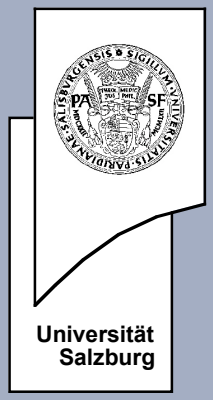


Framework Architectures

O.Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Pree

SOFTWARE RESEARCH LAB

Inhalt

- OO Framework Architectures
- Metainformationen als Basis für die dynamische Konfiguration von Softwaresystemen

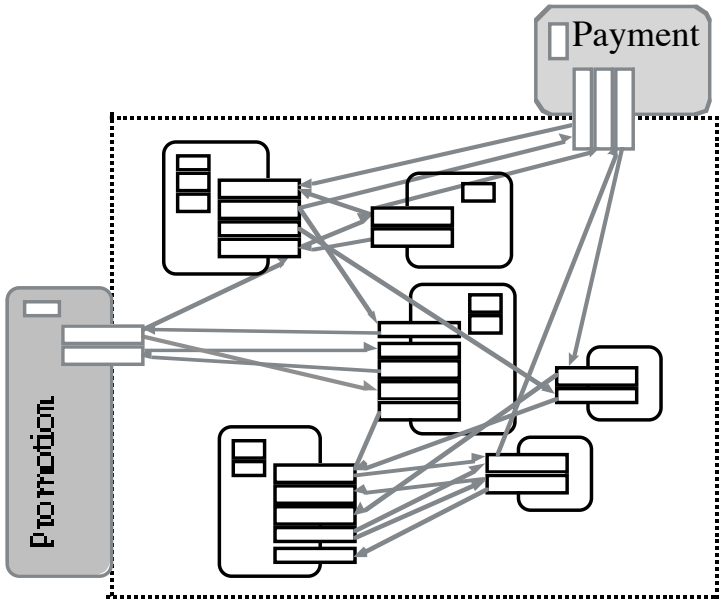
Frameworks
(= Product Lines
= Platforms
= Generic Software)

Frameworks allgemein

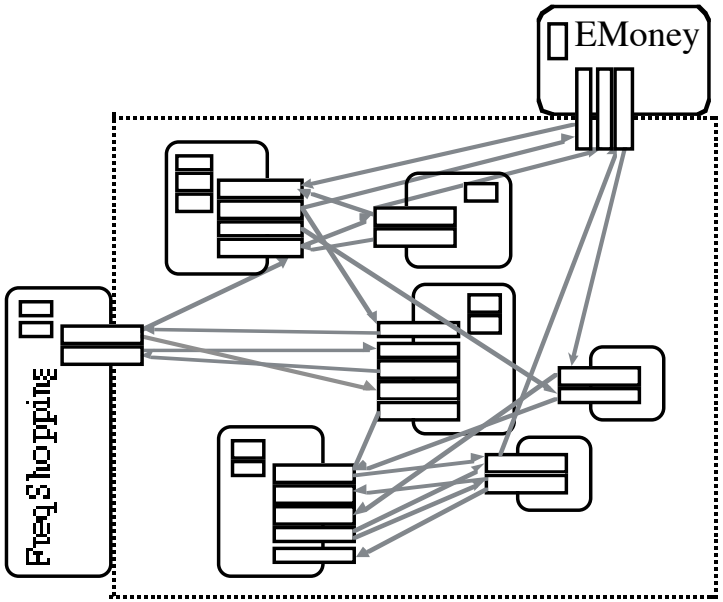
Beispiele für „Nicht-Software“-Frameworks:

- Küchenmaschine: durch Einstecken einer Komponente wird das vorhandene „Halbfertigfabrikat“ zum fertigen Mixer oder Fleischwolf
- neue Automodelle gleichen meist „im Kern“ (Chassis, Getriebe, Motorpalette) den Vorgängermodellen

Konfiguration durch Einstecken von SW-Komponenten



vor der Adaptierung



nach der Adaptierung

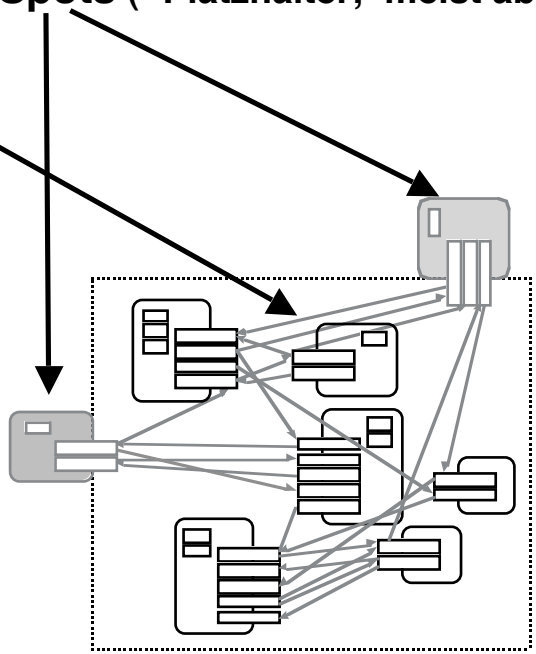
SOFTWARE RESEARCH LAB

Universität Salzburg

Frameworks— Definition & Dynamische Sicht

Framework:= a piece of software that is extensible through the callback style of programming

OO Frameworks: **Komponenten**
+ Interaktion + Hot Spots (=Platzhalter; meist abstrakte Klassen oder Schnittstellen)



Black-Box versus White-Box Framework-Komponenten

- **Black-Box:** Wiederverwendung ohne jegliche Anpassung: **„Plug & Work“**
- **White-Box:** Anpassung durch **Unterklassenbildung** erforderlich
- **Je reifer** ein Framework ist, **umso mehr Black-Box-Komponenten** sind enthalten.

