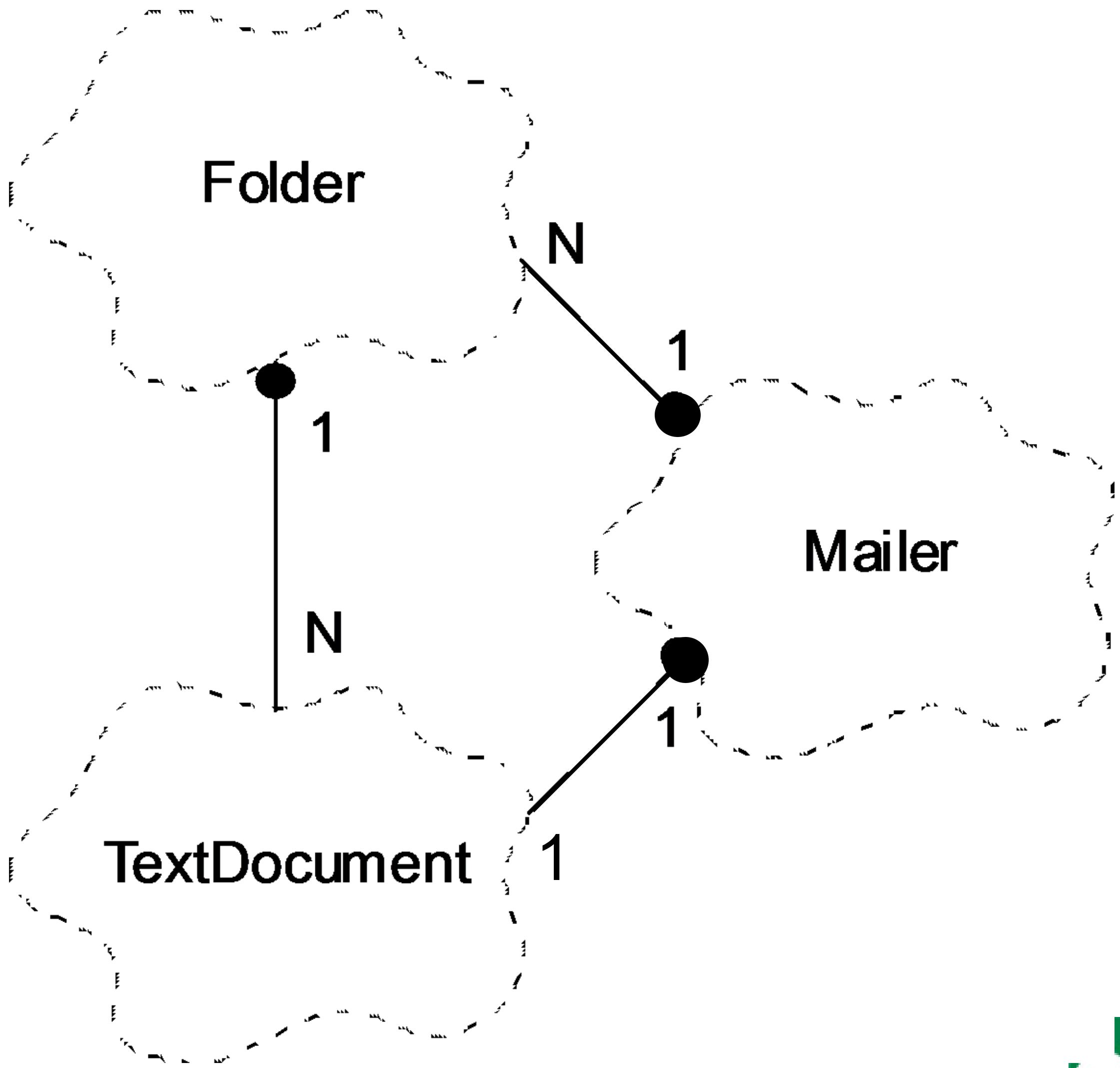
- Class/Object relations and dependencies
  - Inheritance
  - Association
  - Has_a (by value, by reference)
  - Uses_a (by value, by reference)
- Class attributes
  - Is_abstract, is_metaclass
  - Is_parameterized
- Access rights

