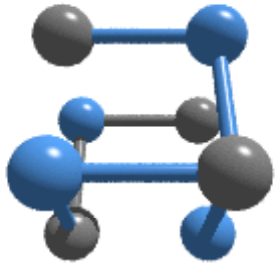


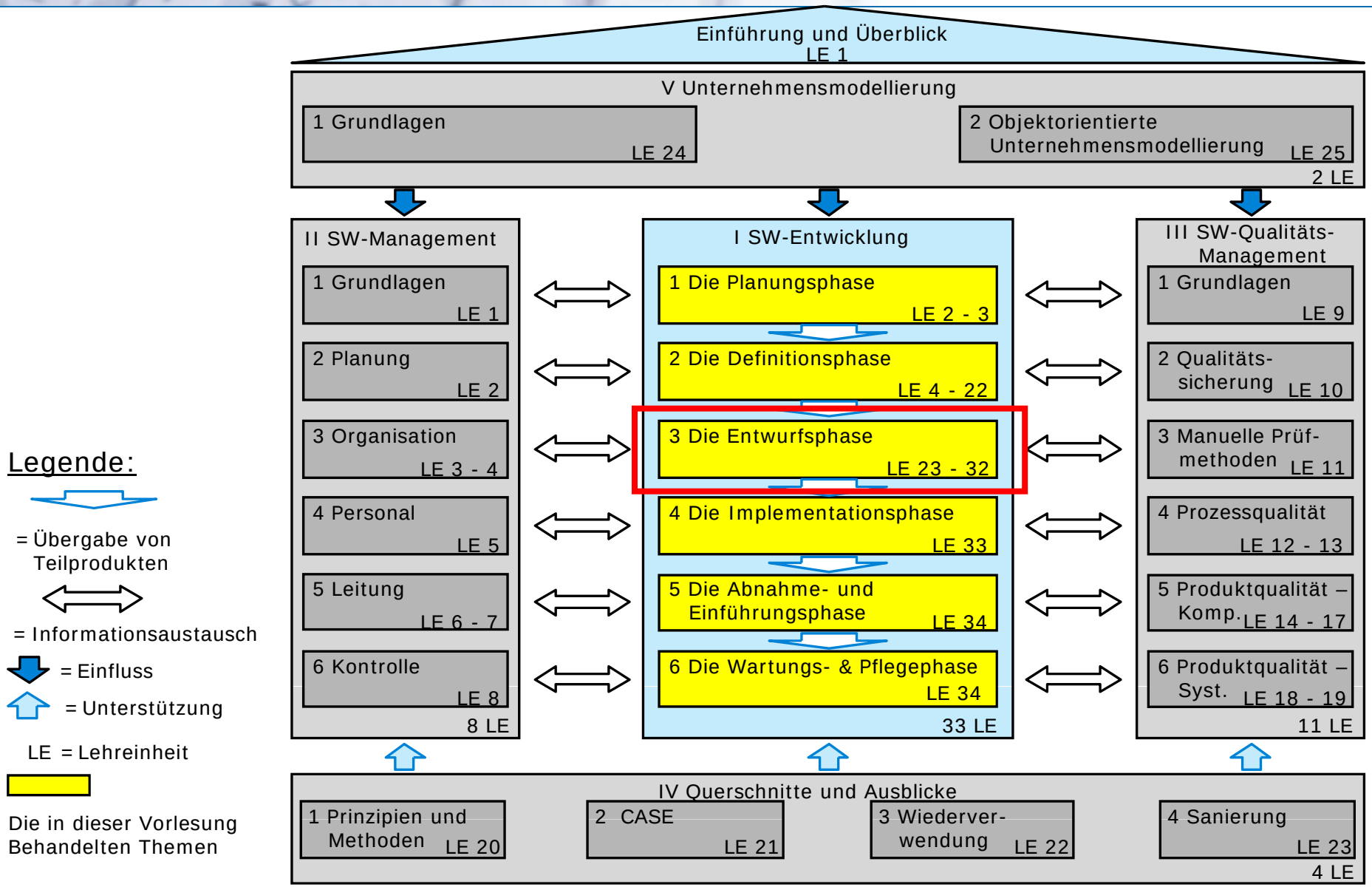
Vorlesung Softwaretechnik
- OO-Design, Komponenten -

Prof. Dr.-Ing. habil. Klaus-Peter Fähnrich

Wintersemester 2009/2010

UNIVERSITÄT LEIPZIG

Institut für Informatik
Betriebliche Informationssysteme



Überblick LE 27

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

LE 27: Objektorientiertes Design

- Konzepte
  - Objekt/Klasse
  - Attribut
  - Operation
  - Assoziation
  - Polymorphismus
  - Vererbung
  - Paket
  - Szenario
  - Zustandsautomat
- Klassenbibliotheken und ihre Architektur
- Entwurfsmuster

LE 28: Software-Komponenten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Einführung

- Grundlage für den objektorientierten Entwurf ist in der Regel das **OOA-Modell**.
- Das entstehende **OOD-Modell** wiederum bildet die Grundlage für die Implementierung.
- Die Implementierung erfolgt in einer oder mehreren konkreten **Programmiersprachen**.

- Bezeichner-Syntax
  - Alle Namen des OOD-Modells müssen der Syntax der Ziel-Programmiersprache entsprechen.
  - Bezeichnungen aus dem OOA-Modell, die dieser Syntax nicht entsprechen, müssen daher manuell oder automatisch umgewandelt werden.

- Deutsch vs. Englisch
  - In der Systemanalyse
    - In der Regel Deutsch und die jeweilige Fachterminologie
  - In der Entwurfsphase und in der Implementierung
    - Üblich Englisch
      - ermöglicht kürzere Bezeichnungen
      - wird in Klassenbibliotheken durchgängig verwendet
    - Die Verwendung von deutschen Bezeichnern hat jedoch den Vorteil, dass man leicht zwischen selbstgeschriebenen und benutzten Klassen unterscheiden kann.

OOD – Objekt/Klasse (Generische Klasse)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Stereotyp

- Zugehörigkeit einer Klasse zu einer bestimmten Entwurfskomponente, z. B. zur GUI-Schicht, kann gezeigt werden.
- Beispiel:
  - Alle Klassen, die zur Datenverwaltung gehören, werden mit «DB» (data base) und alle Klassen zur Realisierung der Benutzungsoberfläche mit «GUI» (graphical user interface) gekennzeichnet.

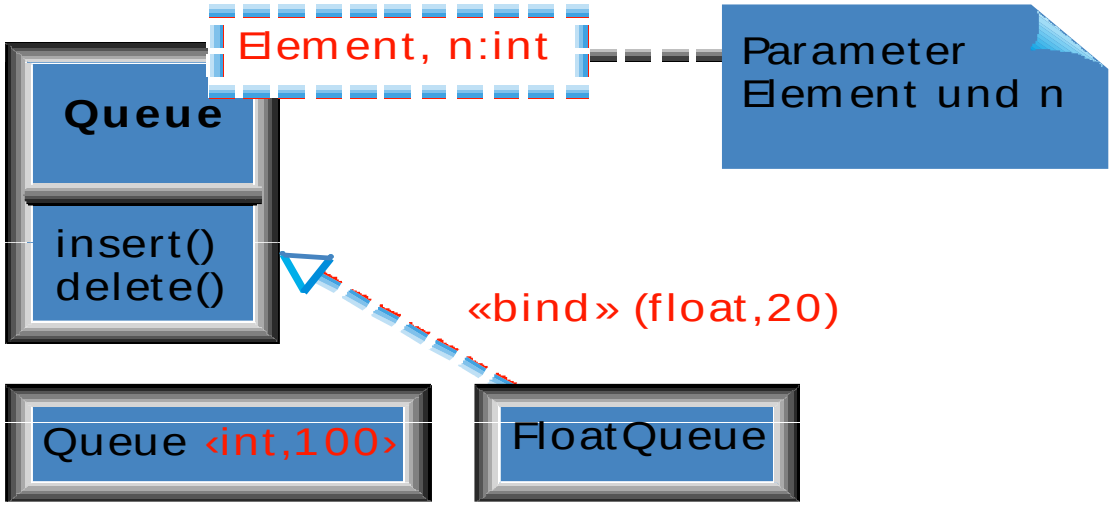
Generische Klasse

- Beschreibt eine Familie von Klassen mit einem oder mehreren formalen Parametern.
- Parameter einer generischen Klasse sind Typ-Parameter oder Konstanten-Parameter.
- Typ-Parameter: Bezeichner, der innerhalb der Klasse wie ein gewöhnlicher Typ verwendet werden kann.
- Konstanten-Parameter: Bezeichner, der innerhalb der Klasse wie eine Konstante des angegebenen Typs verwendet werden kann.
- Formale Parameter müssen an aktuelle Parameter gebunden werden.
  - Für Typ-Parameter: ein konkreter Typ
  - Für Konstanten-Parameter: ein fester Wert
  - Durch die Bindung entsteht eine neue Klasse
  - Beispiel für Meta-Klassen, also Klassen, deren Exemplare (»Objekte«) wieder Klassen sind.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Beispiel generische Klasse

- Es wird eine generische Klasse *Queue* deklariert, die die Operationen `insert()` und `delete()` besitzt.
- Elemente und deren Anzahl, die die Queue verwalten sind noch nicht genauer spezifiziert.
- Der Parameter Element beschreibt einen Typ, daher sind für diesen Parameter keine Angaben notwendig.
- Der Parameter `n` vom Typ `int` gibt die maximale Größe der Warteschlange an.
- Die generische Klasse bildet die Vorlage für die Klasse `Queue <int,100>`, in der maximal 100 int-Werte gespeichert werden können und `FloatQueue`, die maximal 20 float-Werte speichern kann.

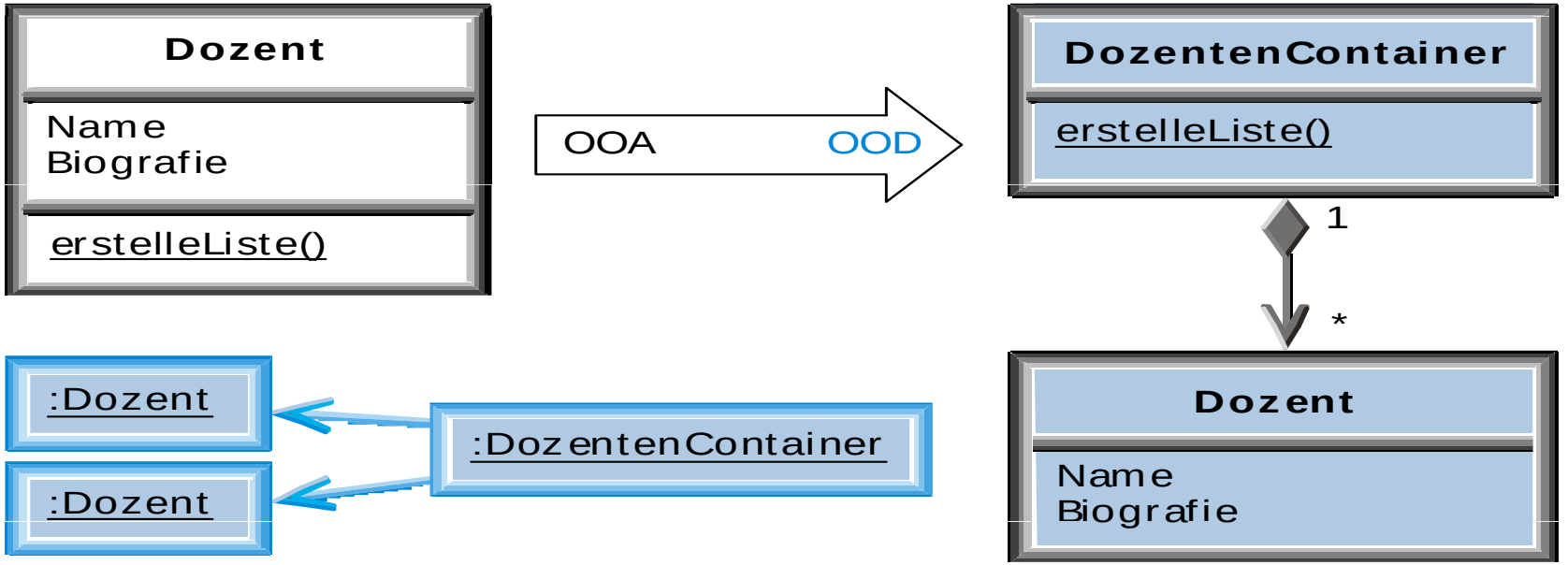


OOD – Objekt/Klasse (Container-Klassen)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Container-Klasse

