Object-Oriented
Software Development

Principles

Organizational Issues

- Info
  - http://people.canoo.com/schaeffer/vl13/oo_se13.html
    - Handouts
    - Hints, infos and links
- Exam
  - 8.1.2014, 14:00 - 15:45
  - **Deadline for sign-up/-off**: 11.10.2013 24:00
  - No retake!
- Literature: will be published for each chapter
- Break?
- Feedback? yes please - instant feedback preferred or by email

Hintergrund

Contents

Introduction and Principles – Contents

- Motivation
  - Software developement challenges
- History
  - From Simula to Java
- Terms and definitions
  - Objects
  - Classes
  - Inheritance
  - Type compatibility
  - Dynamic binding

Challenges

- People & Skills Management
- Complexity & Volume
- Time to Market
- Software Architecture
- Software Quality
- Configuration Management
- System Integration
- Longevity of IT Systems
- Globalization
- New Channels

Complexity

- Complex information models

- Complex system landscapes
  - Mix of "Make" and "Buy"
  - Integration of external system landscapes
    - Frequently caused by merging of companies

- Architecture and design are key to success

- Feasibility
  - Complex projects are likely to fail
  - Projects running late might not be relevant anymore

- Evolvable systems
  - Only systems which don’t have to be perfect and comprehensive with the first release are likely to succeed.
  - Incremental-Iterative Development

Productivity and Quality

- How to improve productivity and quality?
  - Better (rather than more) developers
  - More sophisticated tools, methods, and processes
  - Raise the level of abstraction

- Abstraction is key
  - High-level programming language (z.B. Java) rather than assembler
  - Class libraries and frameworks
  - Model-driven environments (MDA)
  - Domain specific languages

- Reuse
  - (Semifinished) reusable components
  - Class libraries and frameworks
  - What are the prerequisites for reuse?

History – The Beginnings of OO

- 1972: Modular Programming
  - D. L. Parnas: "On the Criteria to be Used in Decomposing Systems into Modules"
  - System structure
  - Comunication among subsystems

- 1983: Abstract Data Types
  - A.V. Aho, J.E. Hopcroft, and J.D. Ullman: "Data Structures and Algorithms"
  - Equality of data structures and algorithms
  - Micro structure

- OO concepts were advanced in parallel
  - Origin: in 1967 Simula was released
  - Introduced concepts such as modularization und abstract data types
  - Remained a niche language

Simula 67

- Simulation of complex environments
  - Building models of the reality

- Simula 67
  - Developed by Ole-Johan Dahl und Kristen Nygaard at the Norwegian Computing Centre, Oslo
  - Based on Simula I
  - General purpose programming language

  - Provides all relevant concepts of object-oriented programming
  - Small user community (prob. ahead of the times)

  - Impact on:
    - Smalltalk
    - C++
    - VLSI Design

Simula 67

Ole-Johan Dahl & Christen Nygaard

Smalltalk

- Xerox Parc
  - Innovations:
    - Workstation (Hardware, Bitmap Display)
    - Laser Printing
    - Ethernet
    - Programming languages (Smalltalk, Mesa)
    - Interpress (precursor to Postscript)
  - Nursery of object-oriented programming

- Research project: effective computer usage
  - Time frame: 1971 - 1983
  - Alan Kay, Dan Ingalls, Adele Goldberg

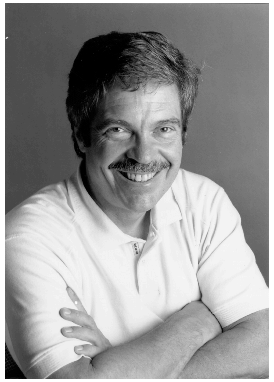
- Results
  - Concepts & programming language (Smalltalk)
  - Development environment & class libraries
  - Graphical user interface

Smalltalk

Byte, Aug. 1981

Adele Goldberg

Dan Ingalls



Alan Kay

C++ and Java

- C++
  - Conceived by Bjarne Stroustrup at the AT&T Labs (1983)
  - Based on C and influenced by Simula 67
  - Most popular oo language prior to Java