OO Techniques at the beginning of the 90s

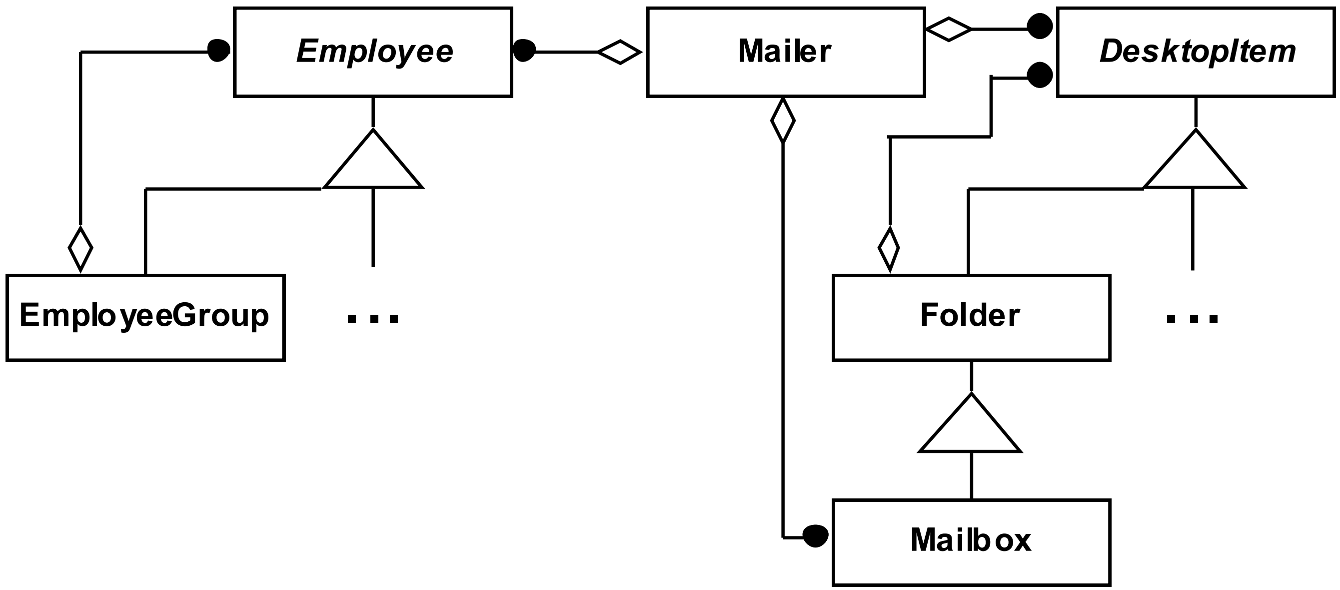
- OOD / Rational Rose
  Grady Booch
- Object Modeling Technique (OMT)
  James Rumbaugh et al.
- OO Software Engineering
  Ivar Jacobson et al.
- OO Analysis (OOA)
  Peter Coad und Ed. Yourdon
- Responsibility-Driven Design (RDD)
  Rebecca Wirfs-Brock et al.
- OO System Analysis (OOSA)
  Sally Shlaer and Steve Mellor
- . . .

UNIVERSITÄT SALZBURG

Example for Booch notation



Example of OMT notation



Common features of OOAD methods (I)

- They aim to represent the physical world without artificial transformations as a software system
  - Application of the same concepts in all phases of software development
  - The border between Analysis and Design becomes more blurred

- Moreover, very vague usage guidelines are indicated

UNIVERSITÄT SALZBURG

Common features of OOAD methods (II)

- OOAD methods permit the modeling of the following aspects of a system:
  - Static aspects
    - The Class/Object model stands in the foreground
    - Higher abstraction levels are represented by Subsystems
  - Dynamic aspects
    - Interaction diagram
    - State diagram
    - Use case diagram

UNIVERSITÄT SALZBURG

Differences between OOAD methods

- The differences between the methods lie mostly in the notation
- The notations are to a large extent language independent

  => Standardization is obvious

All of the OO methodologies have much in common and should be contrasted more with non-OO methodologies than with each other.