- verwaltet Menge von Objekten einer Klasse;
- stellt Operationen bereit, um auf die verwalteten Objekte zuzugreifen.
- Ein Objekt der Container-Klasse wird als Container bezeichnet
- Typische Container: Felder (arrays) und Mengen (sets)
- Namenskonvention: Plural des Klassennamen, u. U. gefolgt von dem Namen Container bestehen

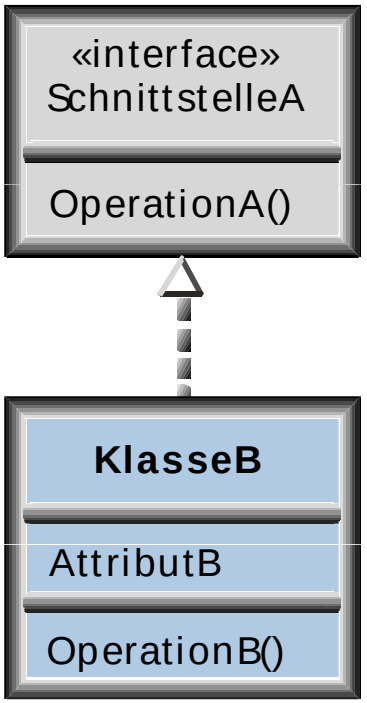


OOD – Objekt/Klasse (Schnittstellen)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

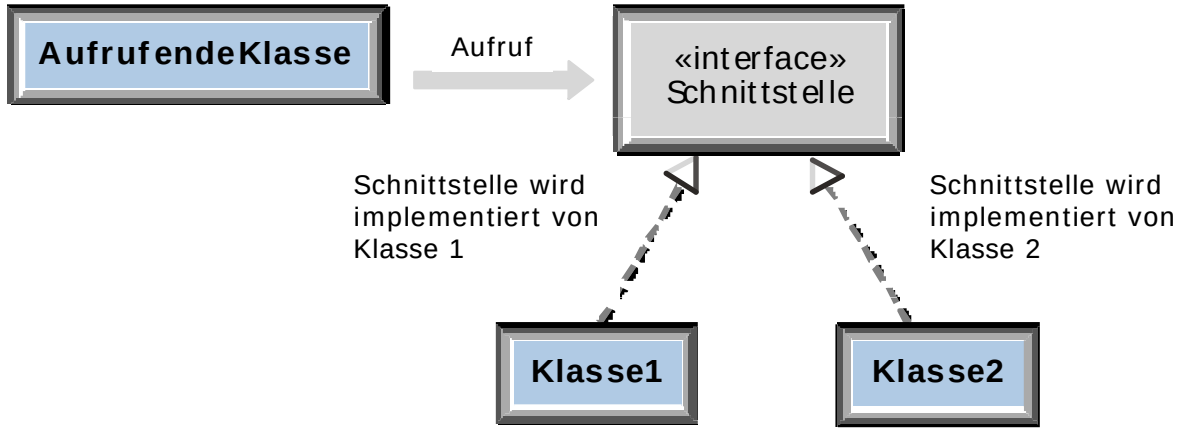
Schnittstelle (interface)

- Funktionale Abstraktionen werden als Operationssignaturen bereitgestellt, die das »Was«, nicht das »Wie« festgelegt.
- Besteht nur aus Operationssignaturen, d. h. sie besitzt keine Operationsrümpfe und Attribute.
- Kann in Vererbungsstrukturen verwendet werden.
- Äquivalent zu einer Klasse, die keine Attribute und ausschließlich abstrakte Operationen besitzt.
- Verschiedene Klassen können dieselbe Schnittstelle unterschiedlich implementieren.
  - Für die aufrufende Klasse ergibt sich dadurch keine Änderung, da die Schnittstelle unverändert bleibt.
- Konvention: Schnittstellennamen am Anfang ein großes „I“ für Interface ergänzen.

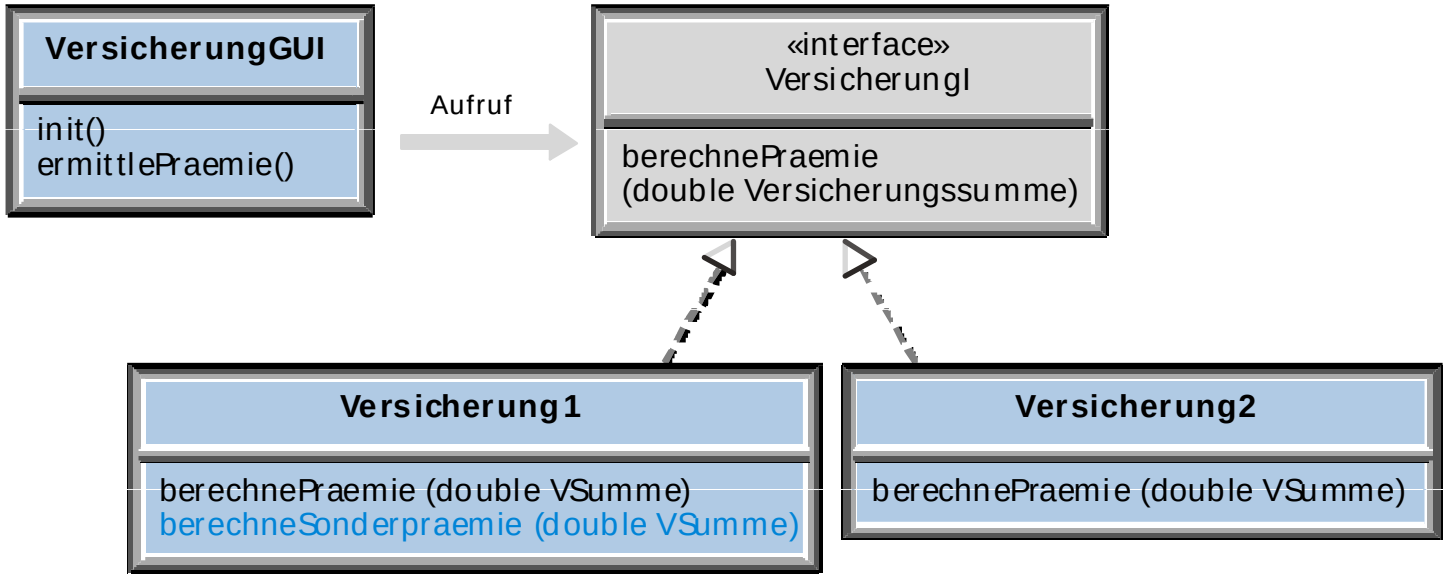


Schnittstellenkonzept

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



- Beispiel:



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

OOD – Attribute

- Attribute sollen prinzipiell als *protected* oder *private* vereinbart werden.
  - Jedes Objekt sieht alle *protected*-Attribute seiner Oberklassen und kann direkt darauf zugreifen.
  - Sind Attribute *private*, dann sehen Objekte die Attribute ihrer Oberklassen nicht, sondern dürfen nur über entsprechende Operationen zugreifen.
  - Vorteil
    - Veränderungen der Attribute wirken sich nicht auf die Unterklassen aus.
    - Die Realisierung der Unterklasse wird unabhängig von der Darstellung der Attribute der Oberklasse.
  - Nachteil
    - Zusätzliche Lese-/Schreiboperationen.
- Klassenattribut: Kann auf zwei Arten im OOD-Modell realisiert werden:
  - Als Klassenattribut
  - Als Objektattribut einer separaten Klasse
    - Diese Klasse besitzt dann nur ein einziges Objekt mit dem Wert des Klassenattributs.
    - Alle Objekte, für die das Attribut gelten soll, müssen dasselbe Objekt der separaten Klasse referenzieren.
- Abgeleitetes Attribut
  - Wichtiges Entwurfskonzept
  - Kann entweder als Attribut – mit entsprechender Konsistenzprüfung – oder durch eine Operation realisiert werden, die stets den aktuellen Wert ermittelt.

Verkapselung und Geheimnisprinzip

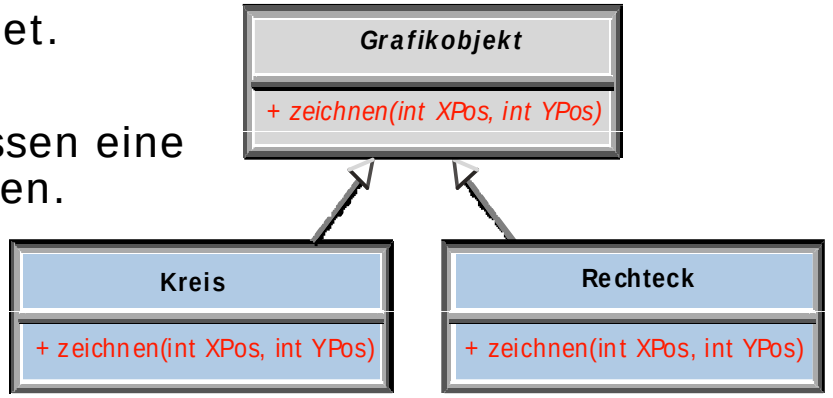
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Geheimnisprinzips (information hiding)
  - Zustand eines Objekts und Implementierung der Operationen außerhalb der Klasse nicht sichtbar.
- Verkapselung (encapsulation)
  - Zusammengehörende Attribute und Operationen, in einer Einheit – der Klasse – gekapselt.
  - Entgegen dem Geheimnisprinzip können die Attribute und die Realisierung der Operationen durchaus nach außen sichtbar sein.
  - z. B. erlauben Java und C++ mit ihren verschiedenen Sichtbarkeiten die Verkapselung ohne und mit Einhaltung des Geheimnisprinzips.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

OOD – Operationen

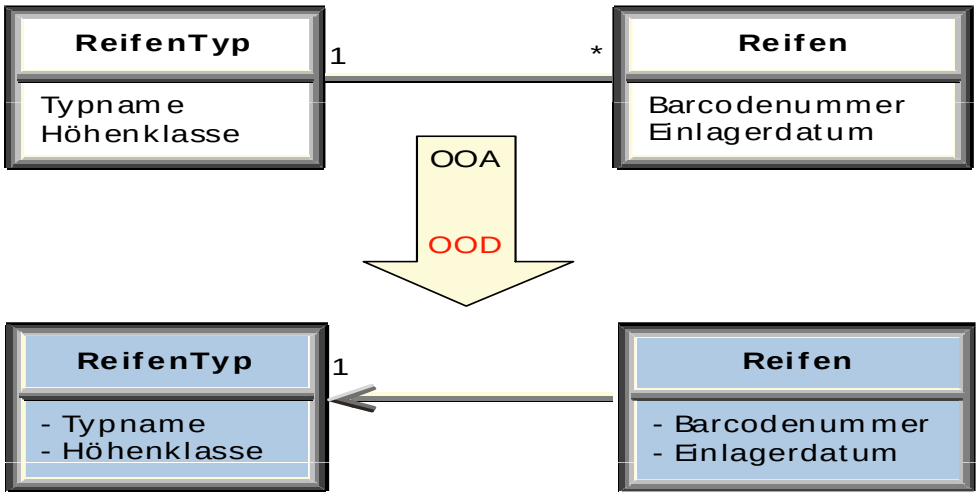
- Sichtbarkeiten (analog wie Attribute)
  - *private (-), protected (#), public (+)*
- Signatur
  - Besteht aus Namen der Operation, den Namen und Typen aller Parameter, Ergebnistyp der Operation.
  - Menge aller Signaturen: Schnittstelle der Klasse
- Beschreibung einer Operation
  - Bei Bedarf kann eine Beschreibung mittels Vor- und Nachbedingungen erfolgen.
  - Vorbedingung beschreibt, welche Bedingungen vor dem Aktivieren einer Operation erfüllt sein müssen.
  - Nachbedingung beschreibt die Änderung, die durch die Operation bewirkt wird.
  - Für die Implementierung einer Operation wird auch der Begriff »Methode« verwendet.
- Abstrakte Operation
  - Werden verwendet, um für Unterklassen eine gemeinsame Schnittstelle zu definieren.
  - Besteht nur aus der Signatur
  - Besitzt keine Implementierung



OOD – Assoziationen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Im Entwurf wird festgelegt, ob Assoziationen uni- oder bi-direktional implementiert werden.
- Man spricht von der Navigation der Assoziation.