Abstrakte Klassen und Schnittstellen

Abstrakte Klassen standardisieren die Schnittstelle (den „Stecker“) für Unterklassen.

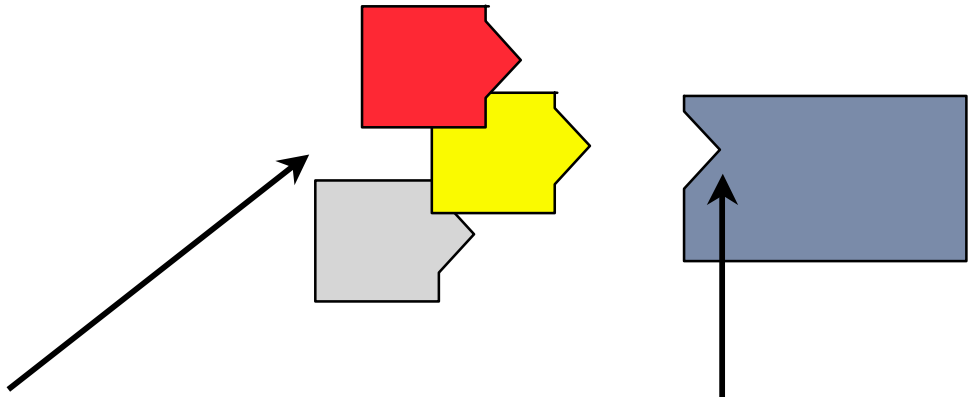
Aufgrund der dynamischen Bindung **können andere Systemteile/Komponenten nur aufgrund des Protokolls (= angebotene Methoden) von abstrakten Klassen bereits implementiert werden.**

Somit können Halbfertigfabrikate (=Frameworks) softwaretechnisch elegant entwickelt werden.

Anlaoges gilt für Schnittstellen (Interfaces).

Polymorphismus

Sogenannte Objekttypen sind **poly** (= viel) **morph** (= Gestalt). Anschaulich ist das mit **„Steckerkompatibilität“** vergleichbar:



