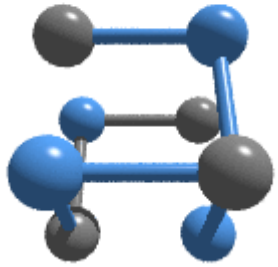


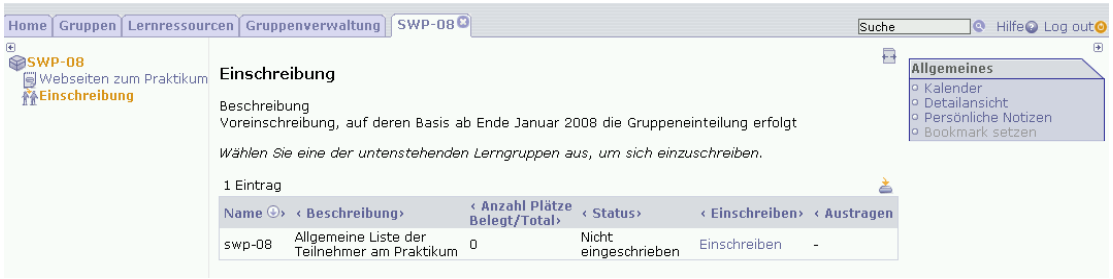
3. Übung Softwaretechnik
- Entwurf / Design -
Wintersemester 2007/2008

Thorsten Berger, Thomas Riechert

Organisatorisches

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Voranmeldung Softwaretechnik-Praktikum im Sommersemester 2008
  https://olat.informatik.uni-leipzig.de:9101/olat/auth/repo/go?rid=11403282



- Auswertung der bisherigen Teilnahme an den angebotenen Online Übungen
- Klausurtermin: voraussichtlich 12.2., 9:00 Uhr
- Lehrveranstaltungsevaluation
  - Zugangsinformationen für Studierende
  - Kennung: kpf-swt-07
  - Passwort: tjwqc630
  - URL: https://www.umfragen.uni-bonn.de/leipzig/lehre

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Agenda

- **1. Teil, Präsenz:**
  - Überblick Software-Architekturen (T. Berger)
- **2. Teil, Gruppenarbeit:**
  - Aufgaben 1 bis 5

Software-Architektur

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- *„eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie Beschreibung ihrer Beziehungen“* (Balzert00, S. 716)
- (funktionale, nichtfunktionale und Qualitäts-) Anforderungen aus Lasten-/Pflichtenheft verbinden mit Architektur-/Design-Mustern und Technologien
- Architektur-Sichten (Kruchten95):

Architektur-Sicht	Charakteristik
Logische Sicht (Logical view)	beinhaltet funktionale Anforderungen der Plattform
Prozess-Sicht (Process view)	beschreibt Verhalten bzw. Abläufe im System zur Laufzeit (Prozesse, Threads und ihre Interaktionen)
Entwicklungs-Sicht (Development view)	beschreibt Architektur als Komposition von Schichten, Subsystemen, Komponenten, Klassen und Schnittstellen
Physische Sicht (Physical view)	beschreibt die physische Verteilung von in den vorherigen Sichten identifizierten Elementen auf Dateien, Verzeichnisse und Ausführungseinheiten (Knoten, Betriebssysteme)

Software-Architektur

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Grobe Architektur:
  - Grobgranulare Sicht auf die Software
  - Zerlegung des Gesamtsystems in Teilsysteme (Komponenten)
  - hauptsächlich durch nichtfunktionale und Qualitäts-Anforderungen beeinflusst (Entwicklungsfähigkeit, Wartbarkeit, Geschwindigkeit, Sicherheit)
  - Architekturmuster:
    - Verteiltes System: Client/Server, Broker, Vermittler
    - Schichten (Layer)
    - Pipes and Filters
    - Interaktionssystem: Model-View-Controller (MVC), Model-View-Presenter (MVP), Rich-Client
    - Adaptives System: Mikrokernel
    - ...
  - Modellierung:
    - UML-Diagramme: z.B. Verteilungsdiagramm, Komponentendiagramm, Paketdiagramm, Interaktionsübersichtsdiagramm
    - Schichten-Diagramm
    - n-Tier-Modell von SUN
    - Architekturbeschreibungssprachen (ADL)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Detaillierte Architektur:
  - Feingranulare Sicht auf die Software
  - Zerlegung der Teilsysteme in Komponenten
  - stärker von funktionalen Anforderungen beeinflusst
  - Design-Muster (Design Patterns, Gamma95):
    - Factories
    - Singleton
    - Observer
    - Value Object (VO) / Data Transfer Object (DTO)
    - Data Access Object (DAO)
    - ...
  - Modellierung:
    - UML-Diagramme: z.B. Klassendiagramm, Sequenzdiagramm, Komponentendiagramm, Zustandsdiagramm, Timing-Diagramm
    - Architekturbeschreibungssprachen (z.B. xADL, Mae, ACME, SDL)