- Realisierung mittels Zeigern
  - Jede Richtung einer Assoz-iation kann mittels Zeigern zwischen Objekten realisiert werden.
  - Dann kennt jedes Objekt seine assoziierten Objekte.
  - Durch die Operationen muss sichergestellt werden, dass alle Verbindungen konsistent auf- und abgebaut werden.
  - Eine Kardinalität von 0..1 oder 1 wird dabei durch einen einzelnen Zeiger realisiert.
  - Liegt eine Kardinalität größer 1 vor, dann muss eine Menge von Zeigern gespeichert werden. Wenn keine Ordnung der Assoziation definiert ist, dann können Container-Klassen wie Set, Bag etc. verwendet werden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

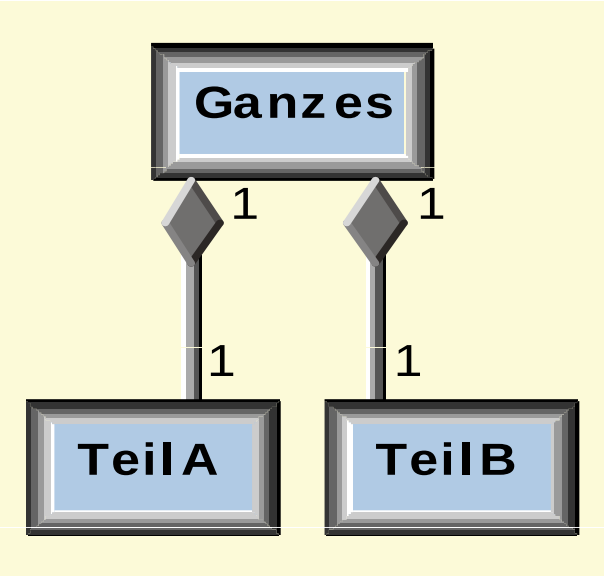
Aggregation / Komposition

- Aggregation
  - Realisierung wie die »normale« Assoziation
  - Ein Ganzes muss jedoch stets seine Teile kennen. Navigation vom Ganzen zu den Teilen möglich.
- Komposition
  - Navigation vom Ganzen zu den Teilen möglich.
  - Operationen, die das Ganze betreffen, müssen sich auch auf seine Teile auswirken.
  - Das Ganze und die Teile sind als Einheit zu betrachten
    - Der Zugriff im Dialog und das Erzeugen der Teile erfolgen immer über das Aggregatobjekt.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Realisierung Komposition