Objekte, die zu diesem Stecker kompatibel sind

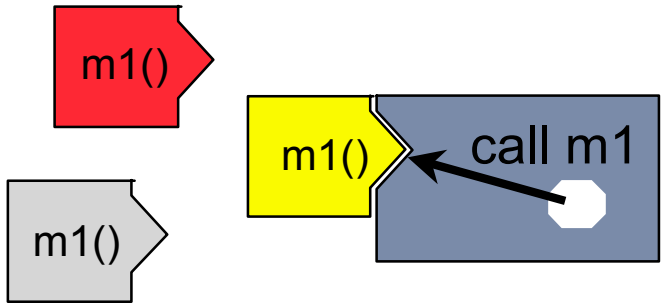
„Stecker“-Standard

SOFTWARE RESEARCH LAB

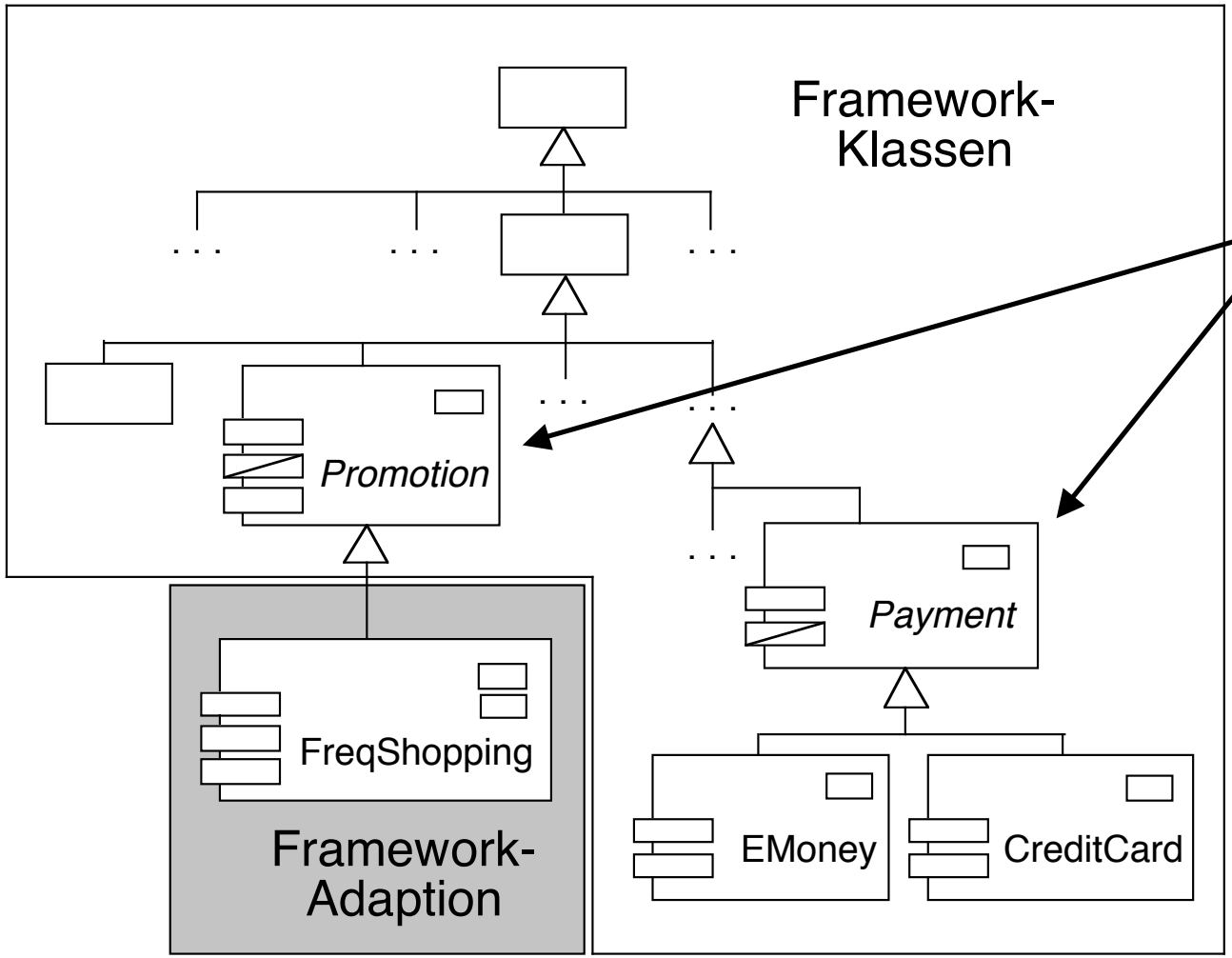
Universität Salzburg

Dynamische Bindung

Dynamische Bindung heißt, daß es **vom eingesteckten Objekt abhängt, welche Methode tatsächlich ausgeführt wird.** Das gelbe Objekt implementiert m1() zB anders als das rote Objekt:



Frameworks—Statische Sicht

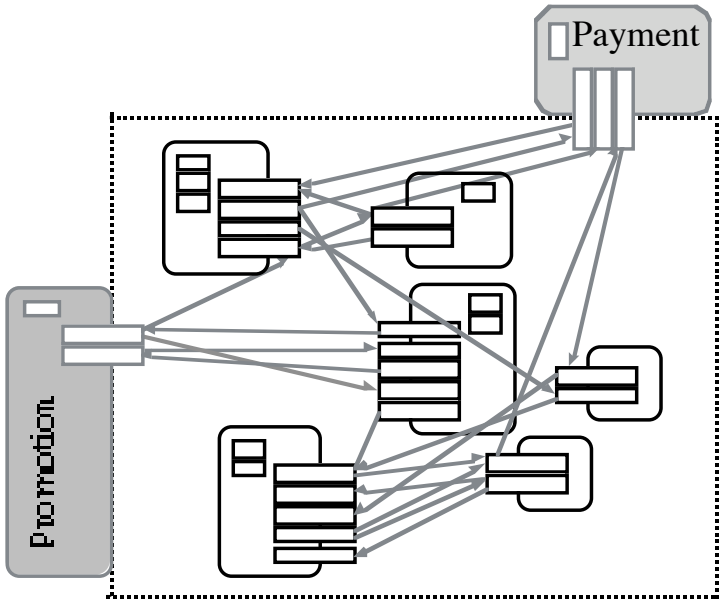


Schnittstellen/ abstrakte Klassen (hier mit jeweils einer abstrakten Methode)