Software-Architektur

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Aktivitäten beim Architektur-Entwurf:
  - Abstraktion
  - Modellierung
  - Simulation
  - Prototyping
  - Validierung
- Wichtige Prinzipien
  - Trennung der Zuständigkeiten (Separation of Concerns, SoC)
    - Zerlegung des Systems in Teile mit hoher Kohäsion
    - Lose Kopplung der Teile
    - Problem: bei OOP nur eine Dekomposition möglich (sog. dominante Dekomposition), Überschneidungsproblem
  - Gesetz von Demeter (Law of Demeter, LoD)
    - Objekte kommunizieren nur mit direkt referenzierten Objekten
    - Verringerung Kopplung und Erhöhung Wartbarkeit
  - Service Abstraction
    - Zugriff auf Implementierung nur über Interfaces
    - Erhöhung Wartbarkeit und Entwicklungsfähigkeit
  - Nutzung von Frameworks

Komponenten-Architekturen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- *"Eine Software-Komponente ist ein Software-Element, das konform zu einem Komponentenmodell ist und gemäß einem Composition Standard ohne Änderungen mit anderen Komponenten verknüpft und ausgeführt werden kann.„* (Councill,Heinemann03)

- Komponenten-Standards und -Frameworks:
  - Microsoft: .NET, COM, DCOM, OLE
  - OMG: Corba Component Model (CCM)
  - SUN: JavaBeans, J2EE Enterprise Java Beans
  - Spring-Framework
  - Open Services Gateway initiave (OSGi)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

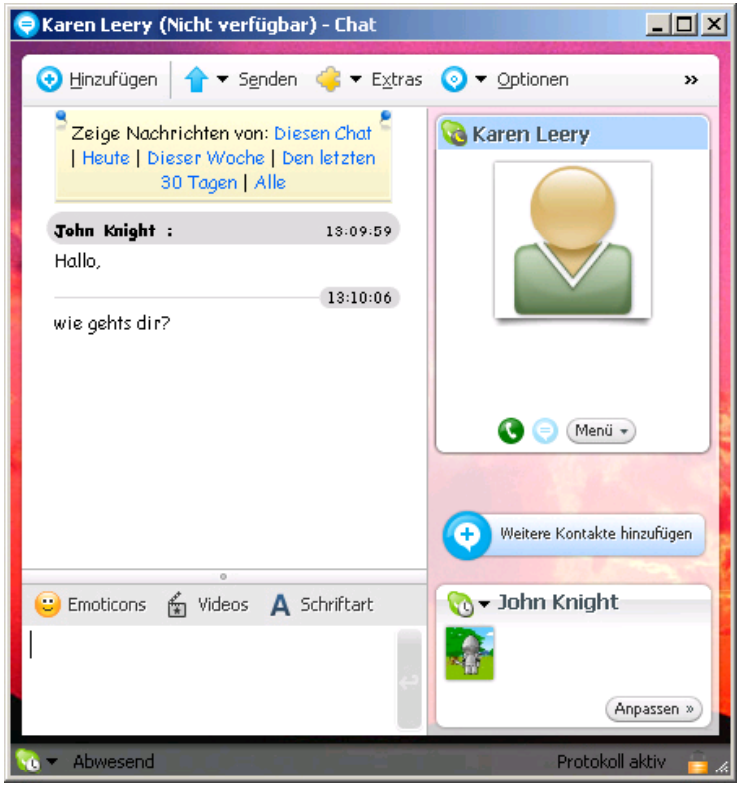
Agenda

- **1. Teil, Präsenz:**
  - Überblick Software-Architekturen (T. Berger)
- **2. Teil, Gruppenarbeit:**
  - Aufgaben 1 bis 5