- Realisierung Komposition
  - Kann auch über echtes physisches Enthaltensein (by `value`) realisiert werden.
  - by `value`-Realisierung besitzt den Vorteil, dass die Teile automatisch mit dem Ganzen erzeugt bzw. gelöscht werden.
  - Auch das Kopieren des Aggregatobjekts bezieht sich immer automatisch auf seine Teile.
  - by `value` & by `reference`-Realisierung in C++:




```
class Ganzes
{   TeilA    einTeilA;
    TeilB* einTeilB;

public:
    Ganzes()
    { einTeilB = new TeilB;}
    ~ Ganzes()
    {delete einTeilB;}
};
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Assoziationen

- {ordered}
  - Assoziationen, deren Objektverbindungen (links) geordnet sind.
  - Verwendbar zur Realisierung eine Container-Klasse, die eine Ordnung ihrer Elemente ermöglicht (z.B. Array, Vector).
- {sorted}
  - Ordnungskriterium sind die Elementwerte.
  - z.B. können alle Kundenobjekte nach der Kundennummer sortiert sein.
  - Genauere Informationen durch eine separate Restriktion formulieren.

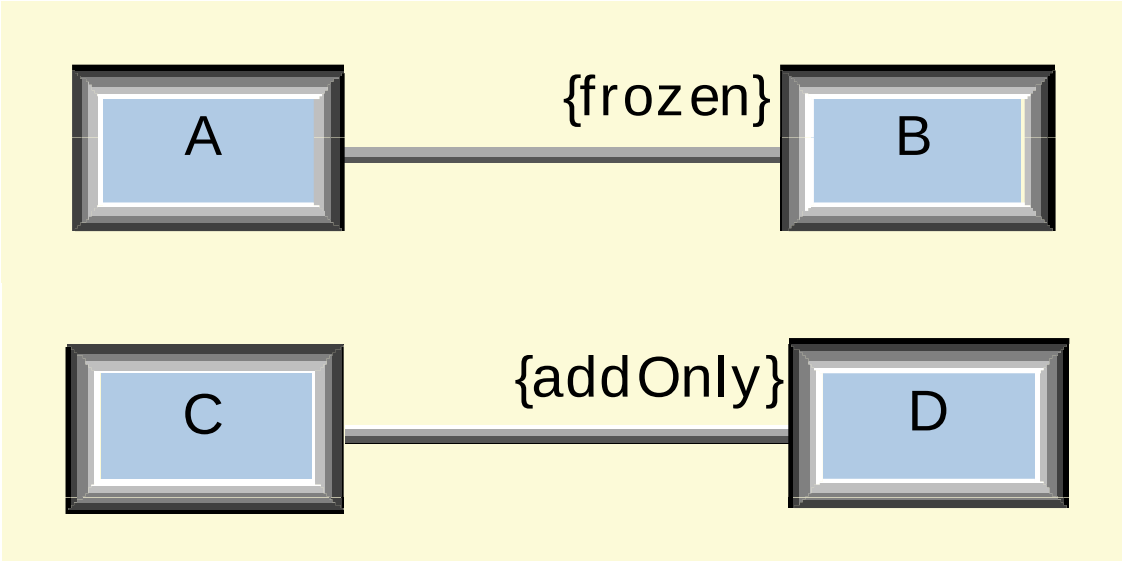
Merkmale können auf jeder Seite einer Assoziation angegeben werden

- {frozen}
  - Objektverbindungen können weder hinzugefügt noch gelöscht oder geändert werden, nachdem ein Objekt der Klasse B erzeugt und initialisiert wurde.
- {addOnly}
  - Dieses Merkmal gibt an, dass für ein Objekt der Klasse D – bei einer many-Kardinalität – zwar weitere Verbindungen eingetragen, vorhandene Verbindungen aber nicht entfernt werden dürfen.

Wird kein Merkmal angegeben, dann können Objektverbindungen beliebig hinzugefügt und entfernt werden.

Assoziationen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



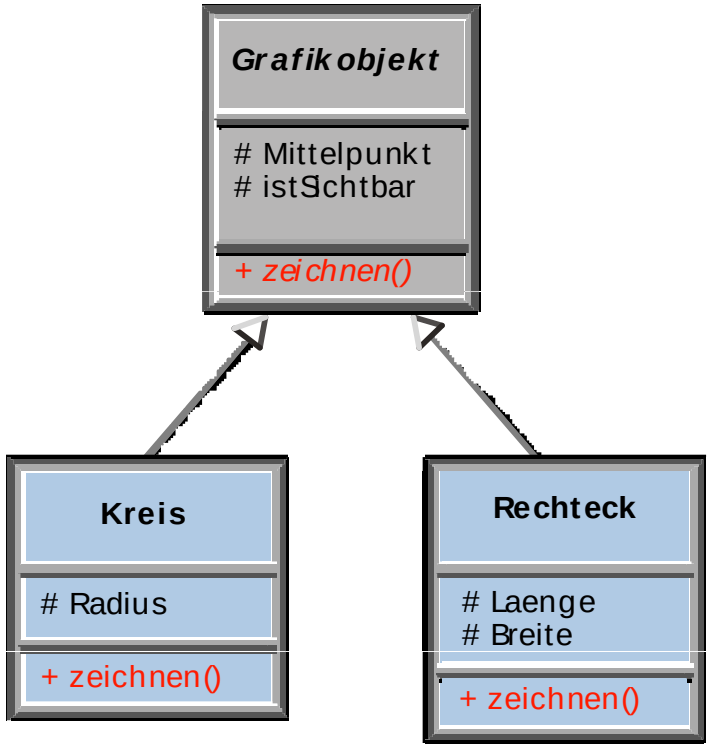
- UML: Sichtbarkeiten
  - +, #, – oder ein Schlüsselwort (z.B. {public}) als Präfix des Rollennamens;

Polymorphismus

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Ermöglicht es, den gleichen Namen für gleichartige Operationen zu verwenden, die auf Objekten verschiedener Klassen auszuführen sind.
- Der Sender muss nur wissen, dass ein Empfängerobjekt das gewünschte Verhalten besitzt.
- Er muss nicht wissen, zu welcher Klasse das Objekt gehört.
- Dieser Mechanismus ermöglicht es, flexible und leicht änderbare Softwaresysteme zu entwickeln.

  - Beispiel:
    - Es wird ein Zeiger `pGrafik` deklariert: `Grafikobjekt pGrafik;`
    - Operationsaufruf `pGrafik.zeichnen()` kann unterschiedliche Wirkungsweisen besitzen.
    - Gilt `pGrafik = new Kreis`, dann wird die Operation `Kreis.zeichnen()` aktiviert.
    - Gilt `pGrafik = new Rechteck`, dann wird `Rechteck.zeichnen()` ausgeführt.



Polymorphismus

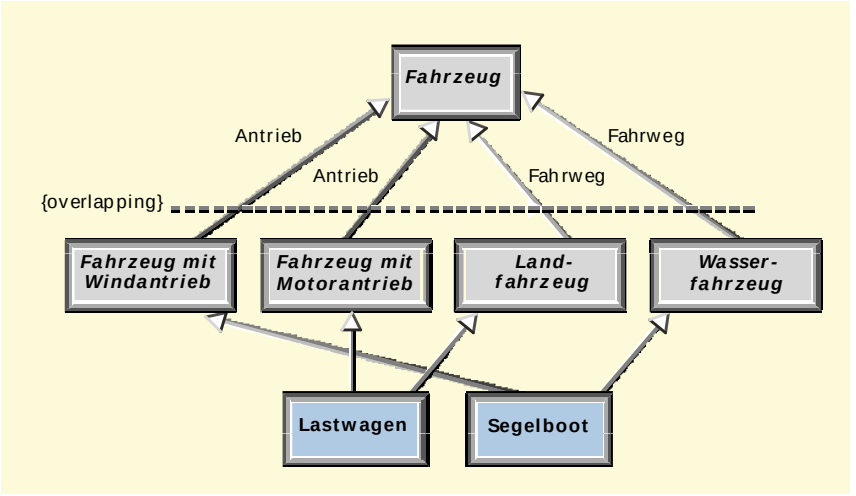
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Erst zur Laufzeit wird bestimmt, ob der Zeiger `pGrafik` auf ein `Kreis`- oder `Rechteck`-Objekt zeigt
- Polymorphismus und spätes Binden (late binding) sind untrennbar verbunden.
- Ist zur Übersetzungszeit die Klasse des Objekts nicht bekannt, dann kann noch nicht bestimmt werden, welche Operation ausgeführt wird.
  - Spätes Binden: Operation wird erst zur Ausführungszeit an ein bestimmtes Objekt gebunden.
  - Java: Alle Operationen sind polymorph
  - C++: Operationen müssen explizit polymorph deklariert werden.

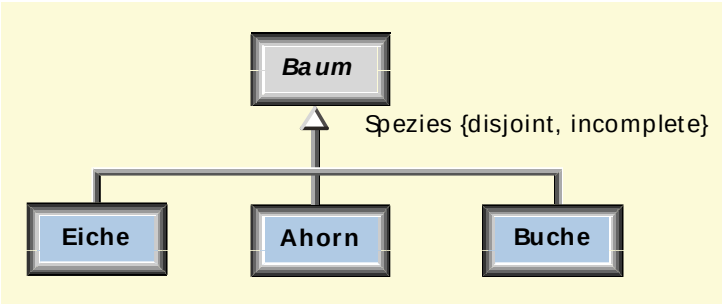
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

OOD – Vererbung

- Einfachvererbung in OOA → plus Mehrfachvererbung in OOD
  - Java: Ersatzform Schnittstellen
  - Beispiel für die Mehrfachvererbung



- overlapping
  - Eigenschaften der Unterklassen überschneiden sich.
- disjoint
  - Eigenschaften der Unterklassen überschneiden sich nicht.
- complete
  - Die Menge der Unterklassen ist vollständig.
  - Weitere Unterklassen werden aufgrund der Problemstellung nicht erwartet.
- incomplete
  - Die betreffende Vererbungsstruktur enthält einen Teil der Unterklassen.
  - Es gibt weitere Unterklassen, die das Modell noch nicht enthält.



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

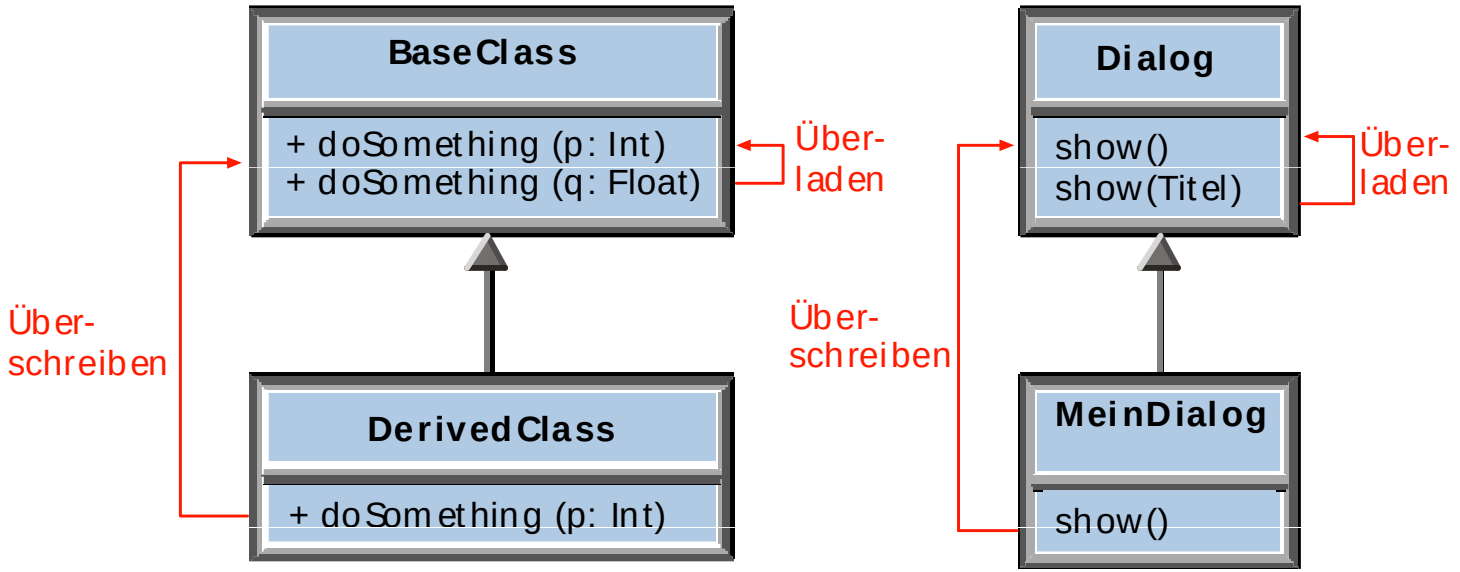
Überschreiben

- Überschreiben bzw. Redefinition
  - Wenn eine Unterklasse eine Operation der Oberklasse – unter dem gleichen Namen – neu implementiert
  - Vorteil: Ein Programmierer, der eine Vererbungsstruktur benutzt, kann die verschiedenen (Unter-)Klassen verwenden und muss sich keine Gedanken darüber machen, zu welcher Unterklasse ein spezielles Objekt gehört
  - Diese Eigenschaft erfordert spätes Binden bzw. die Verwendung polymorpher Operationen
  - Beim Überschreiben einer Operation müssen die Anzahl und Typen der Ein-/ Ausgabeparameter gleich bleiben.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Überschreiben vs. Überladen

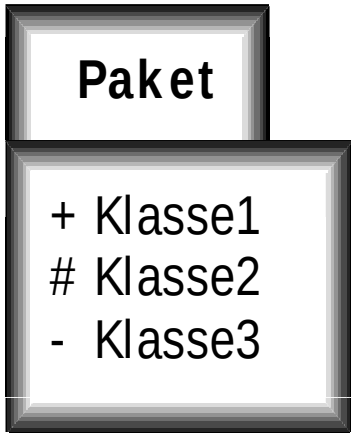
- Überladen, wenn derselbe Operationsname innerhalb einer Klasse mit verschiedenen Parameterlisten verwendet wird.
- Beim Überschreiben muss die Operation der Unterklasse kompatibel mit derjenigen der Oberklasse sein.
  - Dabei kommt es häufig vor, dass bei der Implementierung von `DerivedClass.doSomething()` die gleichnamige Operation der Oberklasse aufgerufen wird.
  - Hierdurch zeigt sich das typische Verhalten der Unterklasse als Erweiterung der Oberklasse.



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

OOD –Paket

- Pakete dienen dazu, (Modell-) Elemente – insbesondere Klassen – zu Gruppen zusammenzufassen und als Ganzes zu behandeln.
- Unterstützen die Darstellung von alternativen Entwürfen oder von Entwürfen für verschiedene Plattformen.