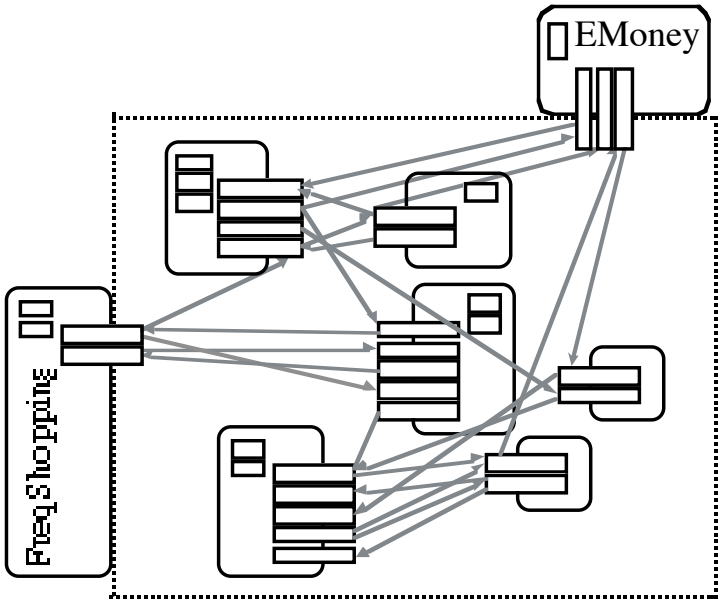
SOFTWARE RESEARCH LAB

Universität Salzburg

Konfiguration durch Einstecken von SW-Komponenten



vor der Adaptierung

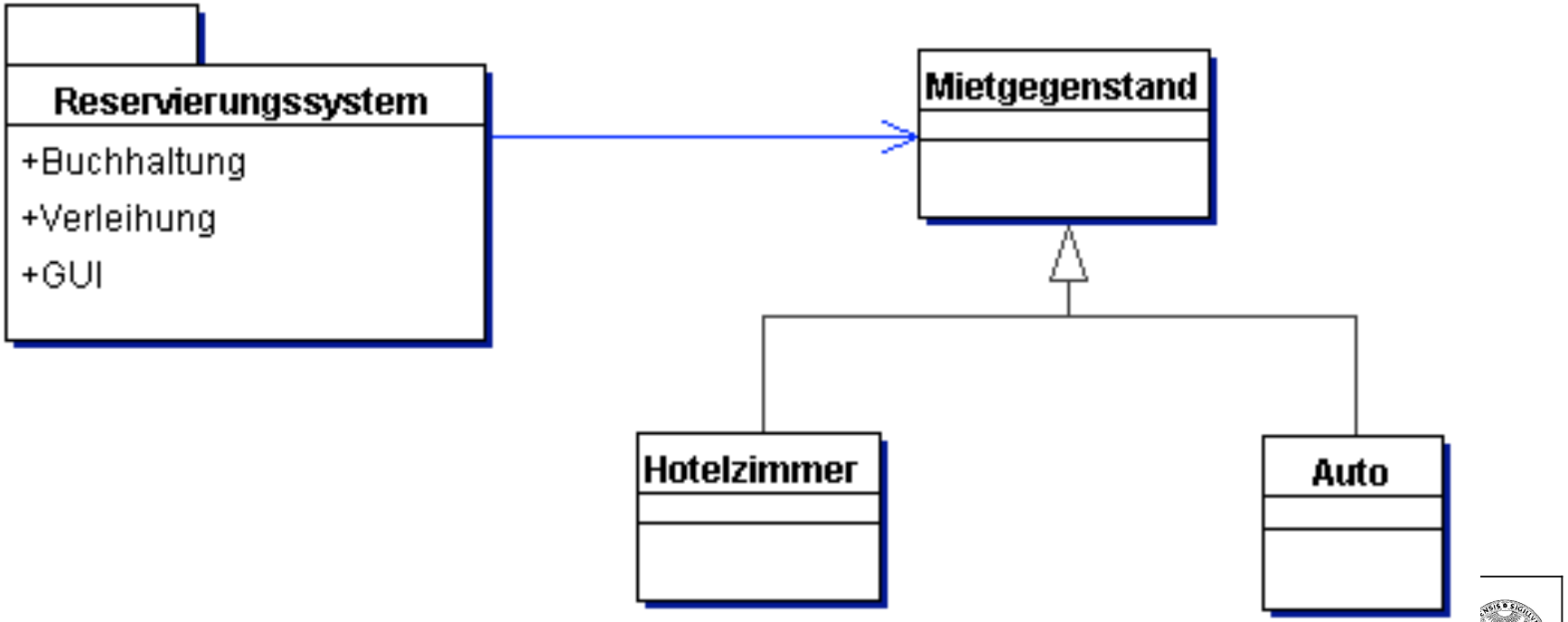


nach der Adaptierung

SOFTWARE RESEARCH LAB

Universität Salzburg

weiteres Beispiel: Hotelreservierung



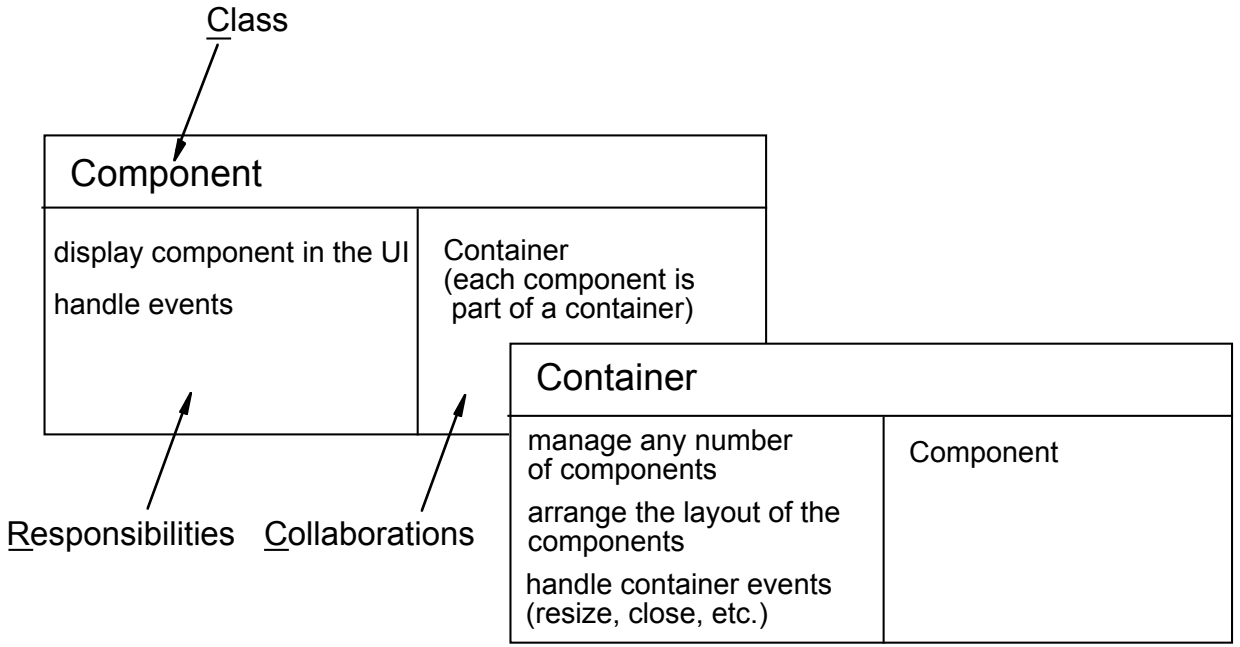
Swing-Framework?