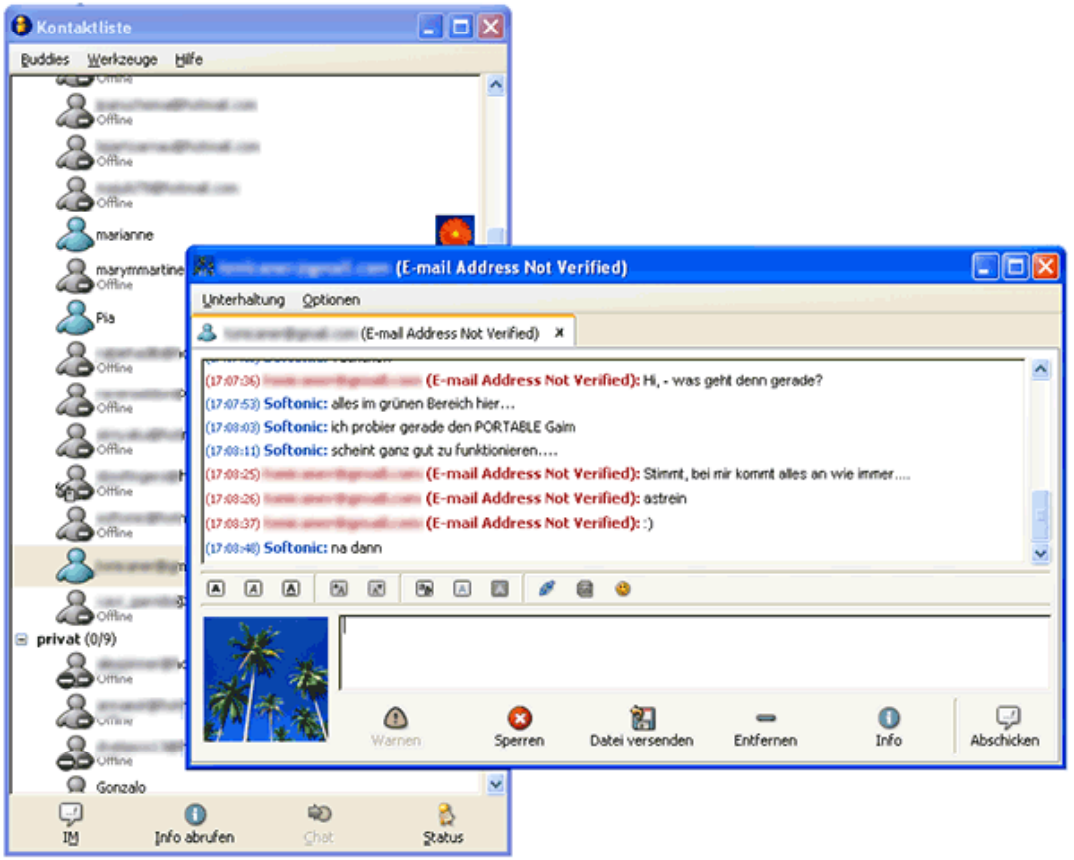
Aufgabe 1 / Software Ergonomie (20 min)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Beurteilen Sie einen von Ihnen benutzen Instant Messenger (z.B. Pidgin oder Skype) nach den Grundsätzen ergonomischer Dialoggestaltung (EN ISO 9241-10 und EN ISO 14915-1).