- Pakete können ineinandergeschachtelt werden.
  - Dies ermöglicht eine Modellierung des Systems auf verschiedenen Abstraktionsebenen.
- Sichtbarkeiten
  - Analog zu den Attributen und Operationen einer Klasse
  - +, *public*: Für alle Pakete sichtbar, die das betreffende Paket importieren.
  - #, *protected*: Für alle Pakete sichtbar, die das betreffende Paket spezialisieren.
  - -, *private*: Nur in dem betreffenden Paket sichtbar.
  - Wenn ein Element in einem Paket A sichtbar ist, dann ist es auch in allen Paketen A1, A2 sichtbar, die in A enthalten sind.



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

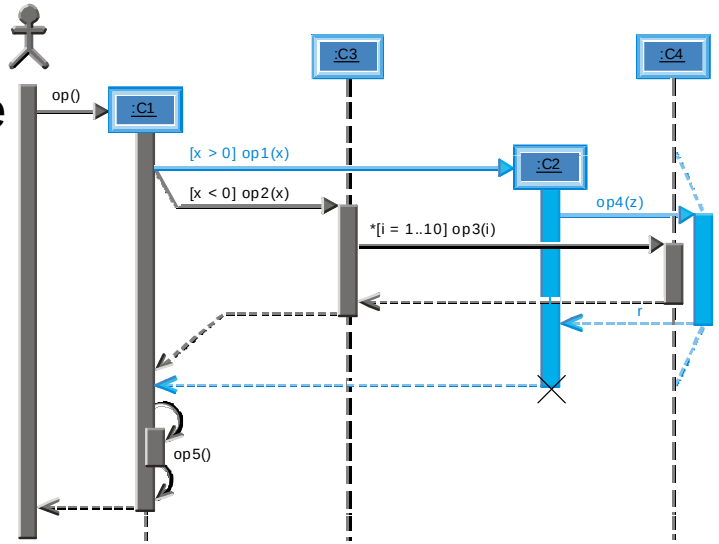
Import, Paketvarianten und -stereotypen

- *import*
  - Zwischen 2 Paketen kann eine import-Beziehung definiert werden.
- Paketvarianten
  - Durch Vererbungsbeziehung dargestellt
  - Das spezialisierte Paket kann geerbte Elemente neu definieren und zusätzliche Elemente hinzufügen.
  - Ein spezialisiertes Paket kann überall dort benutzt werden, wo das allgemeinere Paket verwendet werden kann.
- Stereotypen
  - Der Paketname kann durch einen darüberstehenden Stereotypen ergänzt werden, um die Bedeutung des Pakets im System deutlich zu machen.
  - Die UML definiert für Pakete eine Reihe von Standard-Stereotypen
    - *subsystem*: Das betreffende Paket modelliert ein unabhängiges Teilsystem.
    - *system*: Das betreffende Paket repräsentiert das gesamte System.

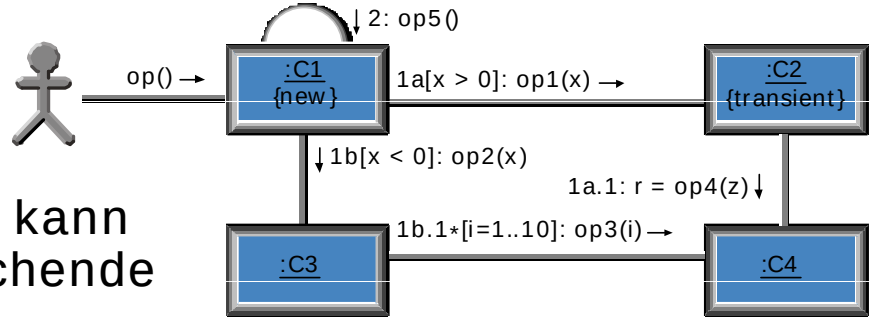
OOD – Szenario

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Zusammenarbeit der Objekte kann nur mittels geeigneter Szenarios durch Sequenz- und Kollaborationsdiagramme beschreiben werden.
- Sequenzdiagramme
  - Eine Verzweigung des Kontrollflusses tritt auf, wenn mehrere Botschaftspfeile vom selben Punkt ausgehen.
  - Jeder Pfeil ist mit einer Bedingung (guard condition) beschriftet.
  - Die Objektlinie kann in zwei oder mehrere Linien verzweigen, die zu einem späteren Zeitpunkt wieder zusammengeführt werden.



- Kollaborationsdiagramm
  - Ausgangsbasis bilden die Objekte und ihre Verbindungen untereinander.
  - Die Reihenfolge der Operationen kann man zum Schluss durch entsprechende Nummern hinzufügen.



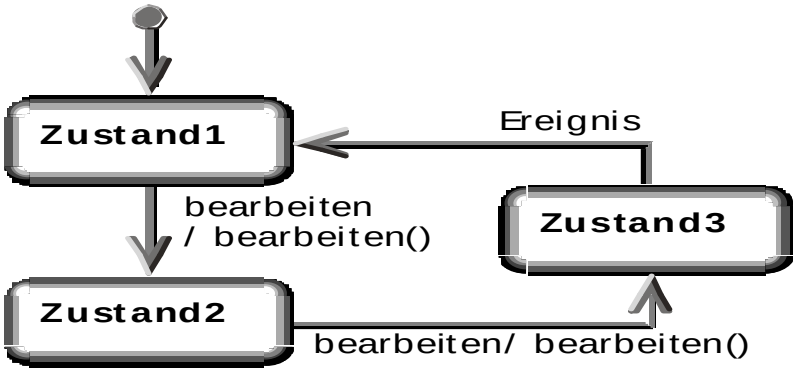
Lebenszyklus und Aktivitätsdiagramme

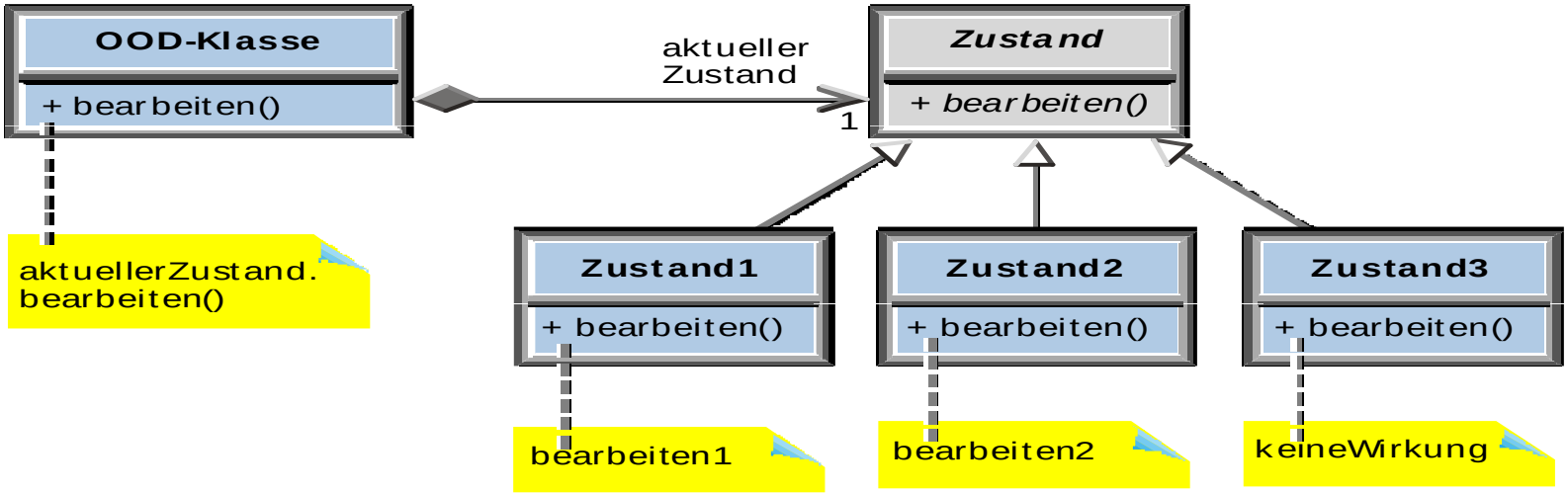
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Lebenszyklus
  - Lebenszyklen aus Entwurfssicht überarbeiten und mit den entsprechenden Operationen ergänzen.
  - Realisierung
    - Jede Klasse mit einem Lebenszyklus erhält im Entwurf ein private-Attribut `classState`.
    - In diesem Attribut wird der aktuelle Zustand des Objekts gespeichert.
    - Jede Operation, die im Lebenszyklus aufgeführt ist, muss dieses Attribut abfragen, bevor sie ihre Verarbeitung durchführt.
    - Ist mit dieser Operation ein Zustandswechsel verbunden, dann muss sie das Zustandsattribut aktualisieren.
- Aktivitätsdiagramm (activity chart)
  - Sonderfall des Zustandsdiagramms
  - Dient dazu, die interne Verarbeitung zu spezifizieren, wobei jeder Zustand einen Schritt eines Algorithmus beschreibt.
  - Im Entwurf kann es sinnvoll zur Beschreibung von komplexen Operationen eingesetzt werden.

Zustandsmuster

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme





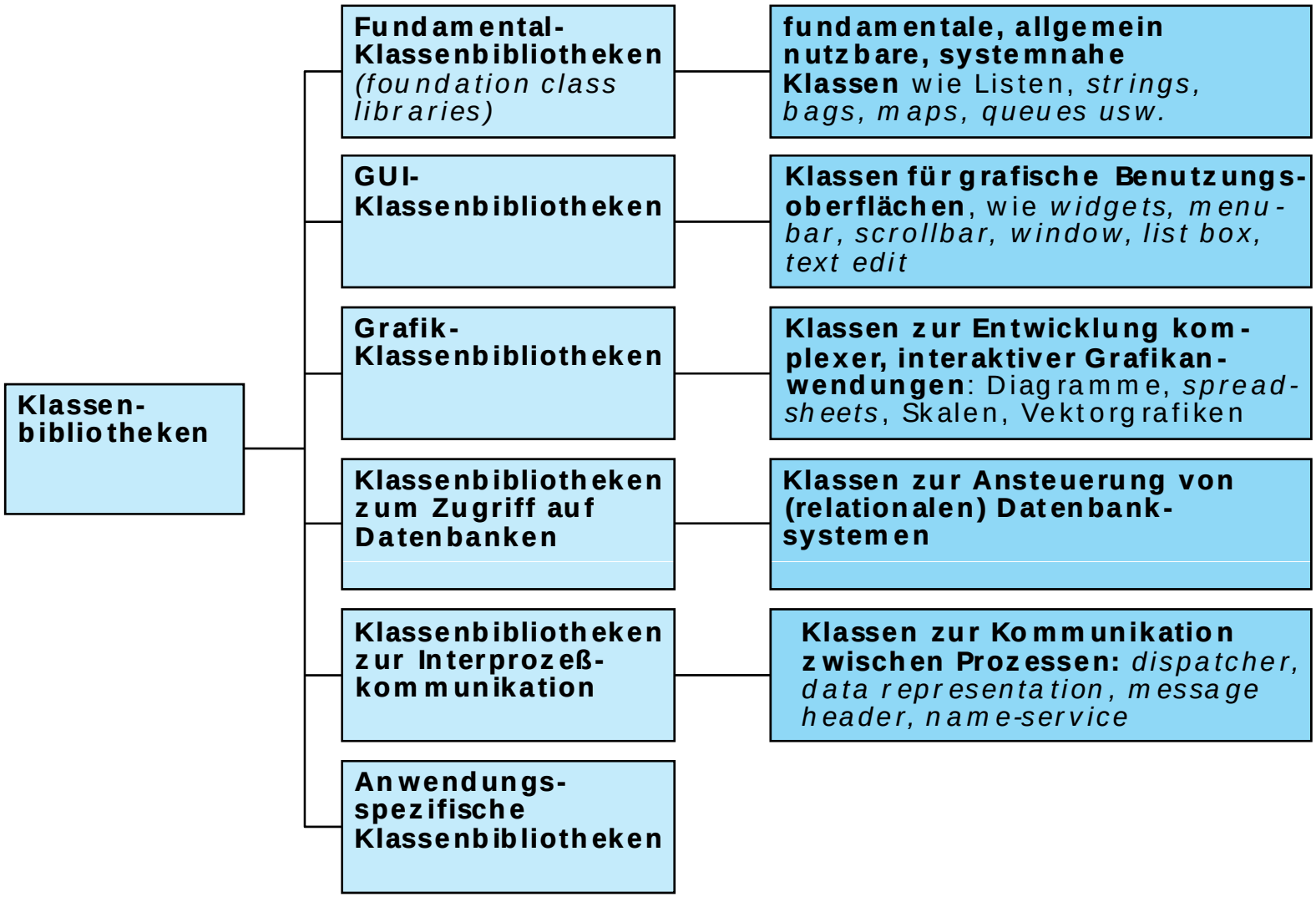
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Klassenbibliotheken

- Organisierte Softwaresammlungen, aus der der Entwickler nach Bedarf Einheiten verwendet.
- Bilden die Voraussetzung, um »Wiederverwendung im Kleinen« zu realisieren.
- Eigenentwicklung und kommerzielle Angebote
- Vorteile von Klassenbibliotheken
    - Eigener Aufwand wird gespart
    - Bessere Qualität, da bereits oft eingesetzt
    - Höhere Flexibilität, da Varianten zur Auswahl
    - Von Spezialisten entwickelt
    - Auswahl zwischen mehreren Herstellern
    - Oft plattformübergreifend
- Nachteile
    - Möglicherweise hoher Einarbeitungsaufwand
    - Evtl. Namenskonflikte beim Einsatz mehrerer Bibliotheken
    - Verschiedene Klassenbibliotheken müssen oft durch eine Zwischenschicht gekoppelt werden.
- Entwurfsziele
    - hohe Laufzeit- und Speichereffizienz
    - volle Nutzung des Sprachumfangs
    - Einfachheit der Benutzung
    - Plattformunabhängigkeit

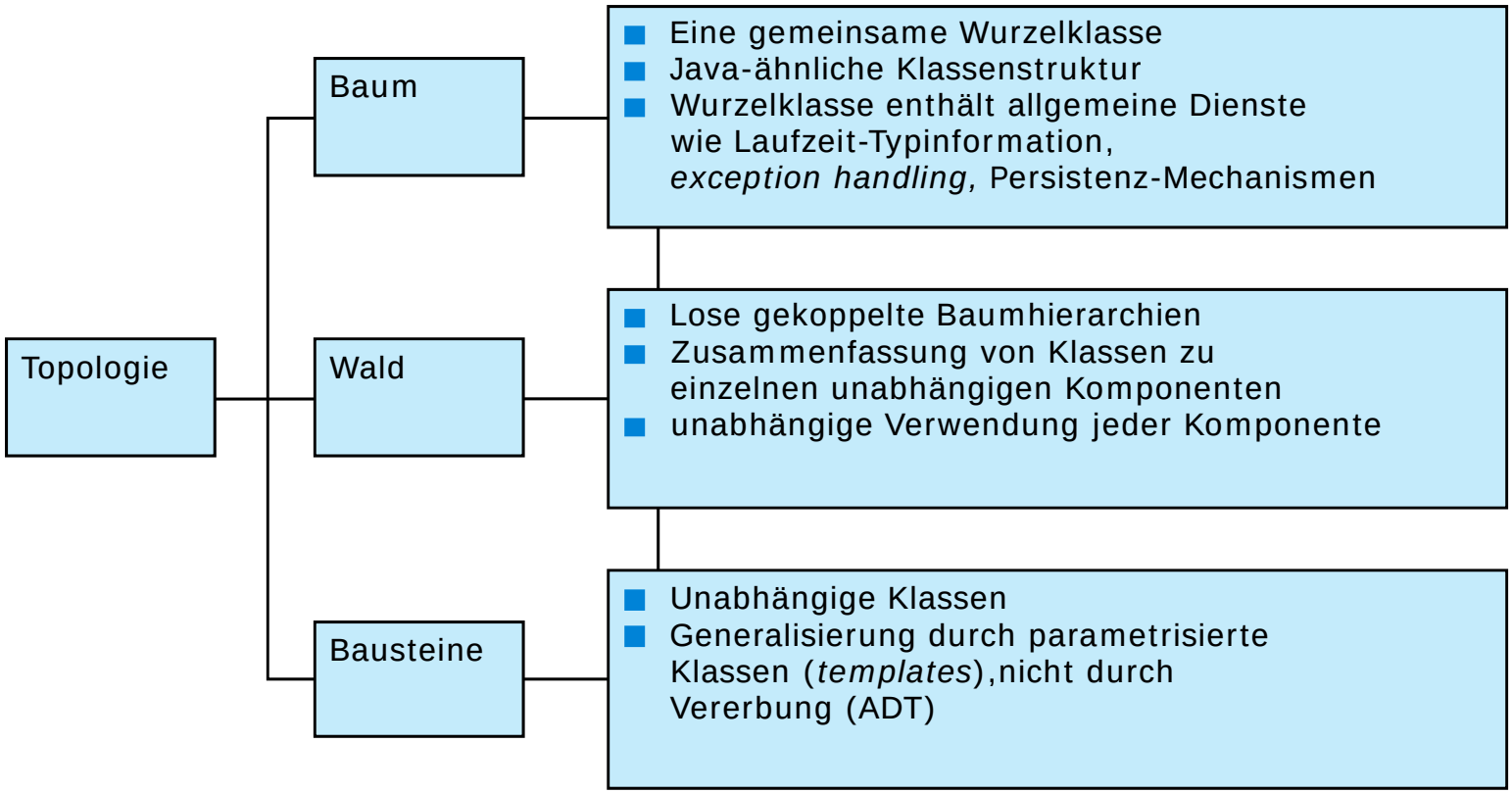
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Anwendungsgebiete



Topologien

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Baum- und Waldtopologie

- Baumtopologie und ihre Bewertung
  - Nutzung der Vererbung
  - alle Klassen können allgemeine Ober-Dienste nutzen
  - leichte Erweiterbarkeit
  - sehr komplexe Baumhierarchien
  - hoher Lernaufwand
  - Anwender muss alle Oberklassen berücksichtigen
  - Overhead durch nicht benötigte Klassen
  - wenn Kopplung durch Aggregation/Assoziationen, dann Einbettung benachbarter Unterbäume
  - Probleme durch Mehrfachvererbung
  - Vererbung oft überstrapaziert
  - Anwendungsbereich: GUI-Bibliotheken

- Waldtopologie und ihre Bewertung
  - leichter Überblick durch flachere Hierarchie
  - leichte Nutzung
  - abgegrenzter Aufgabenbereich
  - höhere Performance als ein Baum
  - manche Lösungen weniger elegant als ein Baum
  - Beziehungen zwischen Teilbäumen sehr mächtig
  - Anwendungsbereiche : Fundamentalklassen

Bausteintopologie

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Bausteintopologie und ihre Bewertung
  - große Unabhängigkeit der einzelnen Klassen
  - hohe Flexibilität