James Rumbaugh

(1991)

UNIVERSITÄT SALZBURG

UML influences

- The Unified Modeling Language contains various aspects and notations from different methods
  - Booch
    - Harel (State Charts)
  - Rumbaugh (Notation)
  - Jacobson (Use Cases)
  - Wirfs-Brock (Responsibilities)
  - Shlaer-Mellor (Object Life Cycles)
  - Meyer (Pre- und Post-Conditions)

UNIVERSITÄT SALZBURG

The UML standard

- The first draft (version 0.8) was published in 1995
- Various adjustments and the inclusion of Ivar Jacobson led to version 0.9 in 1996
- Version 1.0 (an then 1.1) was submitted to the Object Management Group (OMG) in 1997 as basis for standardisation
- Version 1.3 came out in 1999
- Version 1.4.2 became an international standard in 2005
- Current OMG standard: version 2.3
- Version 2.4 available as beta2

UNIVERSITÄT SALZBURG

The Unified Modeling Language (I)

What is the UML?

- Language
  - Communication
  - Exchange of ideas
- Graphical modeling language
  - Drawings, words and rules for representing aspects of systems

UNIVERSITÄT SALZBURG

The Unified Modeling Language (II)

What is UML not?

- No method
  - Specifies how models are made but not which and when
  - This is a task of the software development process

Method = Process + Modeling Language

UNIVERSITÄT SALZBURG

The Unified Modeling Language (III)

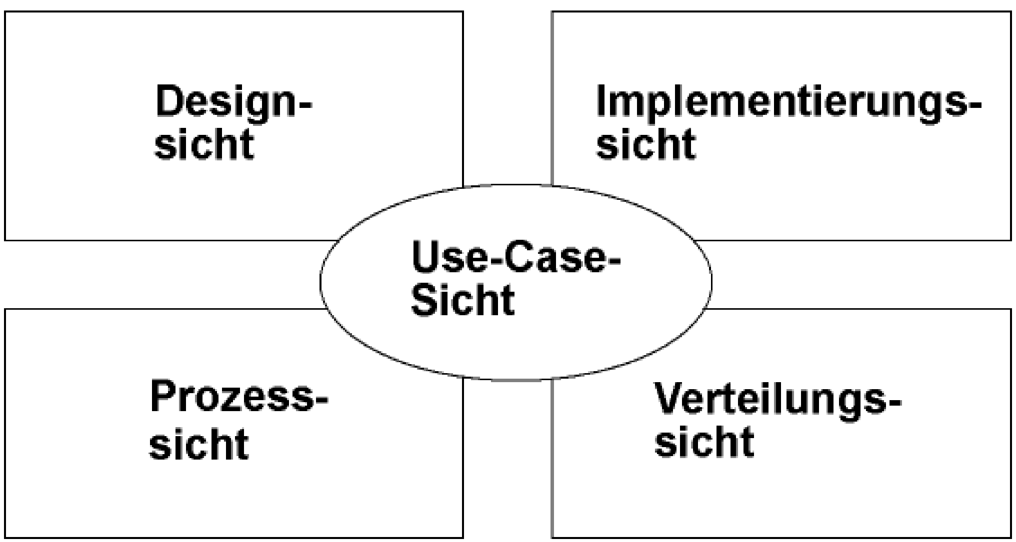
Why is UML needed?

- Model visualization
- Model specification
- Model checking
- System construction
  - Forward and reverse engineering
- System documentation

UNIVERSITÄT SALZBURG

The Unified Modeling Language (IV)

- Models
  - Projections of systems on certain aspects
  - Used for understanding systems