Weitere Screenshots:



PS - Quellen der GAIM Bilder:

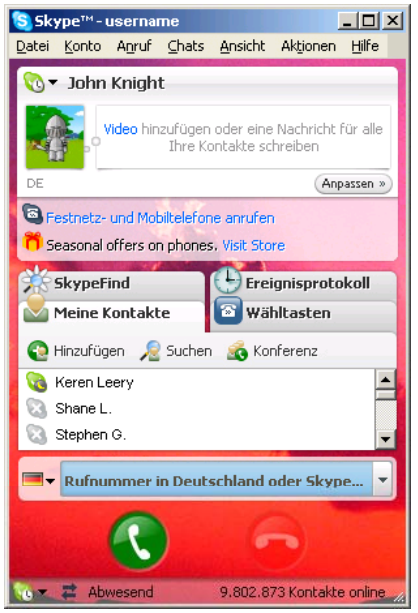
http://screenshots.softonic.com/s2de/50000/50264/0_portable_gaim_03.gif

http://pendriveapps.com/images/gaim.gif

- Wählen Sie für folgende Software-Applikationen geeignete Architekturen aus und skizzieren Sie Ihre Lösung und begründen Sie diese.

  - Web-Mailer
  - Mobiles Ticket-System (Bahn)
  - Instant Messenger

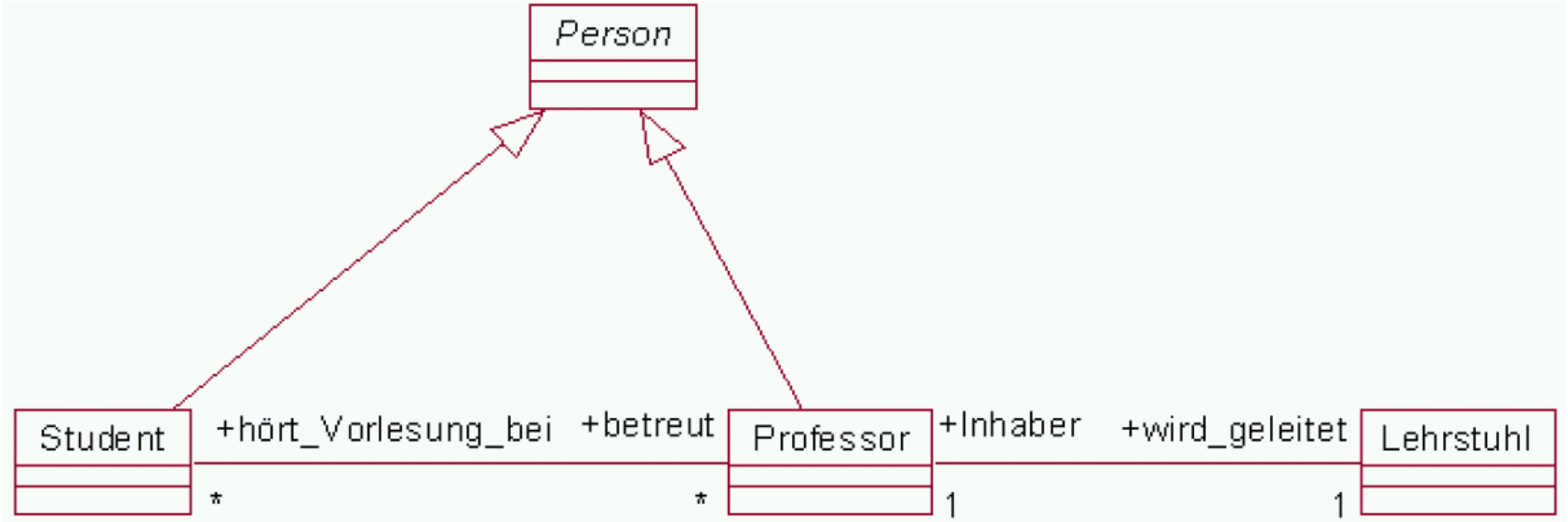
Quelle Ticketautomat: http://www.vwg.de/tickets.html



Aufgabe 3 / Entwurf (20 min)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Setzen Sie das nachfolgende OOA- Diagramm aus einer Lehrstuhlverwaltung in Java-Code um.



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Aufgabe 4 / Datenbanken (20 min)

Ein Student hat eine eindeutige Matrikel-Nr., einen Namen und ein Alter. Er muss sich für mehrere Prüfungsfächer anmelden. Zu jedem Prüfungsfach gibt es eine eindeutige Kursbezeichnung und einen Prüfer. Für ein Prüfungsfach melden sich in der Regel mehrere Studenten an. Für jeden Studenten wird pro Prüfungsfach die Anzahl der Versuche und die Note des letzten Versuchs gespeichert. Ein Student kann Mitglied einer Uni-Sportgruppe sein. In einer Sportgruppe sind mehrere Studenten. Für jeden Studenten wird gespeichert, seit wann er Mitglied in der gewählten Sportgruppe ist. Jede Sportgruppe besitzt eine eindeutige Sportart und einen Trainingsplan.