  - gute Effizienz
  - Erweiterungen haben keine globalen Auswirkungen
  - »Explosion« des Objektcodes durch die Generierung von Klassen mit Hilfe des template-Mechanismus
  - Anwendungsbereich: Fundamentalklassen

Rahmenwerke und Entwurfmuster

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Ein durch den Software-Entwickler anpassbares oder erweiterbares System kooperierender Klassen.
- Es besteht aus konkreten und insbesondere aus abstrakten Klassen, die Schnittstellen definieren.
- Im Allgemeinem wird vom Anwender des Rahmenwerks erwartet, dass er Unterklassen definiert, um das Rahmenwerk zu verwenden und anzupassen.
- Diese selbstdefinierten Unterklassen empfangen Botschaften von den vordefinierten Rahmenwerk-Klassen.
- Zwingt den Bibliotheksbenutzer dazu, die Architektur der eigenen Anwendung in die Bibliothek einzupassen.

Entwurfmuster

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Entwurfsmuster (design pattern)
  - Gibt eine bewährte generische Lösung für ein immer wiederkehrendes Entwurfsproblem an, das in bestimmen Situationen auftritt
  - Analog: Analysemuster in OOA
  - Entwurfsmuster sind i. Allg. technischer, detaillierter und komplexer als Analysemuster.
- Entwurfsmuster...
  - unterstützen die Wiederverwendung von Lösungen
  - dokumentieren existierende und erprobte Entwurfserfahrungen
  - benennen und erklären wichtige Entwürfe
  - helfen bei der Auswahl von Entwurfsalternativen
  - verhelfen dem Entwerfer schneller zum richtigen Entwurf
  - bieten ein gemeinsames Entwurfs-Vokabular und Verständnis für eine Gruppe von Entwicklern.

Beschreibungsschema

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Name des Musters
- Problembeschreibung
  - Das Problem und der Kontext werden erklärt
- Lösungsbeschreibung
  - Gibt die Elemente an, die den Entwurf, die Beziehungen, Verantwortlichkeiten und die Zusammenarbeit ausmachen
  - Die Lösung gibt keinen konkreten Entwurf oder eine Implementierung an
- Konsequenzen
  - Ergebnisse und »trade-offs«
  - Auswirkungen eines Muster auf Flexibilität, Erweiterbarkeit und Portabilität.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Klassifikation von Entwurfmustern

		Zweck		
		erzeugendes Muster	**strukturelles Muster**	**Verhaltensmuster**
Gültig-keits-bereich	**Klasse**	*factory method*	*adapter class*	*interpreter* *template method*
	Objekt	*abstract factory* *builder* *prototype* *singleton*	*adapter (object)* *bridge* *composite* *decorator* *facade* *flyweight* *proxy*	*chain of responsibility* *command* *iterator* *mediator* *memento* *observer* *state* *strategy* *visitor*

Zweck:

- Erzeugendes Muster (creational pattern):
  - Erzeugen von Objekten
- Strukturelles Muster (structural pattern):
  - Komposition von Klassen und Objekten
- Verhaltensmuster (behavioral pattern):
  - Kommunikation und Verantwortlichkeiten zwischen Objekten

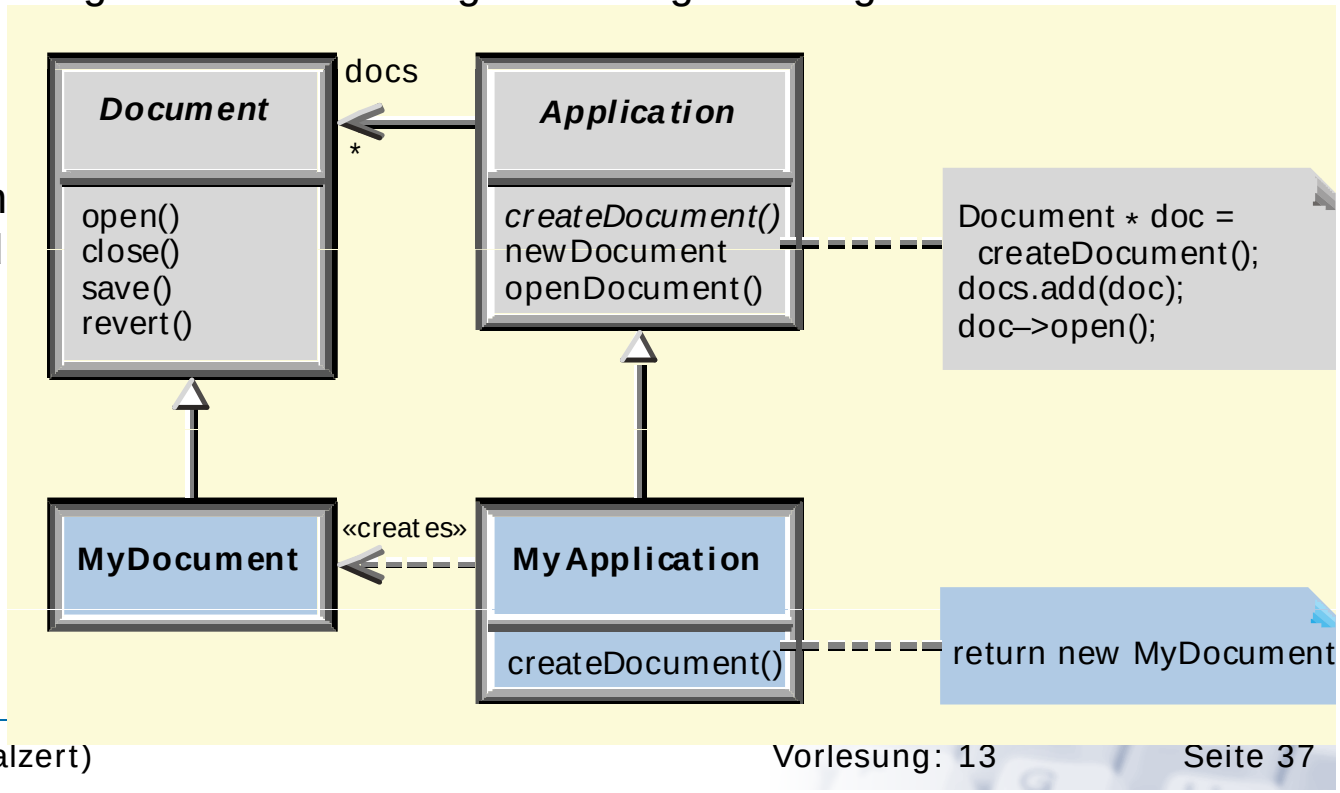
Gültigkeitsbereich:

- Klassen (statisch)
  - Vererbung
- Objekte (dynamisch)
  - Assoziationen, Aggregationen.

Das Fabrikmethode-Muster

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

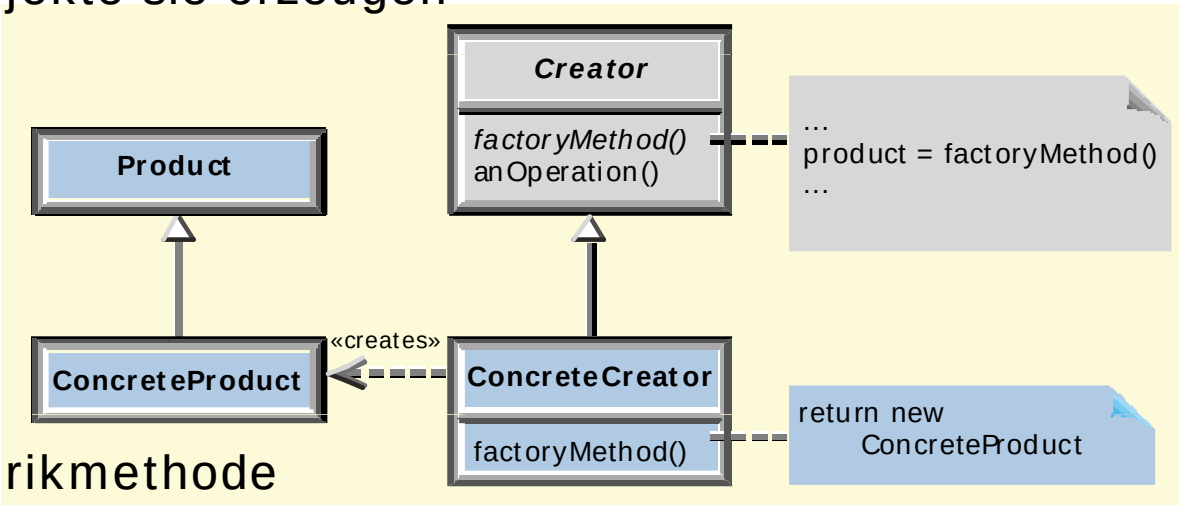
- Zweck
  - Klassenbasiertes Erzeugungsmuster
  - Bietet eine Schnittstelle zum Erzeugen eines Objekts an, wobei die Unterklassen entscheiden, von welcher Klasse das zu erzeugende Objekt ist
  - Auch bekannt als »virtueller Konstruktur« (virtual constructor)
- Motivation
  - Dieses Rahmenwerk eignet sich für das gleichzeitige Anzeigen mehrerer Dokumente
  - Es verwendet die beiden abstrakten Klassen Application und Document und modelliert eine Assoziation zwischen ihren Objekten.



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Das Fabrikmethode-Muster

- Anwendbarkeit
  - Wenn eine Klasse die von ihr zu erzeugenden Objekte nicht im Voraus kennen kann
  - Wenn eine Klasse benötigt wird, deren Unterklassen selber festlegen, welche Objekte sie erzeugen
- Struktur

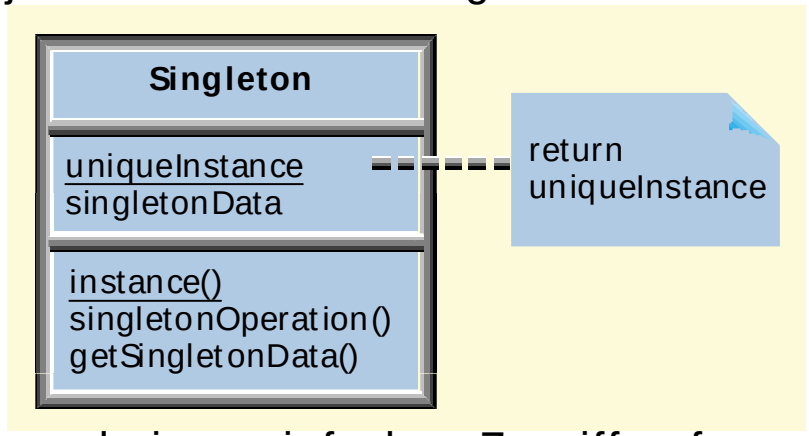


- Interaktionen
  - Der Creator verlässt sich darauf, dass Unterklassen die Fabrikmethode korrekt implementieren
- Konsequenzen
  - Fabrikmethoden verhindern es, dass anwendungsspezifische Klassen in den Code des Rahmenwerks eingebunden werden müssen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Das *Singleton*-Muster

- Zweck
  - Objektbasiertes Erzeugungsmuster
  - Stellt sicher, dass eine Klasse genau ein Objekt besitzt und ermöglicht einen globalen Zugriff
- Motivation
  - Bei manchen Klassen ist es notwendig, dass es genau ein Objekt gibt
  - Auf dieses Objekt muss oft von mehreren anderen Objekten zugegriffen werden
  - Daher muss der Zugriff einfach sein
  - Die Singleton-Klasse muss garantieren, dass nur ein Exemplar erzeugt werden kann und einen einfachen Zugriff auf dieses Exemplar ermöglichen.



- Interaktionen
  - Clients holen sich ausschließlich über die Klassenoperation instance() eine Referenz auf das einzige Objekt
- Konsequenzen
  - Das Singleton-Muster ist eine Verbesserung gegenüber globalen Variablen (die es in Java nicht gibt)
  - Die Singleton-Klasse kann durch Unterklassen spezialisiert werden
  - Werden später mehrere Exemplare benötigt, dann kann diese Änderung leicht durchgeführt werden.

Das *Singleton*-Muster

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

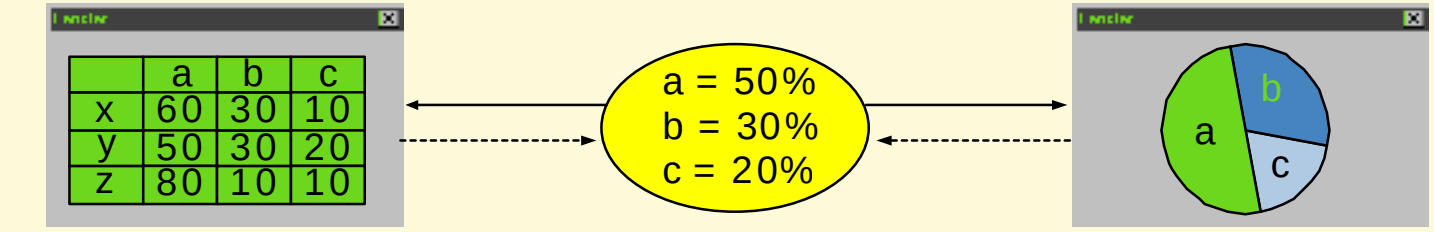
- Klasse Singleton definiert die Klassenoperation instance(), die es dem Client ermöglicht, auf das einzige Exemplar zuzugreifen