Wie bilden die beiden abstrakten Klassen

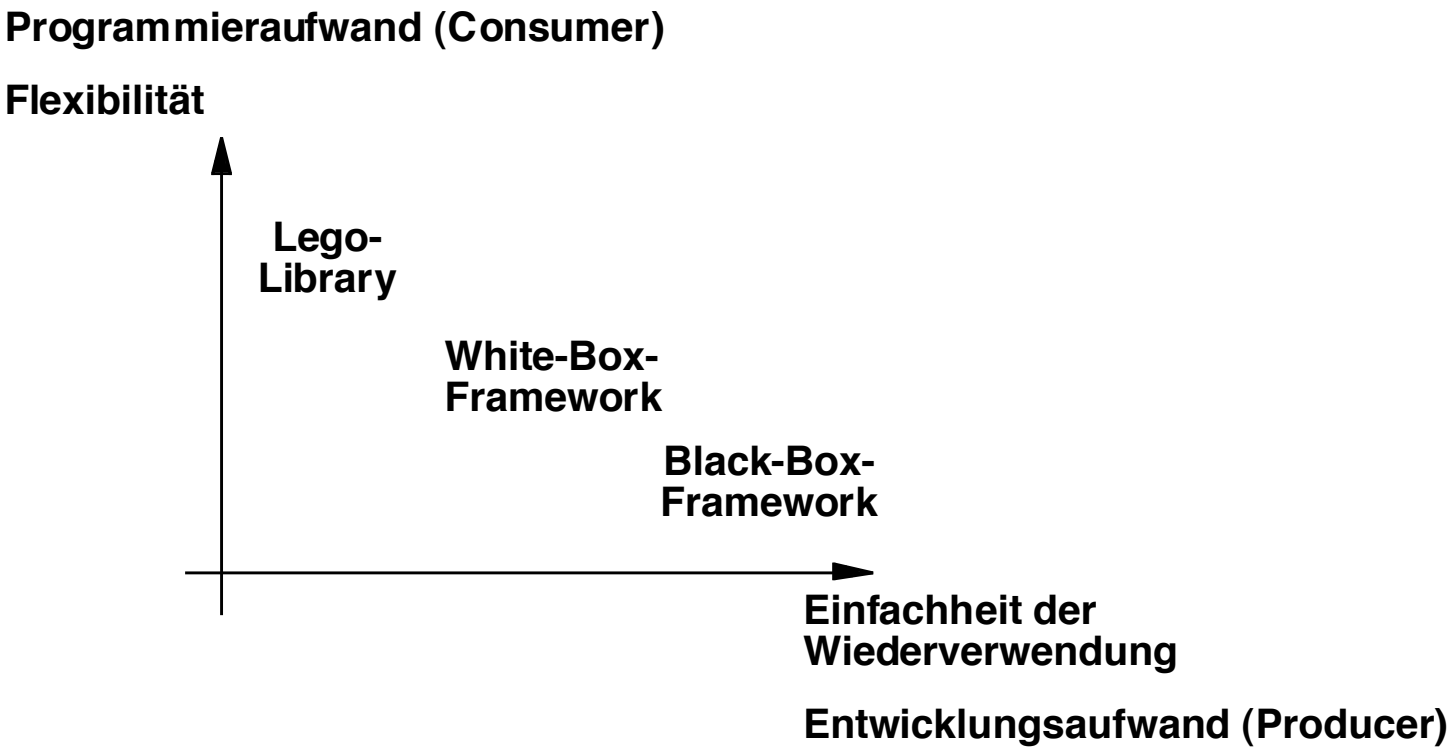
Component

Container

in Swing ein Framework?

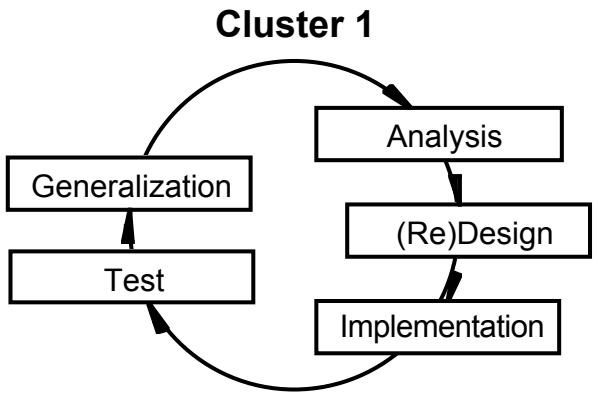


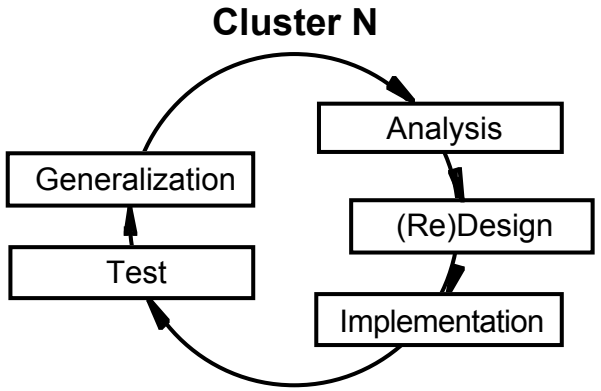
Legokasten/WB-/BB-Framework

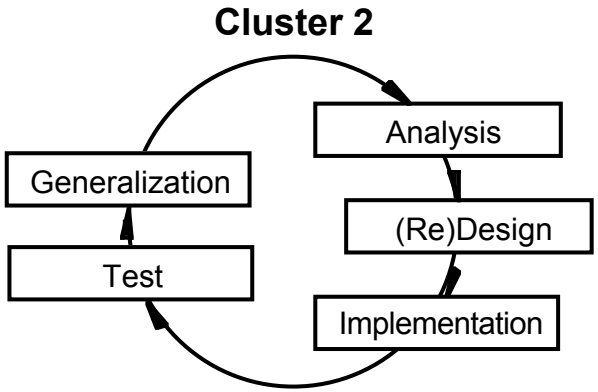


(Darstellung adaptiert aus dem Tutorial von Erich Gamma, OOP'96)

