

4. Programmbausteine, Bibliotheken und Programmgerüste

Konzept **Programmbaustein**:

Programme aus **vorgefertigten Teilen** zusammensetzen - statt ganz neu schreiben
„Baukasten“, industrielle Produktion

Gute Bausteine, Programmgerüste: geplante Wiederverwendung, kostbar, aufwändig

Wichtige Fragen:

- Wie stellt man gute Bausteine her?
- Wie allgemein bzw. anpassbar sind die Bausteine?
- Sind die Bausteine direkt einsetzbar oder realisieren sie nur Teilaspekte und müssen vor der Anwendung komplettiert werden?
- Sind die Bausteine unabhängig voneinander oder wechselseitig aufeinander angewiesen?
- Wie werden Bausteine zusammengesetzt?

nach [Arnd Poetzsch-Heffter: Konzepte Objektorientierter Programmierung, Springer, 2000]

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 401

Ziele:

Wiederverwendbare Bausteine

in der Vorlesung:

Begriffe erläutern

nachlesen:

Poetzsch-Heffter, Kap.4

Beziehungen zwischen Programmbausteinen

Unabhängige Bausteine

Schnittstelle verwendet **nur vom Baustein deklarierte Typen** oder Standardtypen (Grundtypen und die aus java.lang).

Beispiele:

Datentypen

- Listen
- Mengen
- Hashtabellen

Typisch:

Elementanpassung durch generische Typparameter (z. B. in C++: Leda, STL)

lokal verstehen, einfach

isolierte Aufgaben

Eigenständige Bausteine

Bausteinfamilie ist hierarchisch strukturiert.

Schnittstelle verwendet zusätzlich Supertypen (oft abstrakt)

Beispiele:

- die Typen zur Ausnahmebehandlung in Java

- die Stromklassen aus java.io

Typisch: Trotz der hierarchischen Beziehung sind die Bausteine eigenständig verwendbar.

keine Querbeziehungen, nur Abstraktion

Eng kooperierende Bausteine

Bausteinfamilie dient **komplexer**, spezieller softwaretechnischer **Aufgabenstellung**.

Enge Kooperation führt zu **komplexen Abhängigkeiten**, z. B. verschränkt rekursiver Beziehung von Typen und Methoden.

Beispiele:

Programmgerüste, wie

- Java AWT
- San Francisco (IBM):
Abwicklung von Geschäftsprozessen

Typisch: wg. Abhängigkeiten anspruchsvoller

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 402

Ziele:

Klassifikation nach Abhängigkeit

in der Vorlesung:

Begriffe an Beispielen erläutern

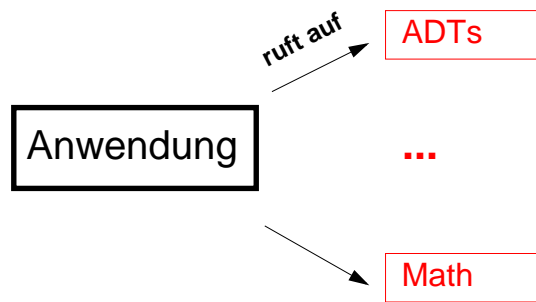
nachlesen:

Poetzsch-Heffter, Kap.4

Unterschied zwischen Programmgerüst und Bausteinbibliothek

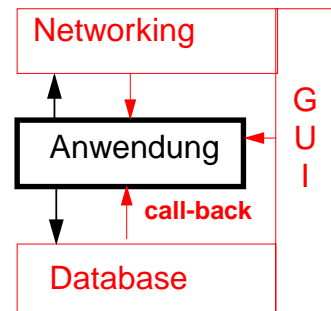
Bausteinbibliotheken

- mit unabhängigen oder eigenständigen Bausteinen
- anwendungsunabhängig
- Ablauf unter Kontrolle des Anwendungsprogramms



Programmgerüst (Framework)