```
class Singleton
{ private static Singleton uniqueInstance;
  //macht den Konstruktor nach aussen unsichtbar
  protected Singleton();
  public static Singleton instance()
  {if (uniqueInstance == null)
   //es existiert noch kein Exemplar
    { uniqueInstance = new Singleton();
    }
    return uniqueInstance;
  }
}
```

Das Beobachter-Muster

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

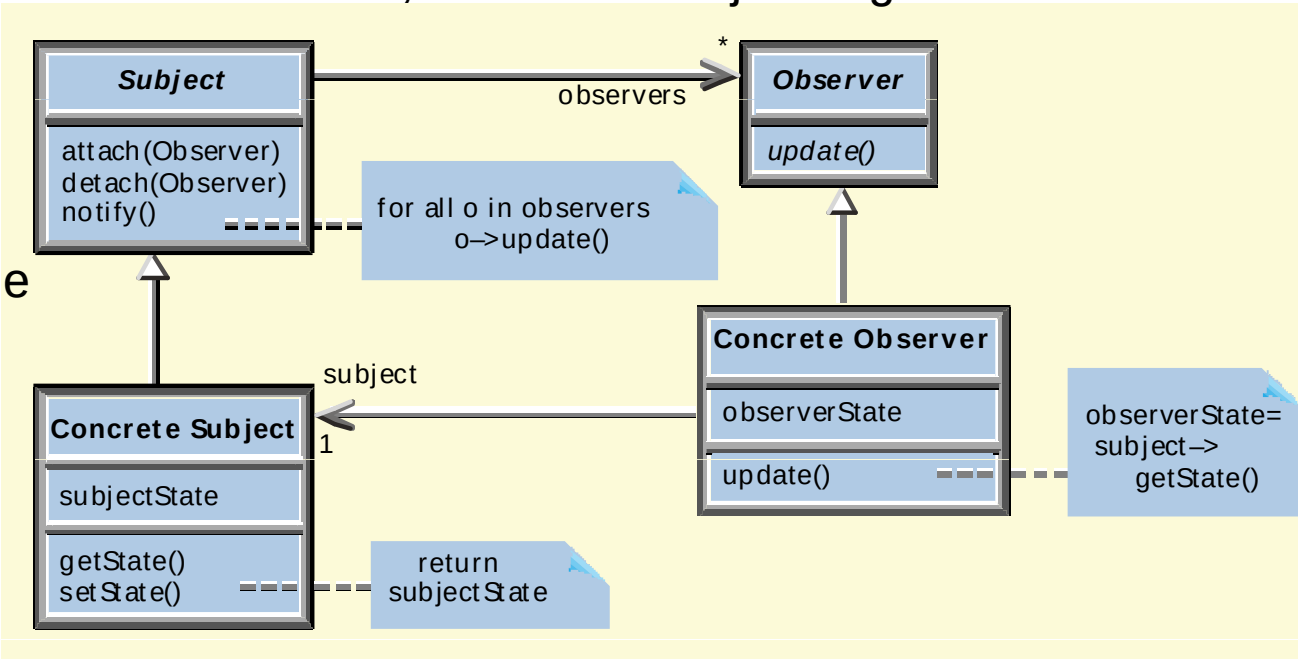
- Zweck
  - Objektbasiertes Verhaltensmuster
  - Es sorgt dafür, dass bei der Änderung eines Objekts alle davon abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.
- Motivation
  - Ein Objekt enthält Anwendungsdaten
  - Diese sollen auf verschiedene Arten angezeigt werden, z.B. als Tabelle und als Kreisdiagramm:



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Das Beobachter-Muster

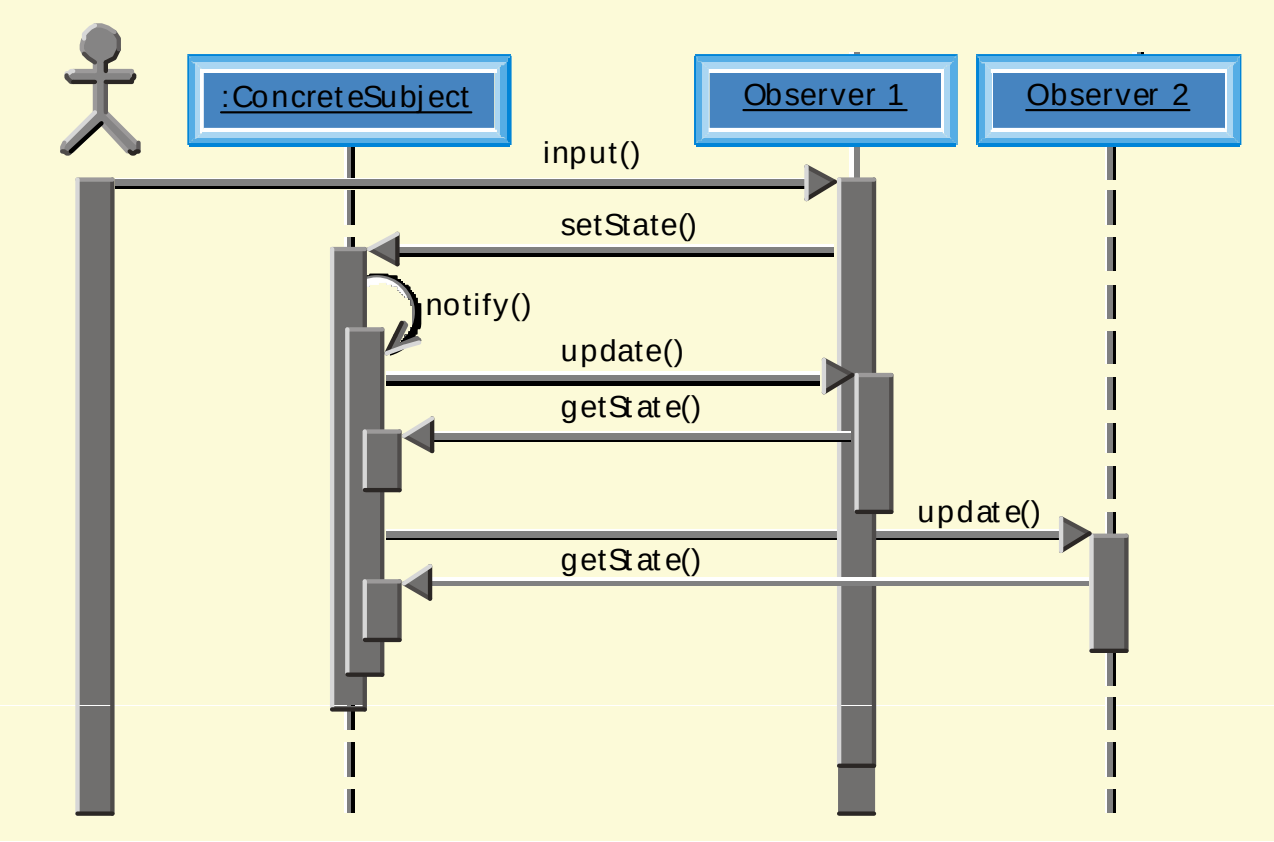
- Anwendbarkeit
  - Eine Abstraktion besitzt zwei Aspekte, die wechselseitig voneinander abhängen
  - Die Kapselung in zwei Objekte ermöglicht es, sie unabhängig voneinander wiederzuverwenden oder zu modifizieren
  - Die Änderung eines Objekts impliziert die Änderung anderer Objekte, und es ist nicht bekannt, wie viele Objekte geändert werden müssen
  - Ein Objekt soll andere Objekte benachrichtigen, und diese Objekte sind nur lose gekoppelt.



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Das Beobachter-Muster

- Interaktionen



- Konsequenzen
  - Das Beobachter-Muster ermöglicht es, Subjekte und Beobachter unabhängig voneinander zu modifizieren
  - Beobachter und Subjekte können einzeln wiederverwendet werden
  - Neue Beobachter können ohne Änderung des Subjekts hinzugefügt werden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zusammenfassung

Das Konzept der Klasse wird im Entwurf um **generische Klassen**, **Container Klassen** und **Schnittstellen** erweitert. Für Attribute und Operationen wird die Notation um Sichtbarkeit erweitert. Bei Operationen ist außerdem die komplette Signatur anzugeben. Die Notation von Assoziationen wird um die Navigation und die Sichtbarkeit erweitert. Assoziationen können auf verschiedene Arten (z. B. mittels Zeigern) realisiert werden.

Der **Polymorphismus** ermöglicht es, flexible Programme zu entwickeln. Im Gegensatz zur Analyse tritt im Entwurf häufig Vererbung auf, wobei außer der Einfachvererbung auch die Mehrfachvererbung vorkommen kann. Zusammenhängende Funktionsabläufe können mittels **Szenarien** dokumentiert werden. Daher sind **Sequenz- und Kollaborations-diagramme** im Entwurf von besonderer Bedeutung. Komplexe Lebens-zyklen aus der Analyse können gut mit dem **Zustandsmuster** in den Entwurf transformiert werden.

Entwurfsmuster beschreiben Lösungen für immer wieder kehrende Entwurfsprobleme. Während Muster nur abstrakte Lösungen bieten, stellen **Rahmenwerke** (frameworks) Klassen bereit, die als Basisklassen für die neu zu erstellende Anwendungen verwendet werden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Literatur

- Balzert H., Lehrbuch der Softwaretechnik, 2. Auflage, Spektrum Akademischer Verlag,Heidelberg, 2000.

- Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

- Unified Modeling Language 1.1, UML Summary Notation Guide, UML Semantics Object Contraint Language Specification, Rational Software Corporation, Santa Clara, Sep. 1997

- Berard E., Essays on Object-Oriented Software-Engineering, Volume 1, Prentice Hall, Englewood Cliffs, 1993

- Booch G., Object-Oriented Analysis and Design with Applications, 2. Edition, The Benjamin/Cummings Publishing Company, 1994

Überblick LE28

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

LE 27 Objektorientiertes Design

LE 28 Software Komponenten

- Halbfabrikate und ihre Schnittstellen
- JavaBeans
- COM
- ActiveX und OLE

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Begriffe

- Komponentenbasierte Softwareentwicklung
  - Erlaubt die einfache, schnelle und preiswerte Herstellung individueller, integrierter Anwendungen durch Zusammenbau von vorgefertigten Halbfabrikaten bzw. Komponenten.
- Dazu benötigt man Halbfabrikate, die
  - im Allgemeinem kleiner als Anwendungen sind, d. h. einen stärkeren Komponenten- bzw. Bausteincharakter besitzen;
  - deutlich größer als Klassen sind, d. h. mehr Funktionalität kapseln;
  - Komplexität verbergen.
- Halbfabrikat (component-ware)
  - Ist ein abgeschlossener, binärer Software-Baustein, der eine anwendungsorientierte, semantisch zusammengehörende Funktionalität besitzt, die nach außen über Schnittstellen zur Verfügung gestellt wird.
  - Beim Entwurf des Halbfabrikats wurde auf hohe Wiederverwend-barkeit großer Wert gelegt.
  - Beispiele
    - Rechtschreibprüfung, Silbentrennung, Seitenvorschau, Druckknöpfe, Textfelder, Tabellen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Einsatz von Software-Komponenten

- Vorgehensweise
  1. Auswahl eines geeigneten Architekturrahmens
  2. Auswahl geeigneter Halbfabrikate
  3. Anpassung der Halbfabrikate
  4. Verbinden der Halbfabrikate
  5. Überprüfung, ob das entstehende Produkt den gewünschten Anforderungen entspricht
- Notwendige Voraussetzungen
  - Sprachunabhängigkeit
  - Plattformunabhängigkeit
  - Verteilbarkeit
- Komponentenmodelle für Clients
  - JavaBeans-Modell von Sun
  - COM/ActiveX-Modell von Microsoft
- Komponentenmodelle für Server
  - COM+ von Microsoft
  - Enterprise JavaBeans von Sun
  - CORBA-Modell der OMG

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

JavaBeans

- Jede Java-Klasse ist im Prinzip eine JavaBean
- Um jedoch die Kompositions- und Anpassungsfähigkeit einer JavaBean optimal zu unterstützen, müssen gewisse Konventionen beachtet werden.
- Eigenschaften einer JavaBeans-Komponente:
  - Introspektion: Komponente besitzt die Fähigkeit, relevante Informationen nach außen offen zu legen.
  - Anpassbarkeit: Ein Entwickler, der die Komponente nach ihrer Übersetzung und Auslieferung benutzt, kann sie an seine Bedürfnisse in gewissem Maße anpassen. Er kann Eigenschaften setzen, die das Erscheinungsbild und Verhalten beeinflussen.
  - Eigenschaften :
    - Indizierte Eigenschaften