OO:
Cluster-Modell (Bertrand Meyer)



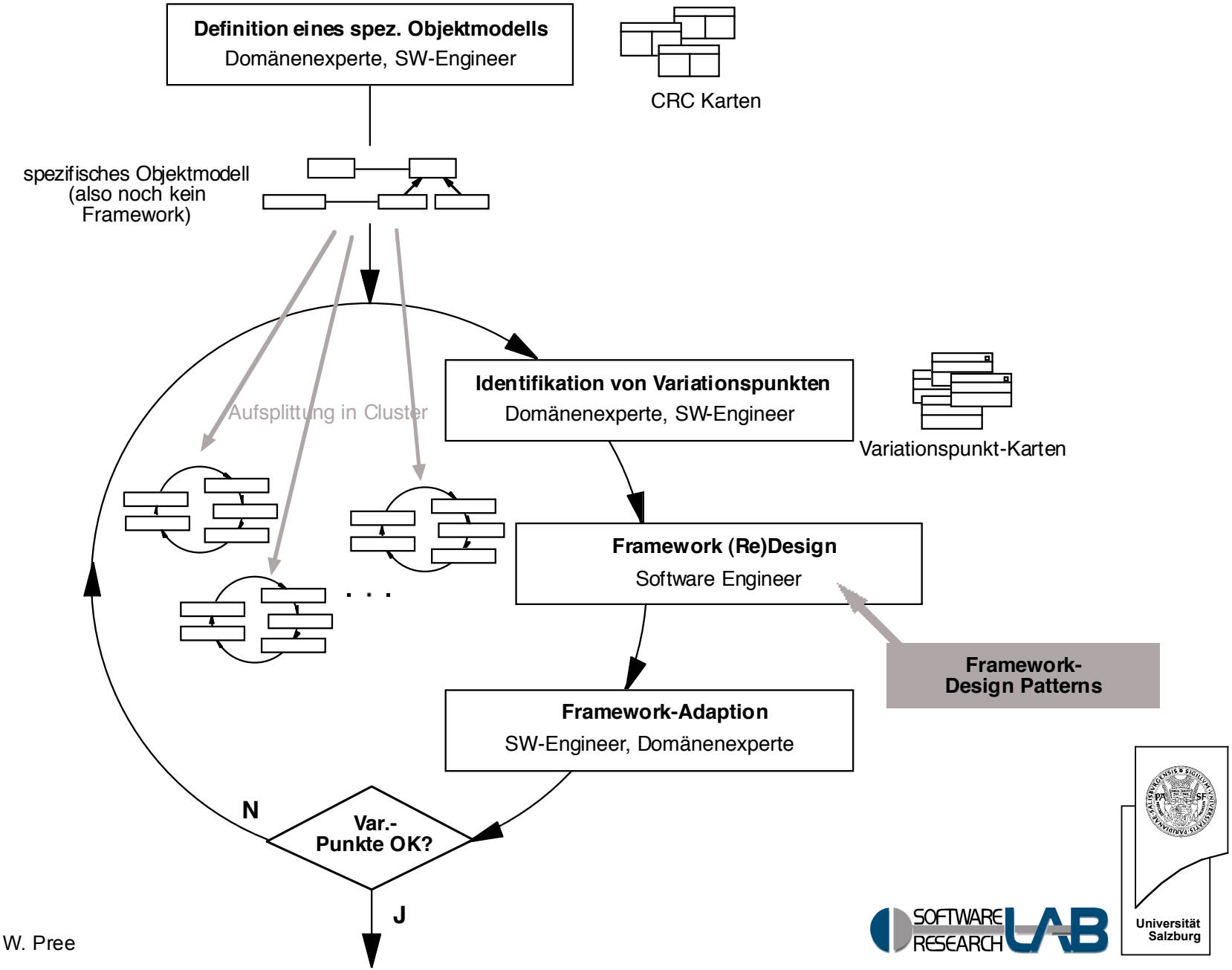




. . .

zB Design und Implementierung können sich in verschiedenen Clustern überlappen

erweitertes Cluster-Modell (B. Meyer) bei Framework-Entwicklung



Dynamische Konfiguration von Frameworks

- White-Box :: Black-Box Plug&Work
- Meta-Information

Fallstudie
Rounding Policy

Rounding Policy

Kontext: Wir nehmen an, es sei eine Klasse CurrencyConverter mit folgenden Eigenschaften bereits implementiert:

Der CurrencyConverter speichert intern eine Hauptwährung (zB US$) und eine Umrechnungstabelle:

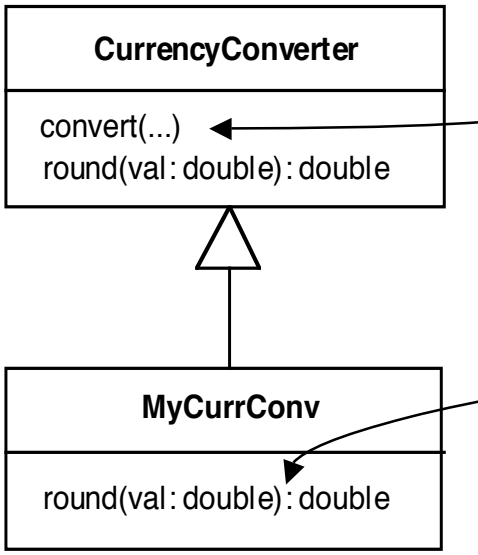
1 US$=	Valuten (sell):	0.87	Valuten (buy):	0.92
	Devisen:	0.90	Devisen:	0.91
	Euro			

...