- Java
  - James Gosling led the development of Java for the "Green" project (1995)
  - Based on C and influenced by Objective-C (Smalltalk)

Breakthrough of OO

- 1980-ies: oo proliferated in the Academia
  - 1986 first conference dedicated to oo (OOPSLA)
  - Early adopters in the software development community

- 1990-ies: gaining popularity in professional software development
  - Fueled by Java and the Internet hype

- > 2000
  - Widespread usage in strategic enterprise domains
    - Started on the presentation layer (Web)
    - Popular environment for enterprise backends (Java EE)
  - Further acceptance with C# and .Net

Object-Oriented Principles

- Must have
  - Objects
  - Inheritance
  - Dynamic binding

- Nice to have
  - Classes
  - Garbage collection
  - Persistency

Objects

- Object is an instance of an abstract data type:
  - Structural properties
    - State (well encapsulated)
  - Functional properties
    - Object reacts to external requests
- Objects must be uniquely identifiable
- Objects are communicating via messages
  - Receiving object executes corresponding method
  - Contrasting with procedural programming:
    - Object more important than procedure / method
    - Method to be executed is not explicitly defined
  - Asynchronicity is not implied (by default)

Classification and Specification

- What is more important – values or types?
  - Objects are values (more or less ➜ advanced principles)

- Philosophical question
  - Plato:
    - Abstractions (ideally prototypes) are more important than concrete objects
    - Concrete objects are imperfect instances of an abstraction
  - Kant:
    - Categories (types) take precedence
    - Categories filter resp. classify the reality and interpret phenomena

- Typed universes are easer to handle
  - Abstraction reduces complexity

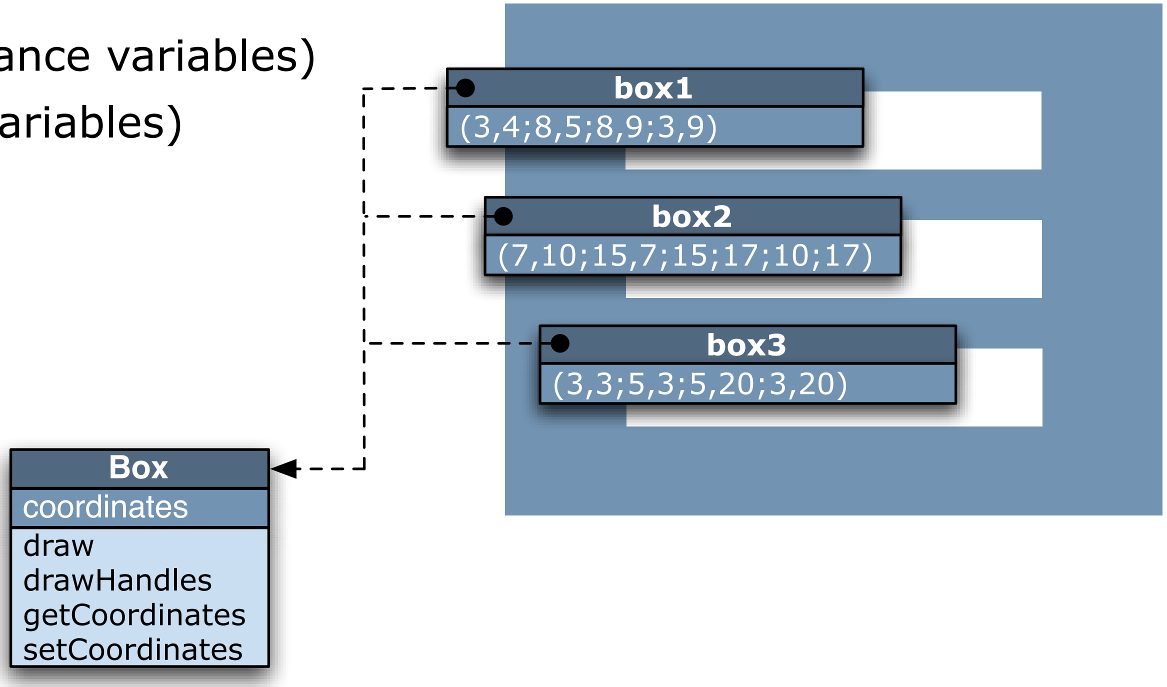
- Classification and specification require types