      - Können mehrere Werte annehmen, die über einen Index angesprochen werden.
    - Gebundene Eigenschaften
      - Werden verwendet, um Änderungen von Eigenschaften externen Beobachtern mitzuteilen .
  - Ereignisse: Eine Komponente kann sich zur Laufzeit bei einer anderen Komponente als Empfänger eines Ereignisses an- und abmelden.
  - Operationen: Alle zusätzlich zu den `get`- und `set`-Operationen implementierten öffentlichen Operationen werden nach außen exportiert.

Component Objekt Model

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- COM (Component Objekt Model) ist Bestandteil der Windows-Betriebssysteme
- Im Gegensatz zu JavaBeans definiert COM einen binären Standard
  - Keine Vorgabe, wie die Bindung einer konkreten Programmiersprache an COM erfolgen soll.
- Unabhängig von Programmiersprachen
- Erzeugung von Objekten
  - Clients können von einer COM-Komponente Objekte erzeugen und diese COM-Objekte benutzen, indem sie einen Zeiger auf eine Schnittstelle der Komponente anfordern.
  - Über diesen Zeiger kann der Client dann Operationen der Komponente aufrufen.
- COM-Komponenten implementieren im Allgemeinem mehrere Schnittstellen.
- Besitzen zwei Schnittstellen-Namen
  - Textueller Name, der nicht eindeutig ist und per Konvention mit dem Buchstaben »I« beginnt.
  - GUID (Global Unique Identifier) – global eindeutige Nummer, die nach einem speziellen Algorithmus gebildet wird und in Zeit und Raum eindeutig ist.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

COM-Komponenten

- Binärcode
- kann eine oder mehrere Schnittstellen implementieren
- können nicht voneinander erben
- nur Schnittstellen-Vererbung
- Es kann nur über ihre Schnittstellen kommuniziert werden.
- Ein Client kann einen Zeiger auf alle Schnittstellen, die eine Komponente implementiert, mit Hilfe der Operation `QueryInterface` und der entsprechenden IID erhalten.
- *Wie erzeugt man nun aber COM-Objekte und wie gelangt man an den ersten Schnittstellenzeiger?*
  - Jede COM-Komponente wird, wie die Schnittstellen, über zwei Bezeichner angesprochen
    - eine GUID
    - einen Namen
  - Eine COM-Komponente wird auch als COM-Klasse bezeichnet und die zugehörige GUID als Klassen-Bezeichner (*class identifier*) oder kurz CLSID.
  - Die CLSID einer Komponente wird zusammen mit dem Aufenthaltsort des Moduls, das sie implementiert, in der Windows Registry abgelegt.

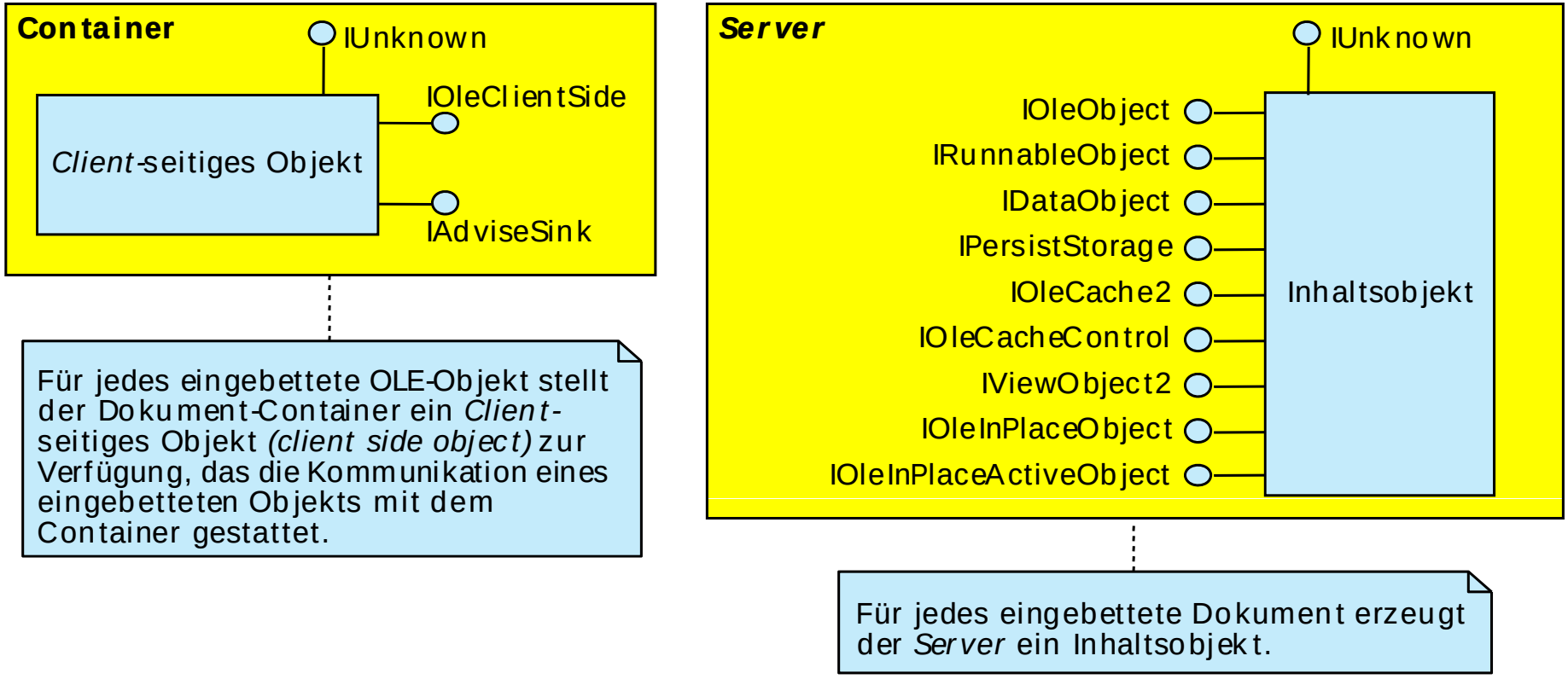
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

ActiveX und OLE

- Anforderung
  - In dokumentenbasierten Anwendungen besteht der Wunsch, Bilder, Tabellen usw. per »drag and drop« in andere Dokumente zu übernehmen.
- OLE-Architektur
  - Dokument-Server: Kann Inhalte verwalten und darstellen.
  - Dokument-Container: Kann Dokumenten-Server aufnehmen.
  - Eingebettete Dokumente sollen dort bearbeitet werden, wo sie im umgebenen Dokument dargestellt werden (in-place activation).
  - Wird einem eingebetteten Dokumenten-Server mitgeteilt, dass der dargestellte Inhalt verändert werden soll, so wird – unsichtbar für den Benutzer – vom Dokument-Server über dem eigentlichen Inhalt ein neues Fenster geöffnet, in dem der Inhalt verändert werden kann.
  - Es entsteht die Illusion, dass der Inhalt des eingebetteten Dokuments direkt im umgebenden Dokument verändert werden kann.
  - Zusätzlich wird der Inhalt und/oder die Semantik einiger Menüs und Werkzeugleisten ausgetauscht.

Relevante Schnittstellen für OLE

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



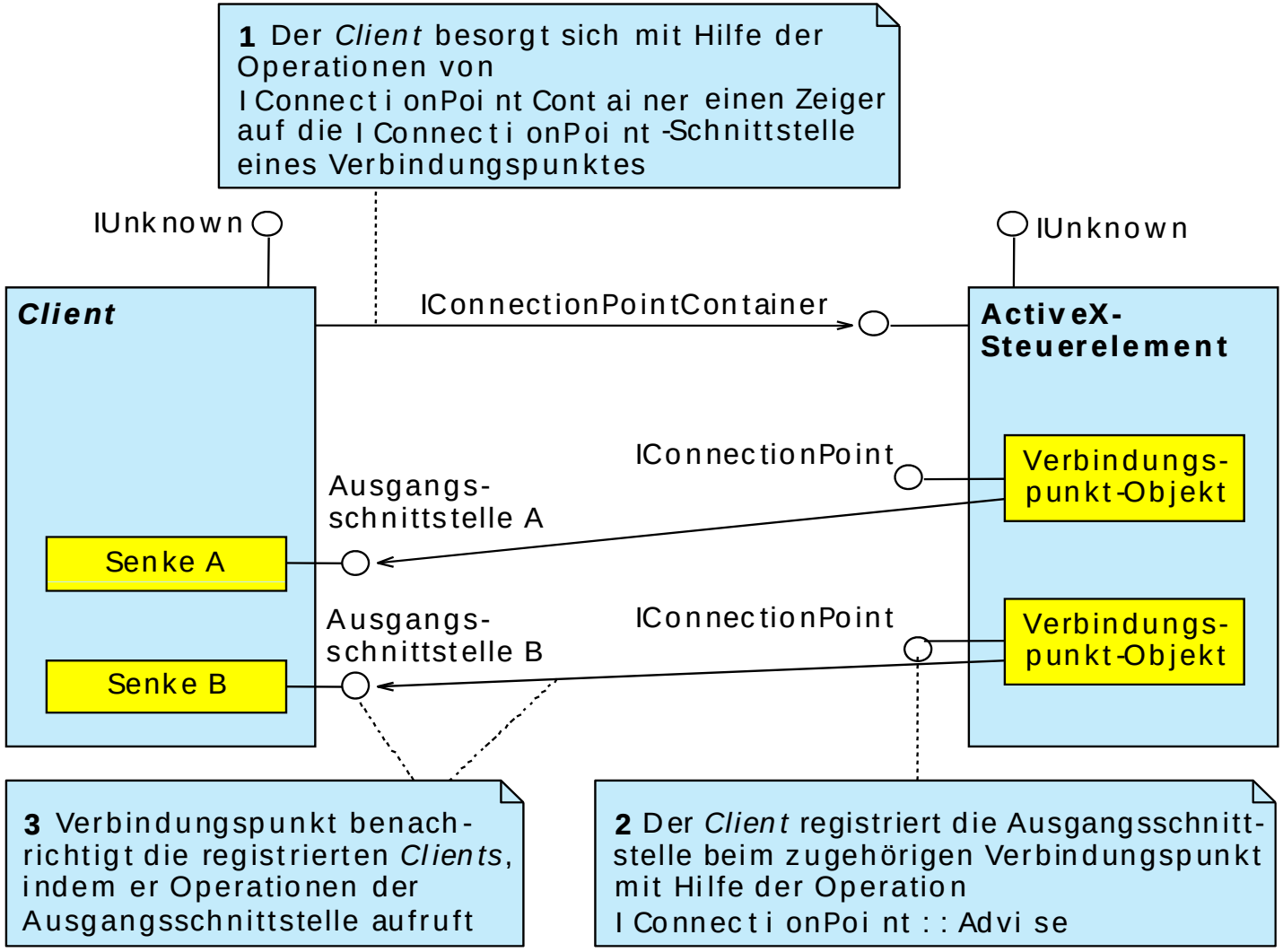
ActiveX

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Selbstregistrierung
  - Jede COM-Komponente ist ein ActiveX-Steuerelement, wenn sie zusätzlich noch die Fähigkeit zur Selbstregistrierung unterstützt.
  - Wird eine Komponente auf einer Web-Seite platziert, dann soll ihre Installation automatisch und dauerhaft erfolgen.
  - ActiveX-Steuerelemente werden auf dem Client dauerhaft installiert (im Gegensatz zu Java).

Client und verbindungsfähiges Objekt

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zusammenfassung

Bei der Entwicklung von Software ist immer zu prüfen, ob eine bestimmte Teilfunktionalität **Halbfabrikate** bzw. **Komponenten** (components) eingesetzt werden können. Bestimmte Anwendungsgebiete erlauben bereits eine komponentenbasierte Softwareentwicklung unter fast ausschließlicher Verwendung von Halbfabrikaten.

Unterstützt eine Komponente die Fähigkeit, Eigenschaften, Operationen, Ereignisse und andere Informationen nach außen zur Verfügung zu stellen, so sagt man, die Komponente ist **introspektiv** oder auch, die Komponente unterstützt die Eigenschaft der Introspektion.

Die Komponentenmodelle, **JavaBeans** (Sun) und **Component Object Model** (Microsoft) unterstützen eine komponentenbasierte Software-Entwicklung.

Eine **JavaBean** muss in der Programmiersprache Java implementiert werden und bestimmte Namenskonventionen berücksichtigen.

COM-Komponenten sind programmiersprachenunabhängige Halbfabrikate, die das Component Object Model auf binärer Ebene spezifiziert.

Unter dem Schlagwort **ActiveX** werden von Microsoft einige Konzepte zusammengefasst, die auf COM basieren. Unter anderem das Erstellen von Verbunddokumenten, was mit **Object Linking and Embedding** (OLE) bezeichnet wird, und die ActiveX-Steuerelemente.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Literatur

- Balzert H., Lehrbuch der Softwaretechnik, 2. Auflage, Spektrum Akademischer Verlag,Heidelberg, 2000.

- Eddon G., Eddon H., Inside Distribuded COM, Entdecken Sie die Programmierung von verteilten Applikationen, MS Press, 1998
- Gamma E., Helm R. Johnson R., Vlissides J., Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995
- Griffel F., Componentware, Konzepte und Techniken eines Softwareparadigmas, dpunkt-Verlag, Heidelberg, 1998
- Szyperski C., Component Software, Beyond Object-Oriented Programming, Addison Wesley, 1997
- Chappel D., Understanding ActiveX and OLE, A guide for developers & Managers, MS Press, Redmond, 1996
- Sun Microsystems: JavaBeans API Specification, http://java.sun.com/beans/1997