Flexibilitätsanforderung: Die Art, wie gerundet wird, soll flexibel einstellbar sein (auf 1, 10, 100, etc.; nur aufrunden; bestimmte Grenzwerte, etc.).

Man überlege sich zwei Entwürfe, wie diese Flexibilität erreicht werden kann.

Entwurf I (nur WB-Konfig.)



template

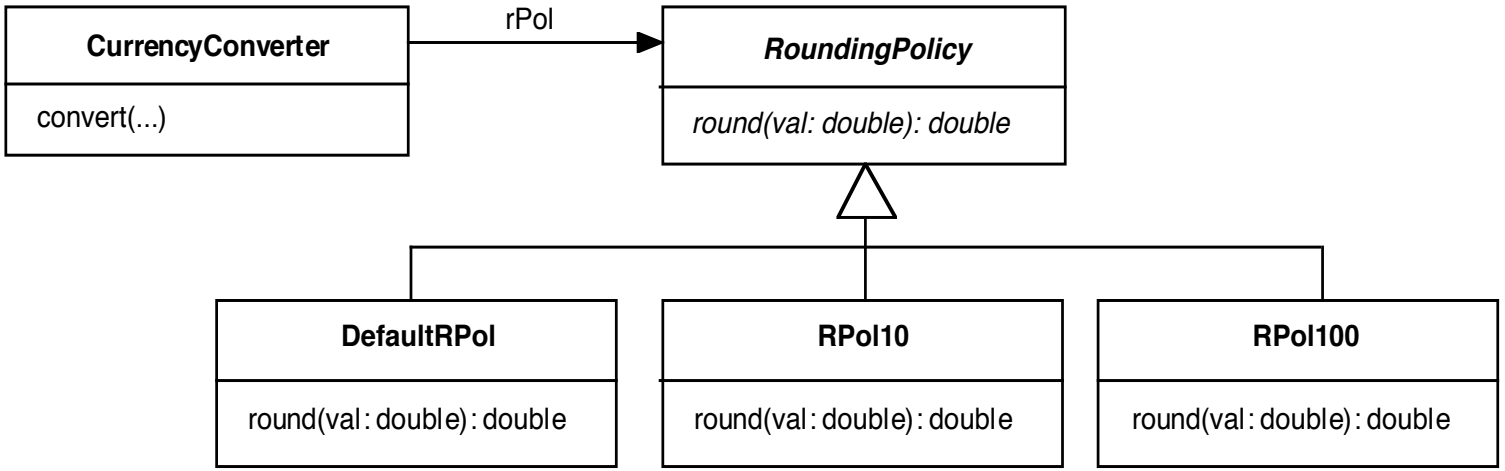
hook

```
public class CurrencyConverter {
   ...
   public void convert(...) {                    // template method
                  double result, value;
                  ...
                  switch (...) {
                     case ... :
                        ...
                        result= round(value);
                        break;
                     case ... :
                        ...
                        result= round(value);
                     ...
                  }
                  ...
   }
   public double round(double val) {             // hook method
                  ...
   }
   ...
}
```


SOFTWARE RESEARCH LAB

Universität Salzburg

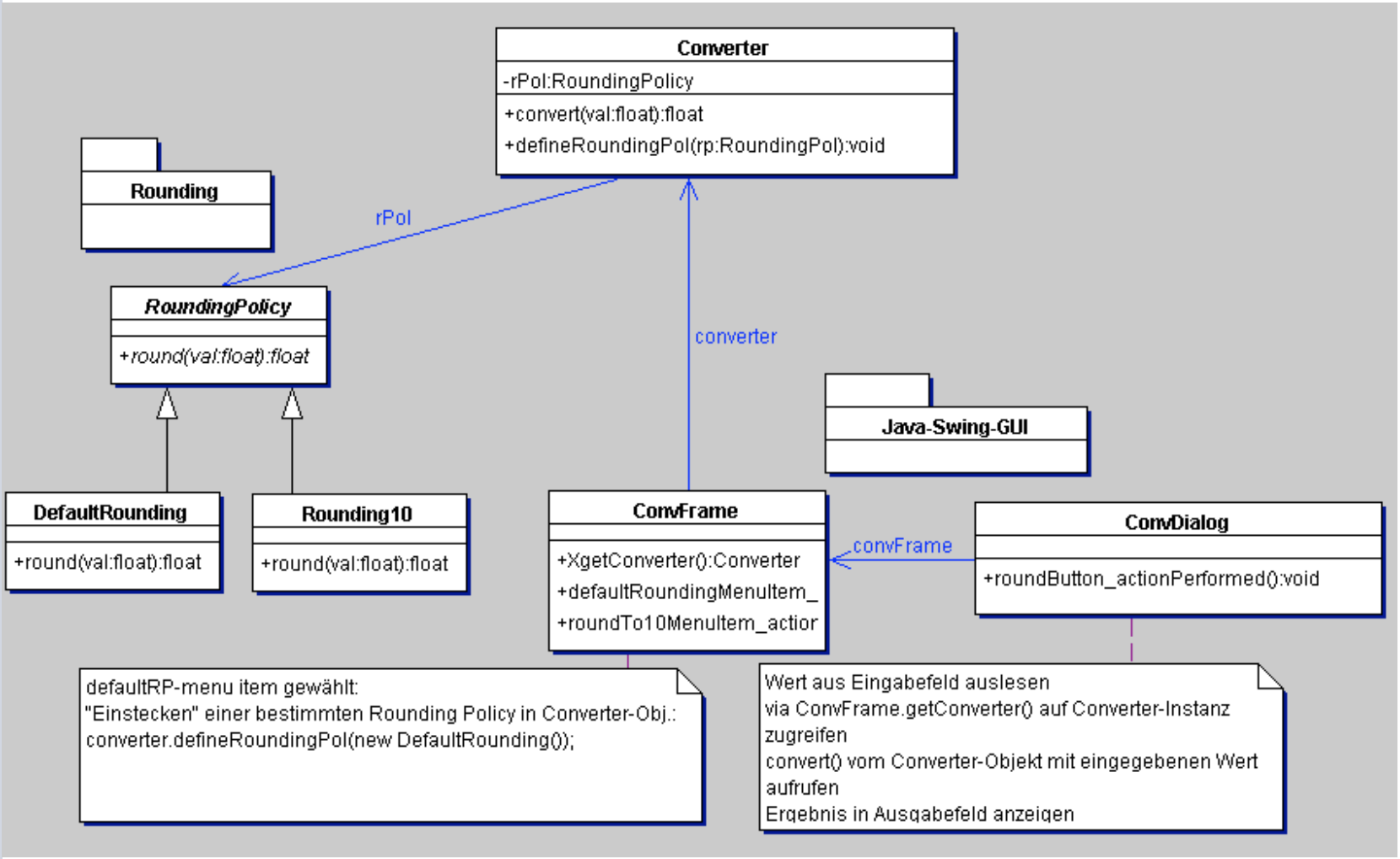
Entwurf II (BB&WB-Konfig.)




```
public void defineRoundingPolicy (RoundingPolicy rp) {
        rPol= rp;
}

public void convert(...) {
        double result, value;
        ...
        result= rPol.round(value);
        ...
}
```