- „fast-fertige“ Anwendung
- anwendungsspezifisch
- Umkehr der Ablaufsteuerung, call-back-Routinen, „don't call us, we call you“



Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 403

Ziele:

Anwendungsstrukturen

in der Vorlesung:

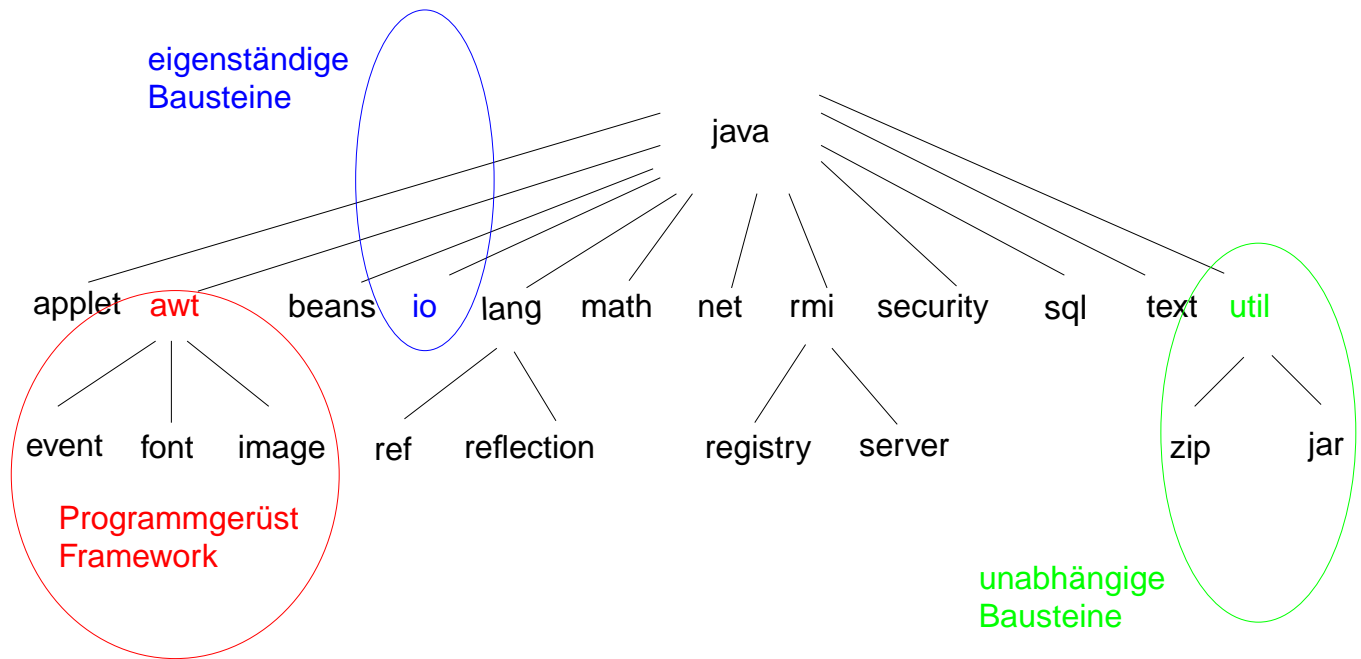
Erläuterung der beiden Strukturen und Beispiele. Abstufungen dazwischen:

- unstrukturierte Aufrufebene,
- Aufruf-Protokolle (Client/Server),
- Zusammenstecken von Bausteinen mit verborgener Kommunikation,
- komplexe Kommunikation im Programmgerüst.

nachlesen:

Poetzsch-Heffter, Kap.4

Die Java-Bibliothek



Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 404

Ziele:

Beispiel für Standardbibliotheken

in der Vorlesung:

Erläuterungen dazu

- Anwendungsbereich einiger Pakete erläutern.
- Strategie: kleine Sprache + große Bibliothek

nachlesen:

Poetzsch-Heffter, Kap.4

Anwendung von Bausteinen: Ausnahmebehandlung in Java

Ausnahmebehandlung als einfaches Beispiel für eine **erweiterbare Bausteinhierarchie**

Zeigt das enge Zusammenspiel zwischen **Sprache und Bibliothek**

Trend bei modernen OO-Sprachen: **Anzahl der Sprachkonstrukte klein** halten, mit Mitteln der Sprache formulierbare Konzepte in die **Bibliothek**;

extrem: In Smalltalk sind `ifTrue`, `ifFalse` Methoden der Bibliotheksklasse `Boolean`

Ausnahmebehandlung in Java:

Bei „abrupter Terminierung“ der Auswertung von Ausdrücken oder Ausführung von Anweisungen wird

- ein Ausnahmeobjekt `e` erzeugt und
- die in der **dynamischen Umgebung** (Laufzeitkeller) nächste, zur Klasse von `e` passende Ausnahmebehandlung mit `e` als Parameter aufgerufen.

3 Arten von Ausnahmen (siehe Klassifikation, OOP-4.7):

- Benutzer hat kaum Einfluss darauf, z. B. erschöpfte Maschinenressourcen (**Error**),
- vermeidbare Ausnahmen wegen Programmierfehlern, z. B. der Versuch, die null-Referenz zu dereferenzieren (**RuntimeException**),
- nicht vermeidbare Ausnahmen, die im Ablauf des Programms vorgesehen sein sollten, z. B. Zugriff auf nicht vorhandene Eingabe-Datei (weitere Unterklassen von **Exception**)

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 405

Ziele:

Beispiel für Bausteine im Sprachkontext

in der Vorlesung:

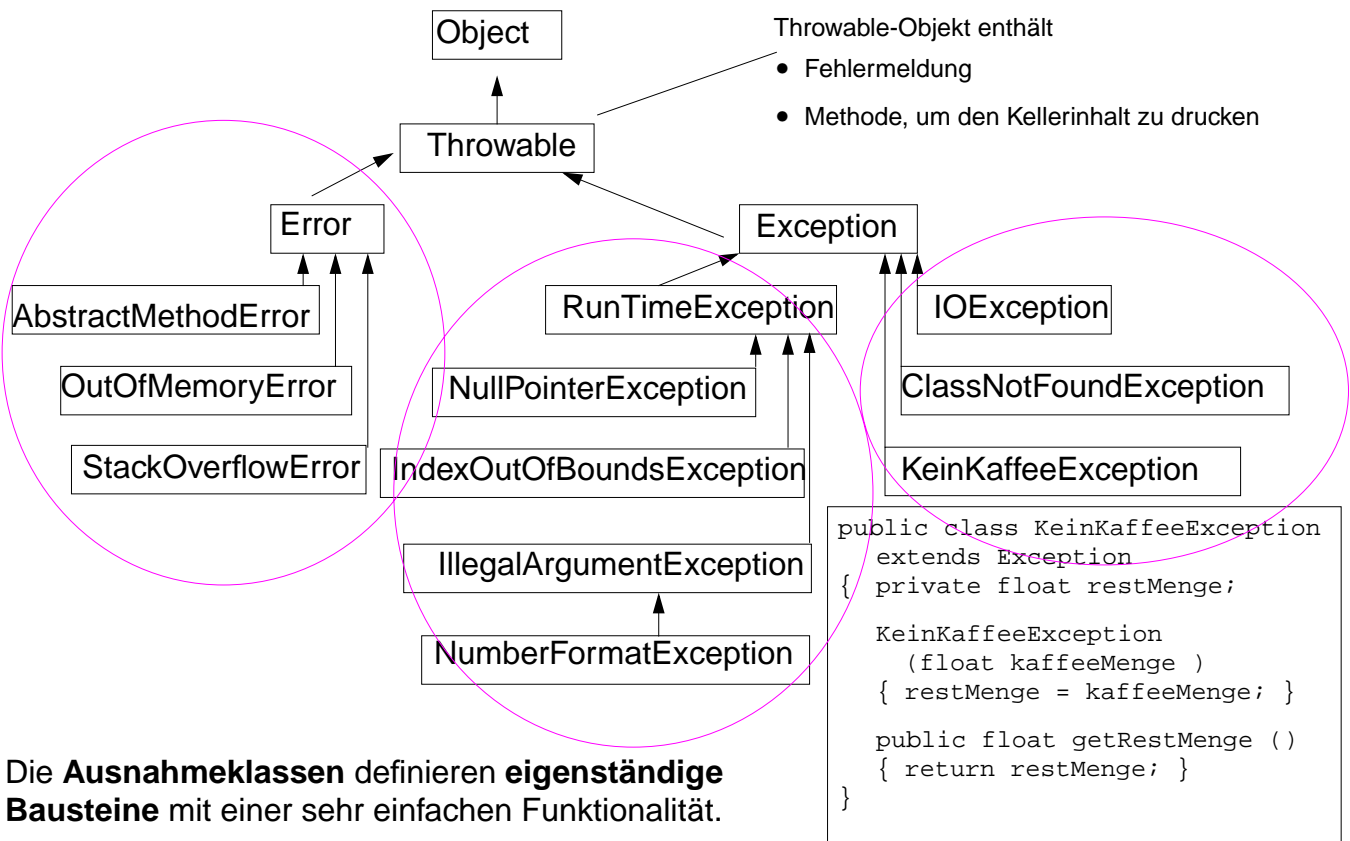
- Zusammenwirken von Sprache und Standard-Bibliothek,
- Grundbegriffe der Ausnahmebehandlung,
- Ausnahmearten