Classes

- Abstract description of an object
  - „Mold“ for creating objects
  - Class is an implementation of an abstract data type

- Structural properties
  - Individual state (instance variables)
  - Shared state (class variables)

- Functional properties
  - Instance methods
  - Class methods



Commonalities and Differences

- Classes (types) define common properties
  - Class as a language construct is not really necessary
  - Prototypes and delegation are sufficient

- How to capture commonalities and differences?

- Biological classification is based on a hierarchical taxonomy
  - Specification is refined along the type hierarchy

- Type hierarchy
  - Specification of commonalities and differences
  - Modelling relations among types

Inheritance

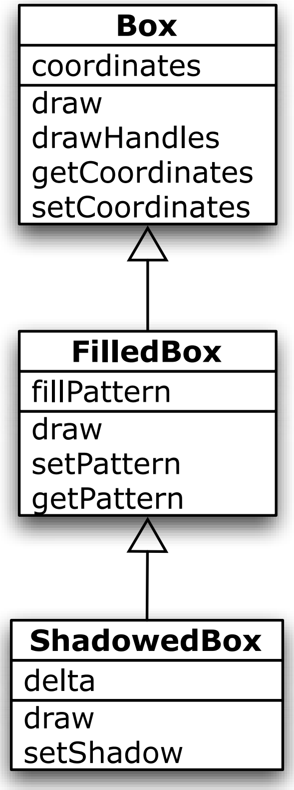
- Reuse requires flexible adaptation to new contexts
  - Components as such are hardly reusable
  - Copy and adapt leads to unmaintainable software
- Ideally: fine granularity in adaptations
- „Programming by Difference“
- Define new types based on existing ones **without copying**
- Procedure
  - Virtual copy of the base class (= superclass)
  - Adding individual state / functions (instance variables / methods)
  - Adding shared state / functions (class variables / methods)
  - Adapting (overriding) instance methods

Type compatibility

- Compatibility of objects
  - When can an object be exchanged for some other object?
  - Structural equivalence
    - Structure (subclass) ⊇ structure (superclass)
  - Functional equivalence
    - Functionality (subclass) ⊇ functionality (superclass)
  - Only publicly visible properties are relevant