Erstellen Sie zunächst eine unnormalisierte Relation, wobei Sie Matrikel-Nr. als Schlüssel wählen. Geben sie dann die 1., 2., und 3. Normalform an!

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Aufgabe 5 / Design-Patterns (45 min)

- Diskutieren Sie folgende Design Patterns:
  - Factories
  - Singleton
  - Observable
  - MVC
- Beziehen Sie sich dabei auch auf die in Aufgabe 3 bereits diskutierten Applikationen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Arbeitsgruppen

I. Frau Bulka

II. Herr Frommhold

III. Herr Berger

IV. Herr Riechert

Quelle: http://www.klimatek.de/shop_content.php/coID/9/content/Klima%20und%20Gesundheit

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Agenda

- **Anhang**

Klassifikation von Entwurfmustern

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

		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

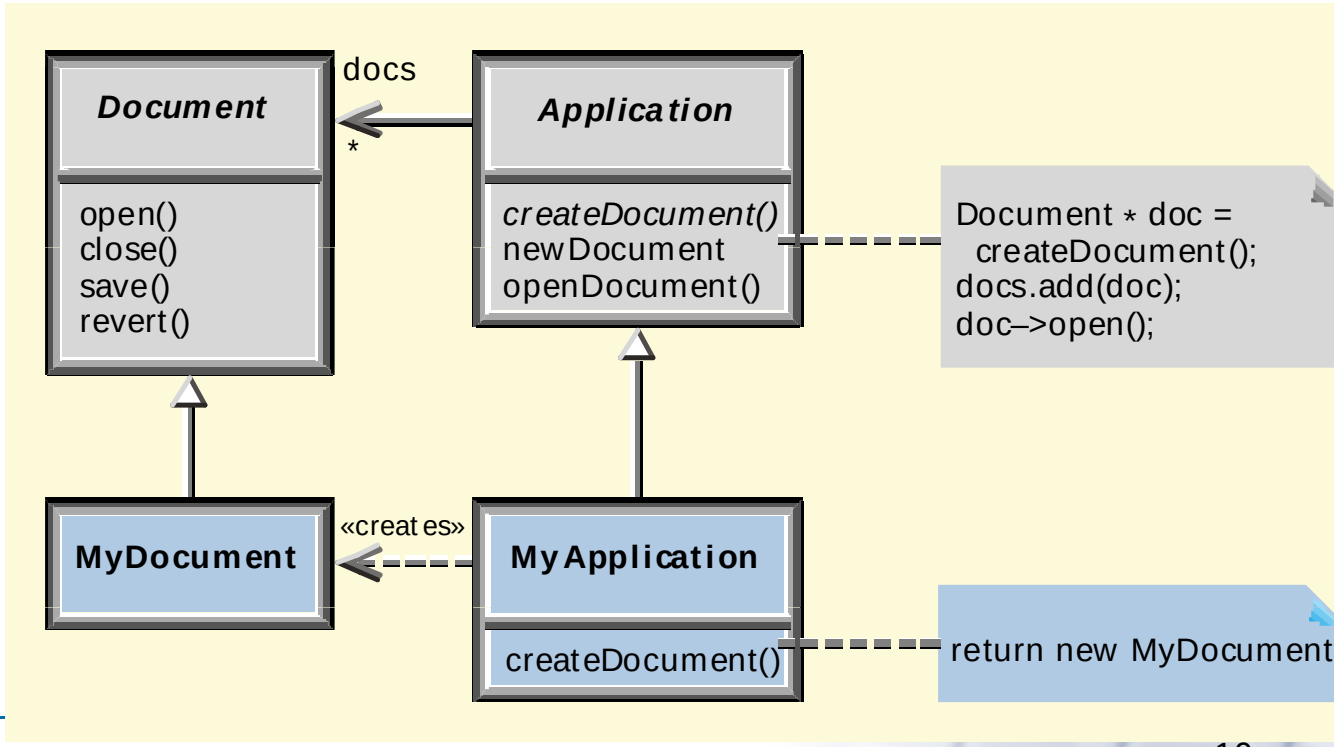
Gültigkeitsbereich:

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

Das Fabrikmethode-Muster

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

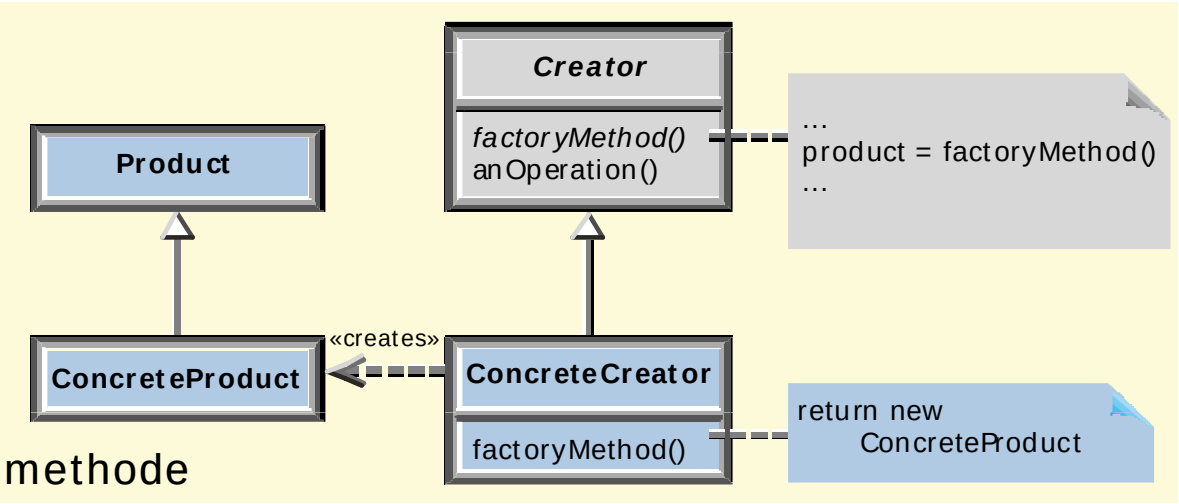
- Zweck
  - Klassenbasiertes Erzeugungsmuster
  - Bietet eine Schnittstelle zum Erzeugen eines Objekts an, wobei die Unterklassen entscheiden, von welcher Klasse das zu erzeugende Objekt ist
  - Auch bekannt als »virtueller Konstruktur« (virtual constructor)
- Motivation
  - Dieses Rahmenwerk eignet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



Das Fabrikmethode-Muster

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- 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

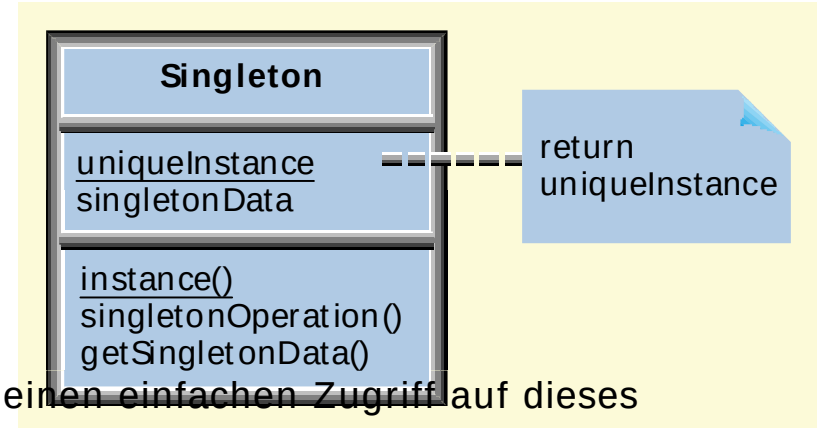


- Interaktionen
  - Der Creator verlässt sich darauf, dass Unterklassen die Fabrikmethode korrekt implementieren
- Konsequenzen
  - Fabrikmethoden verhindern es, dass anwendungsspezifische Klassen in den Code des Rahmenwerks eingebunden werden müssen.