UNIVERSITÄT SALZBURG

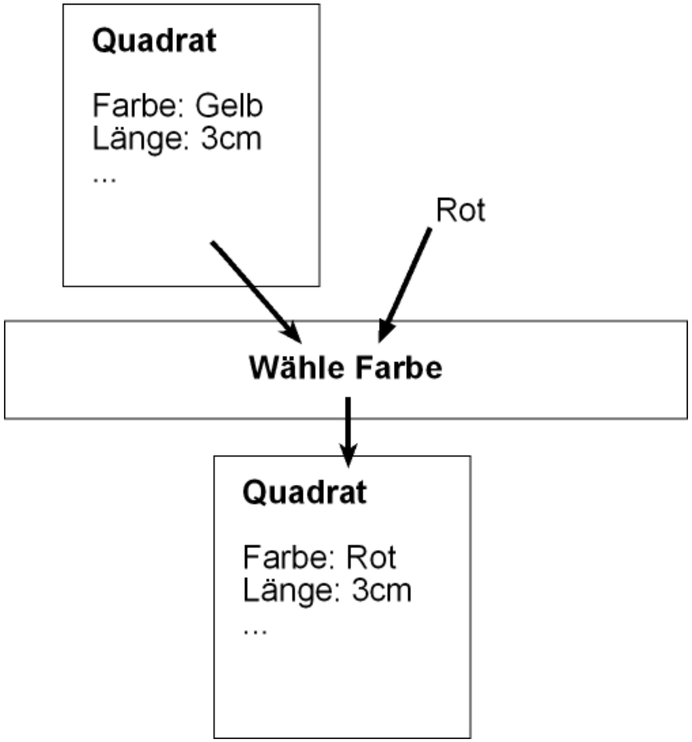
OO concepts
UML representation

- Objects, Classes, Messages/Methods
- Inheritance, Polymorphism, Dynamic Binding
- Abstract Classes, Abstract Coupling

UNIVERSITÄT SALZBURG

OO versus Procedural (I)

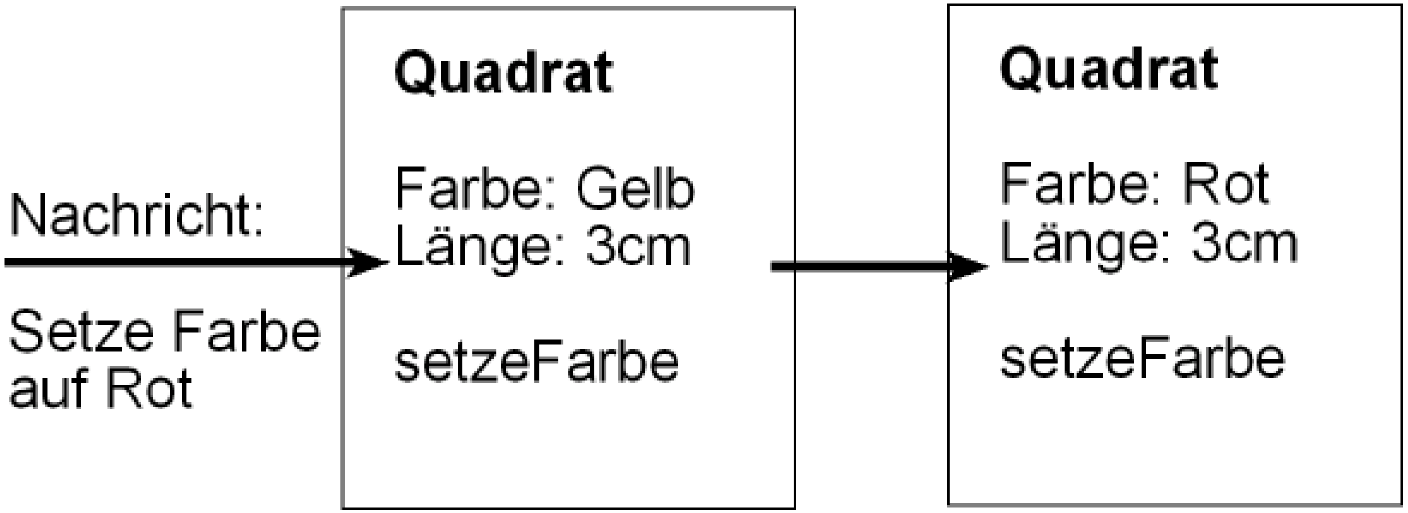
- Procedural: Separation between data and procedures



UNIVERSITÄT SALZBURG

OO versus Procedural (II)

- Object-oriented: Data and procedures form a logical unit → an Object



Objects(I)

An object is a representation of

- A physical entity
  - E.g. Person, Car, etc.
- A logical entity
  - E.g. Chemical process, mathematical formula, etc.

UNIVERSITÄT SALZBURG

Objects (II)

The main characteristics of an object are:

Its identity

Its state

Its behavior

UNIVERSITÄT SALZBURG

Objects (III)

- State

  The state of an object consists of its static attributes and their dynamic values