- Example

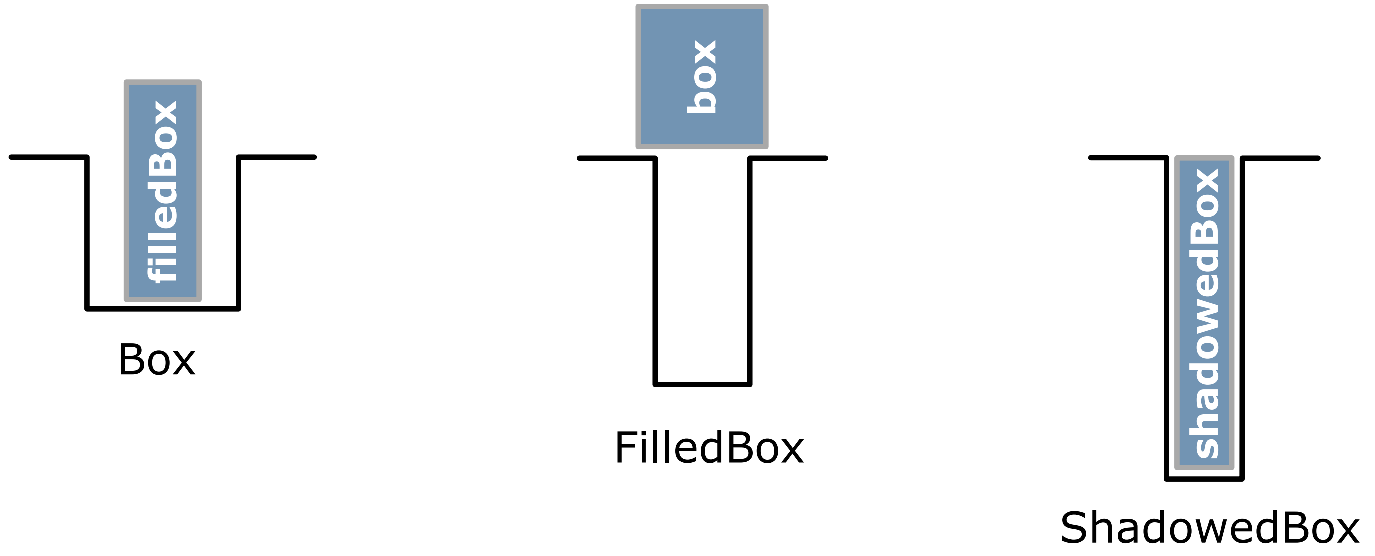
```
ShadowedBox s1, s2;
Box b;
FilledBox: f;
…
s1  = s2;      //kompatibel
b   = s1;      //kompatibel
f   = s2;      //kompatibel
s1  = b;       //nicht kompatibel!!!
```



Type compatibility

Broad (imprecise) specification
“Shallow Knowledge”

Narrow (precise) specification
“Profound Knowledge”



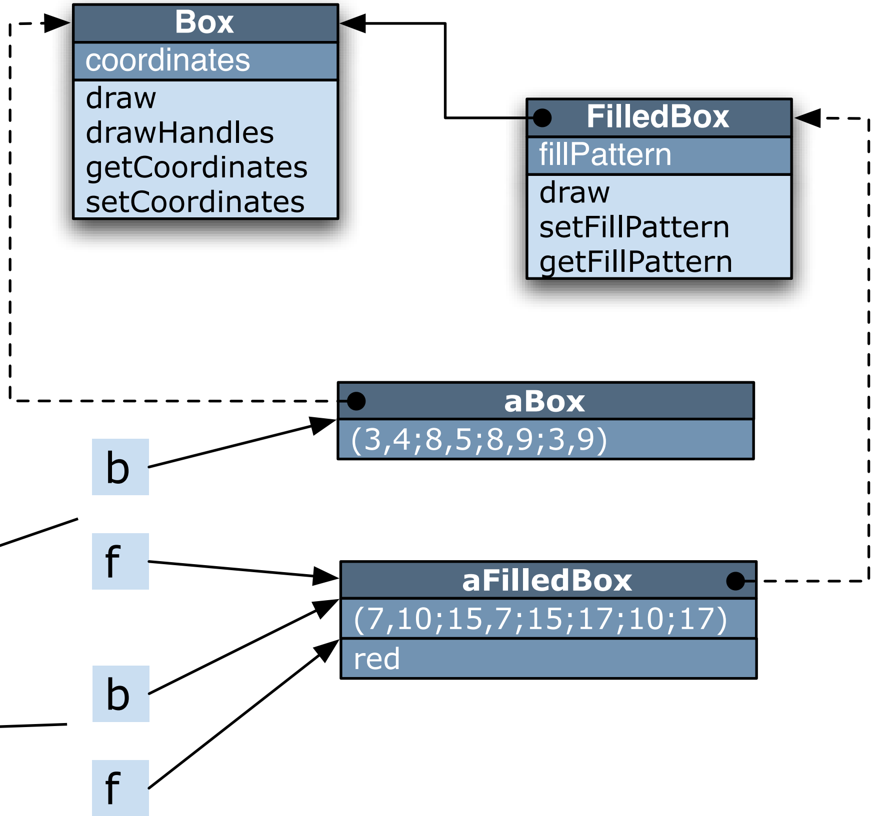
Broad (imprecise) specification
“Shallow Knowledge”