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

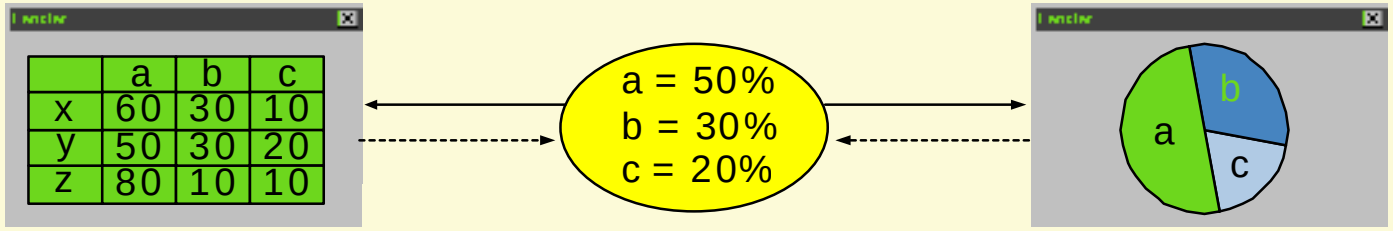
- Klasse Singleton definiert die Klassenoperation instance(), die es dem Client ermöglicht, auf das einzige Exemplar zuzugreifen

```
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

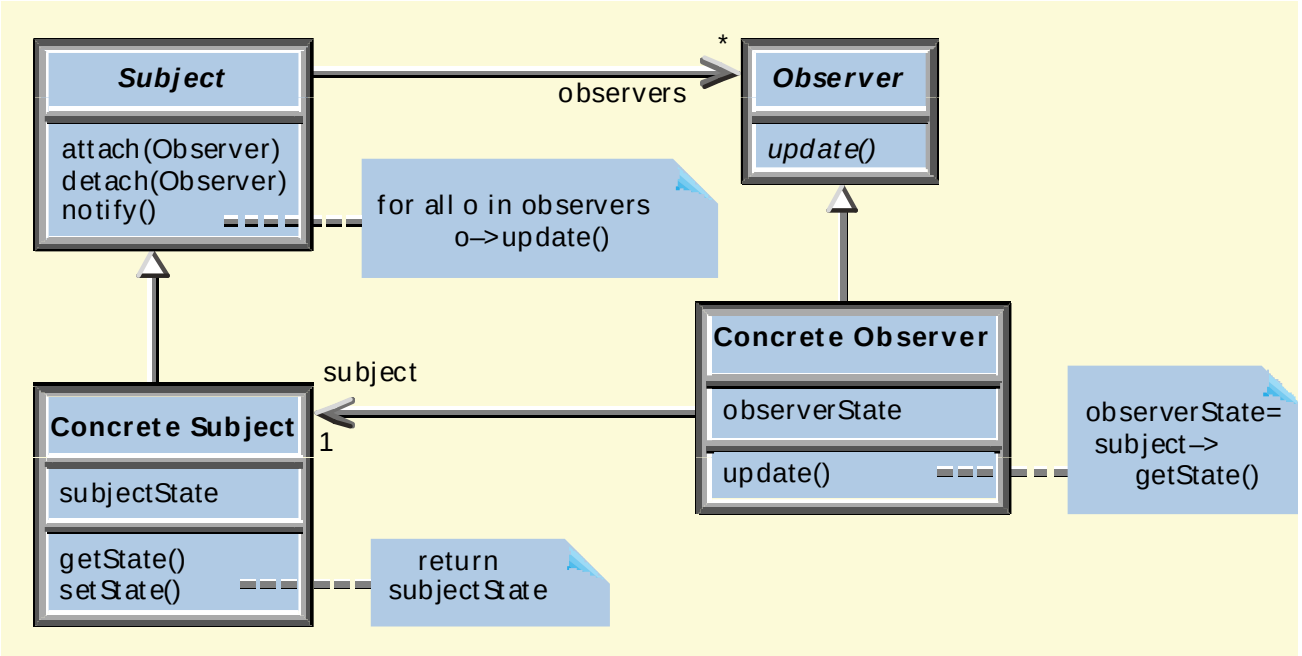
- Zweck
  - Objektbasiertes Verhaltensmuster
  - Es sorgt dafür, dass bei der Änderung eines Objekts alle davon abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.
- Motivation
  - Ein Objekt enthält Anwendungsdaten
  - Diese sollen auf verschiedene Arten angezeigt werden, z.B. als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