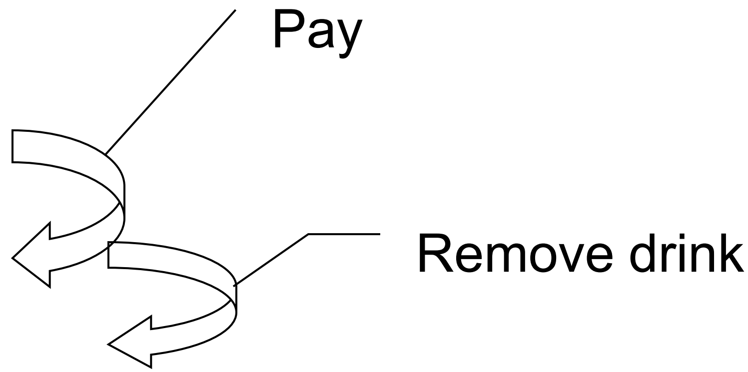
  - Values can be primitive: int, double, boolean

  - Values can be references to other objects, or they can be other objects

UNIVERSITÄT SALZBURG

Objects(IV)

- Example
  - Drinks machine
    1) Ready
    2) Busy
    3) Ready



  - Attributes – values
    - Paid: boolean
    - Cans: number of cans

Objects(V)

- The behavior of an object is specified by its **methods** (=operations)

- In principle, methods are conceptually equivalent to procedures/functions:

  Methods = Name + Parameters + Return values

Objects(VI)

- Example
  - Rectangle
    - Name of the operation: `setColor`
    - Parameter: name of the color (e.g. `Red`)
    - Return values: none
- Calling an operation of an object is reffered to as sending a message to the object

UNIVERSITÄT SALZBURG

Objects(VII)

- Identity

  The identity of an object is the characteristic that differentiates the object from all the other objects

- Two objects can be different even if their attributes, values and methods coincide

UNIVERSITÄT SALZBURG

Object – Orientation

- Classification
  - Object grouping
- Polymorphism
  - Static and dynamic types
  - Dynamic binding
- Inheritance
  - Type hierarchy

UNIVERSITÄT SALZBURG

Classification

- Class

  A class represents a set of objects that have the same structure and the same behavior

  A class is a template from which objects can be generated

UNIVERSITÄT SALZBURG

Classification Example

- Class Person
  - Attributes:
    - Name: String
    - Age: int
  - Operations:
    - eat, sleep, ...

- Object of type Person: Oliver
  - Attributes:
    - Name: Oliver
    - Age: 24

Class as a template/type (I)

- Comparison with C

```
struct{
        int day, month, year;
   } date;
   date d1, d2;
```

⇒ All are accessible

⇒ There is no method

UNIVERSITÄT SALZBURG

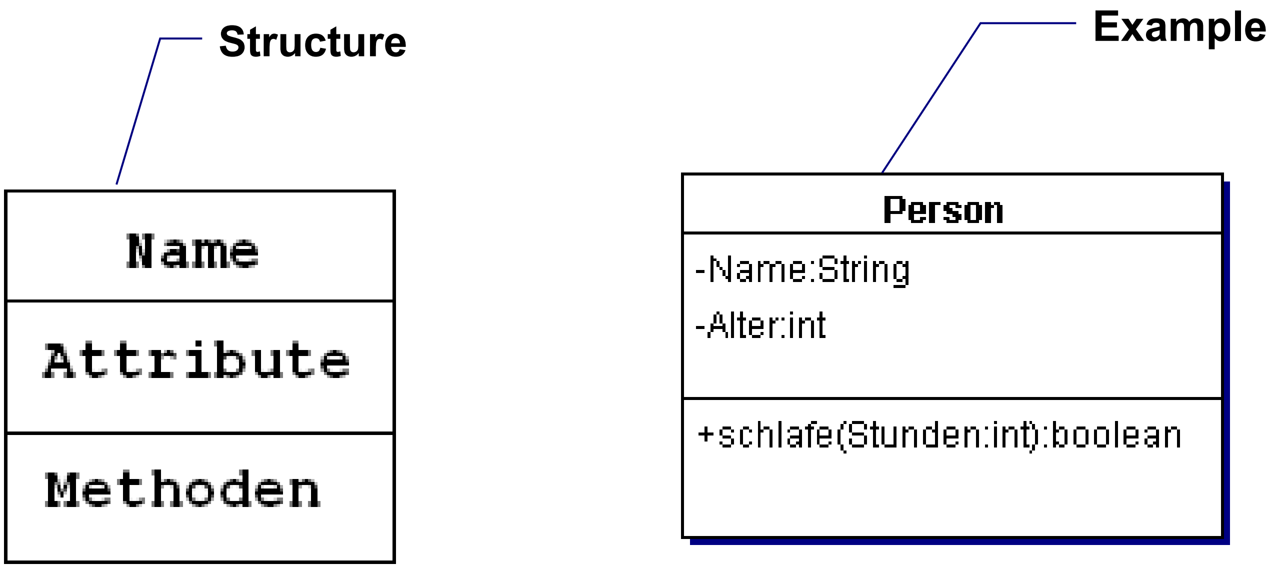
Class as a template/type (II)

A class indicates which type an object has, i.e., which messages understands and which attributes it has.