Narrow (precise) specification
“Profound Knowledge”

- Static vs. dynamic type
  - Static type: defined by declaration
  - Dynamic type: defined by the type of a referenced object

```
//Assumption: class hierarchy slide 24

Box b = new Box();
FilledBox f = new FilledBox();
…
b.draw();
b = f;
b.draw();
```



Dynamic Binding

- Purpose:
  - Along a type hierarchy several implementation of a method may exist
  - Look up the right method according to the dynamic type
  - Dynamic binding is a kind of polymorphism
- Abstraction
  - Implementation can be based mostly on (abstract) base classes
- Factorization
  - Knowledge about derived classes is restricted to a few places
- Substitutability
  - Objects of a base class can be replaced by objects of a derived class
  - **Principle of Substitutability**

Dynamic Binding - Implications

- Shifting control flow to objects
- Shifting type information to objects
- Factorization
  - Avoiding (duplicated) case cascades
- Performance impact at run-time
  - 5-15% overhead compared to static binding
- Optimization by explizit declaration, compiler or linker
  - Explicit declaration of dynamic binding in the base class (as in C++ or C#) requires farsightedness
- Tools are necessary for reading and comprehending software

Garbage Collection

- Challenge
  - Large number of objects is created
  - Most of them are released shortly (e.g. local variables)
  - Explicit release is error-prone! (global view required)
- Solution
  - Run-time support for releasing memory
  - Security and efficiency are negatively correlated
- Advantage
  - Enhanced reliability
  - Less implementation, test and maintainance effort
- Disadvantage
  - Run-time and memory impact
  - Limited suitability for realtime systems
  - Integrating systems without garbage collection?

Persistency

- Object can (should) survive its dynamic environment

- Storing on non-volatile memory

- Challenges:
  - Treating references while storing / reading objects
  - Version control
  - Treatment of unknown object (resp. types)

- Solutions:
  - Object serialization
  - Object-relational mapping
  - Object-oriented databases

OO Concepts in Java

- Object
  - Instance of a class

```
Box b = new Box();
```

- Class
  - Instance variables
  - Class variables
  - Instance methods
  - Class methods

```
public class Box {
    Rectangle fBounds;
    static Rectangle sMinSize();

    public void draw(Graphics g) {…}
    public static void setMinSize(Rectangle r) {…}
}
```

- Inheritance

```
public class FilledBox extends Box {…}
```

- Dynamic binding
  - All instance methods are dynamically bound
- Persistency
  - Interface "Serializable", JPA (Java Persistence API)

Glossary

- Static vs. dynamic type
  - Static type:
    Defined by declaration
  - Dynamic type:
    Defined by the type of a referenced object

- Factorization
  - A fact is defined only once in a software system
  - DRY: **D**on’t **R**epeat **Y**ourself

- Principle of Substitutability
  - An object of class T can always be substituted by an object of class T’ where T’ is derived from T

Literature and References

- Literature:
  - B. Eckel: Thinking in Java
  - T. Budd: Understanding Object-Oriented Programming with Java
  - B. Meyer: Object-Oriented Software Construction
  - D. Flanagan: Java in a Nutshell

- Picture credits
  - O. Dahl & K. Nygard: http://www.ifi.uio.no/~kristen/FORSKNINGSDOK_MAPPE/F_NR_1960.html
  - Byte 8-81: http://www.math.rsu.ru/smalltalk/images/byte.gif
  - Adele Goldberg, Dan Ingalls: http://www.smalltalk.org/
  - Alan Kay: http://www.aes.org/technical/images/Alan_Kay_Photo.jpg
  - B. Stroustrup: http://www.research.att.com/~bs/homepage.html
  - J. Gosling: http://www.apple.com/pro/science/gosling/