Das Beobachter-Muster

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

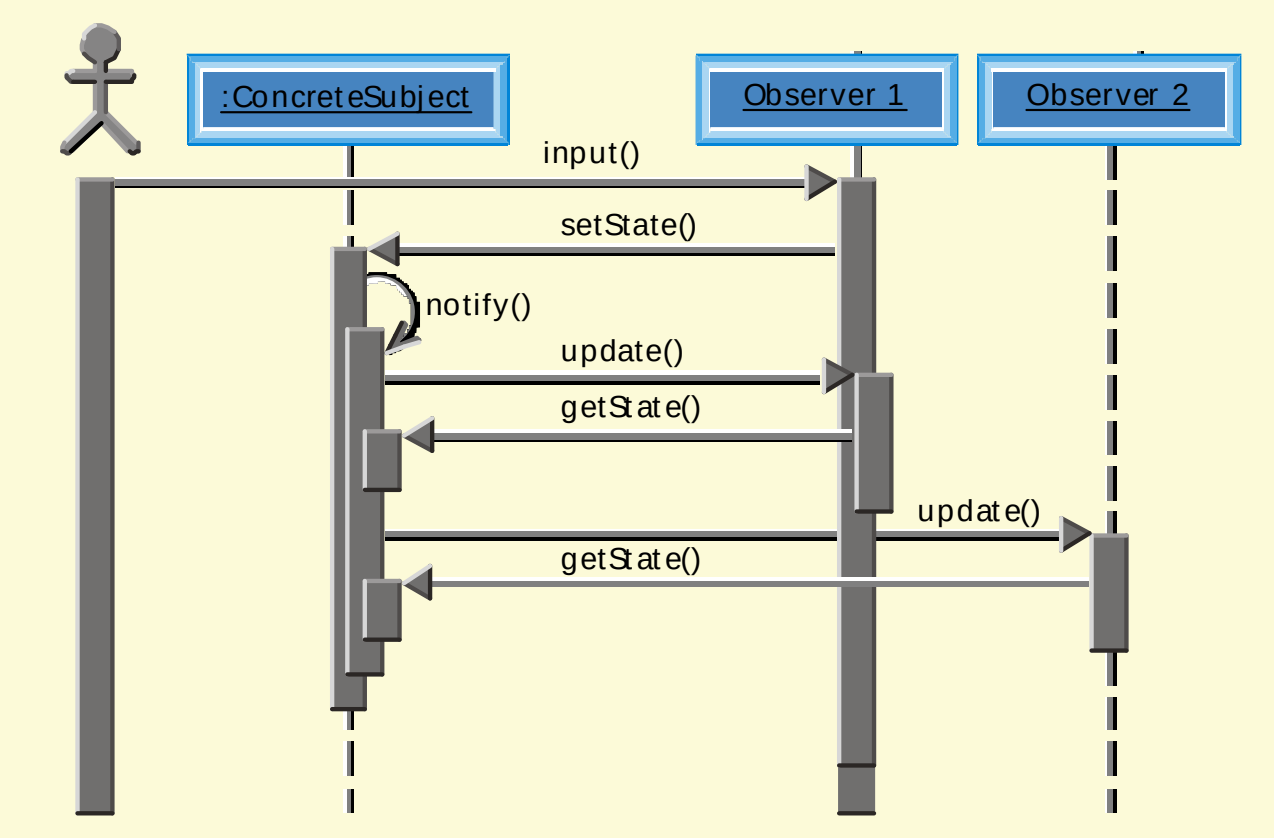
- Anwendbarkeit
  - Eine Abstraktion besitzt zwei Aspekte, die wechselseitig voneinander abhängen
  - Die Kapselung in zwei Objekte ermöglicht es, sie unabhängig voneinander wiederzuverwenden oder zu modifizieren
  - Die Änderung eines Objekts impliziert die Änderung anderer Objekte, und es ist nicht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



Das Beobachter-Muster

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- 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.