Fachliche Klassen + GUI



Dynamische Erweiterung

Wie kann auf Basis des Entwurfs II eine Erweiterung um neue RoundigPolicy-Klassen zur Laufzeit erfolgen?

Überlegen Sie eine mögliche Benutzerschnittstelle und eine entsprechende Dynamic-Computing-Realisierung (siehe nachfolgender Abschnitt).

Metaklassen als Basis für Dynamic Computing

SOFTWARE RESEARCH LAB

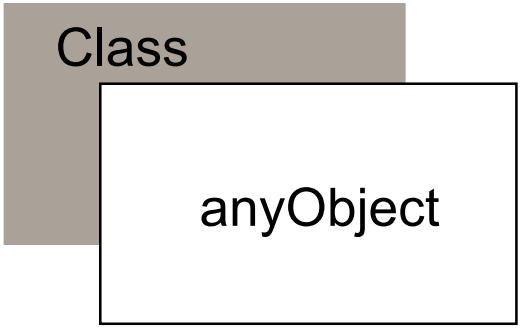
Universität Salzburg

Einführung

- Metaklassen representieren Klassen in der aktuellen JVM sowie in .NET.
- Mit Metaklassen kann man Typen und Elemente einer Klasse (Methoden, ...) zur Laufzeit abfragen. Darüberhinaus kann man eine Klasse, deren Name als String vorliegt, instanzieren, eine Methode aufrufen und Arrays modifzieren (vergrößern, verkleinern).

Klassenobjekte (I)

Jedes Objekt hat gleichsam ein **„Schattenobjekt“**, das eine Instanz der **Klasse Class** ist. Dieses Schattenobjekt kann Fragen über das eigentliche Objekt beantworten (zB welche Instanzvariablen es hat).



Klassenobjekte (II)

Ein Klassenobjekt, also das Schattenobjekt, kann man auf verschiedene Art und Weise erhalten. (Nachfolgende Source-Code-Beispiele basieren auf java.reflect.*)

```
Class c = mystery.getClass();
```

```
TextField t = new TextField();
Class c = t.getClass();
Class s = c.getSuperclass();
```

```
Class c = Class.forName(strg);
```

Metainformationen (I)

Beispiel: Eine Klasse, deren Namen nicht von vorneherein bekannt ist, muß instanziert werden.

nicht möglich:

```
String clName= "A";
Object anA= new clName;
```

Metainformationen (II)

Lösung mittels Klasse Class:

```
String clName= "A";
Object anA;
try {
        Class aClass= Class.forName(clName);
        anA= aClass.newInstance();
} catch (Exception e) {
        System.err.print(e);
}
```

Wozu *Dynamic Computing*?— Das Problem “FatWare”

Zahlreiche Programme sind Repräsentanten von ***Fatware*** (Niklaus Wirth, ETH Zürich), zB bei Desktop-Programmen:

- Spreadsheets mit >> 1000 Funktionen
- Office-Applikationen belegen “zig” MBs auf einer Hard-Disk und sehr viel im RAM
- HW wird langsamer schneller als Software langsamer wird
  (M. Reiser, IBM Zürich)

Dynamic Computing

stattdessen:

- Benutzer arbeitet mit **Basisversionen von Software**.
- **zusätzlich benötigte Komponenten** werden **bei Bedarf** ergänzt, indem sie **nachgeladen und dynamisch eingesteckt** werden
- Eine weitere Möglichkeit bieten **Push-Technologien zum Broadcasting von Updates**. Applikationen sind lokal bei den Clients gespeichert. Updates kommen von der Komponentensendestelle.