- A class consists of
  - A unique name
  - Attributes and their types
  - Methods/Operations

Classes in UML (I)

- UML notation for a class:



UNIVERSITÄT SALZBURG

Classes in UML (II)

Notation for attributes:

A	only the attribute name
: C	only the attribute class
A : C	attribute name and class
A : C = D	attribute default value

timeWhenStarted	→ A
: Date	→ : C
timeWhenStarted : Date	→ A : C
timeWhenStarted : Date = 1.1.1999	→ A : C = D
timeWhenStarted = 1.1.1999	→ A = D

UNIVERSITÄT SALZBURG

Classes in UML (III)

Notation for Methods/Operations:

m()	only the method name
m(arguments): R	method name, arguments type of returning parameter

Example:

printInvoice()	→ m()
printInvoice(itemNo: int): bool	→m(arguments): R

UNIVERSITÄT SALZBURG

Classes in UML (IV)

- Adornments (decorations) : additional graphical elements (represented by triangles in the Booch method)

- Methods and attributes have attached graphic symbols to express access rights: **`public, private, protected`**

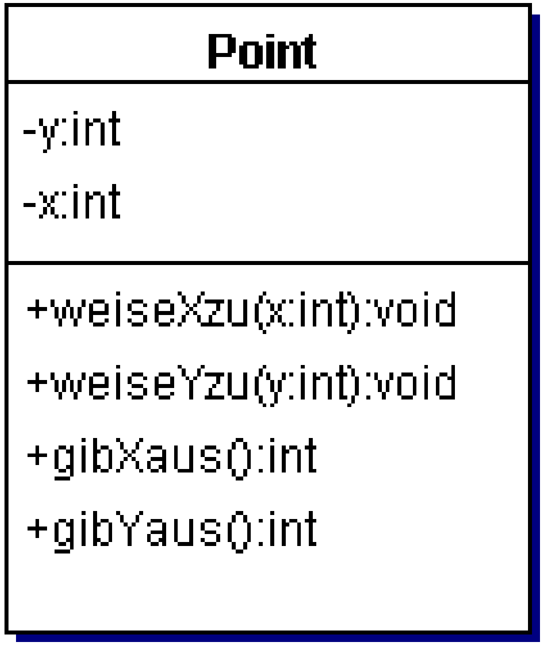
  Example:

  `+sleep(Hours:int)`

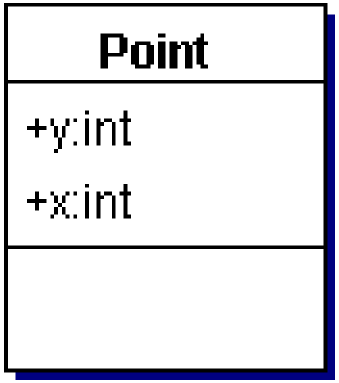
- Standalone adornment: Note

UNIVERSITÄT SALZBURG

Example: access rights



Unnecessary complexity, since there is no dependency between x and y



Better alternative

UNIVERSITÄT SALZBURG

Classes in Java

```
public class Person{

  String name;
  int age;
  ...

  public int getAge(){
    return age;
  }
  public void setAge(int theAge){
    age = theAge;
  }
}
```

Class name

Attributes

Operations

Using classes in Java

- Classes are used in Java to specify the type of variables and to instantiate objects
- Keyword: new
- Example:

```
Person manager = new Person("Martin");
```

Declaration of variable „manager“

Instantiation of an object of class Person with name Martin

UNIVERSITÄT SALZBURG

Example: Hotel reservation

- What can be modeled as classes in a hotel reservation system?
- What attributes will the classes have?
- What operations?
- Which instances (objects) of these classes will there be?
- What sorts of relations will take place between the objects/classes?

UNIVERSITÄT SALZBURG