erläutern.

nachlesen:

Poetzsch-Heffter, Kap.4

Ausnahmehierarchie



Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 407

Ziele:

Erweiterbare Bausteinhierarchie

in der Vorlesung:

Erläuterungen zu

- der Hierarchie,
- Beispielen von Ausnahmen,
- Erweiterungen (über 100 in der Java Bibliothek)
- den Ausnahmearten.

nachlesen:

Poetzsch-Heffter, Kap.4

Kriterium für Entwurfsqualität: Kohärenz

Kohärenz (Bindung): Stärke des **funktionalen Zusammenhangs** zwischen den Bestandteilen eines Bausteins

Ziel: So **stark** wie möglich

Kohärenz einer Klasse:

- Klasse realisiert **nur ein semantisch bedeutungsvolles Konzept**
- ihre **Methoden kooperieren** für diese Aufgabe
- Menge der Methoden ist **vollständig**
- **keine Methode ist überflüssig** für das Konzept

starke Kohärenz weil Komponenten

funktional zusammenhängen

schwache oder keine Kohärenz weil Komponenten

zur gleichen Zeit gebraucht werden
(z.B. Initialisierung)

in der Ablaufstruktur zusammenhängen

zufällig zusammengekommen sind

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 410

Ziele:

Strukturierungskriterium Kohärenz verstehen

in der Vorlesung:

Erläuterungen:

- Kohärenz: ein einziges Konzept; alle Methoden arbeiten an dieser Aufgabe

Kriterien Entwurfsqualität: Kopplung

Kopplung: Stärke der Beziehung, Interaktion zwischen Bausteinen

- Kopplung **zwischen Methoden**, zwischen **Klassen**, zwischen **Paketen**
- Kopplung entsteht durch **Benutzen von Daten** oder durch **Aufrufe** (siehe OOP-4.11)

Ziel: Kopplung so schwach wie möglich

- schwache Kopplung entspricht schmaler Schnittstelle

dadurch bessere

- Zerlegbarkeit
- Komponierbarkeit
- Verständlichkeit
- Sicherheit

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 410c

Ziele:

Strukturierungskriterium Kopplung verstehen

in der Vorlesung:

Erläuterungen:

- Kopplung auf gleicher Abstraktionsebene

Maßnahmen zur Entkopplung

Unnötige Kopplung von Klassen **vermindert** grundlos die **Wiederverwendbarkeit**

Anwendung von Vererbungsparadigmen:

- **Rollen und Eigenschaften:**
Material entkoppelt von Werkzeugen, Verhalten entkoppelt vom Subjekt
- **Spezifikation:**
Implementierung entkoppelt von Anwendungen

Entwurfsmuster:

- **Bridge:** verfeinerte Spezifikation entkoppelt von Implementierungen
- **Strategy:** Implementierung entkoppelt von Anwendungen
- **Abstract Factory:** konkrete Erzeuger der Objekte entkoppelt von Anwendungen
- **Observer:** veränderliches Subjekt entkoppelt von Präsentatoren des Zustandes

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 411

Ziele:

Entkopplung durch Vererbungsparadigmen und Entwurfsmuster

in der Vorlesung:

Erläuterungen dazu

Beispiel zur Entkopplung: Sortieren von Personen-Listen

```
class Person
{
  Name n;
  long PersNr;
  ...
}
```

```
SortedPersonList oop =
    new SortedPersonList();
Person neu;
...
oop.add (neu);
```

Was ist hieran schlecht?

```
class SortedPersonList
{
  Object[] elems;
  ...
  public void add (Person x)
  {
    ...
    Name a = x.getName();
    Name b =
      (Name)elems [k].getName();
    if (a.lessThan(b))
      ...
    ...
  }
}
```

Diese Klasse ist für nichts
anderes zu gebrauchen!

nicht wiederverwendbar!

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 413

Ziele:

Kopplungsbeispiel für nächste die Folie

in der Vorlesung:

Erläuterung des Beispiels

Entkoppeln durch Interfaces

```
interface Comparable
{ public boolean lessThan (Object compareMe);
}

class Person implements Comparable
{ Name n;
  long PersNr;

  public boolean lessThan(Object compareMe)
  { return n.lessThan(((Person)compareMe).n); }
}

class SortedList
{ Object[] elems;
  ...
  public void add (Comparable x)
  { if (x.lessThan(elems[k]))
    ...
  }
}
```

**Schnittstelle
entkoppelt
Bausteine**

**wiederverwendbare
Klasse!**

unabhängig von den
sortierten Elementen

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 414

Ziele:

Entkopplungsmethode: Schnittstellen

in der Vorlesung:

Erläuterungen dazu. Siehe auch in 2.4 Eigenschaftsklassen

Entkoppeln durch Funktoren

Die Vergleichsfunktion wird erst bei Objekterzeugung bestimmt.

Variable Funktionen werden in Objekte verpackt [Functor Pattern von Coplien]

```
import java.util.Comparator; // seit Java 1.2

class Person {Name n; long PersNr; ...}

class PersonComparator implements Comparator
{ public int compare (Object links, Object rechts)
  { return
    ((Person)links).n.lessThan (((Person)rechts).n)?
    -1:((Person)links).n.equal (((Person)rechts).n)?
    0 : 1;
  } }

class SortedList
{ private Comparator cmp; ...
  public SortedList (Comparator c) { cmp = c; }
  ...
  public void add (Object o)
  { ... leq = cmp.compare (o, elements[k]); ...
  } }
```

Funktor:
Verpackung von
Vergleichsfunktion für Person-
Objekte

wiederverwendbar

wird mit
Vergleichsfunktion
initialisiert

Vorsicht: Passt nicht zum Grundsatz „Klasse beschreibt Zustand und Verhalten“.

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 415

Ziele:

Variable Funktionen verwenden

in der Vorlesung:

Erläuterungen dazu

- Technik: Funktionen in Objekte verpackt,
- Wiederverwendbarkeit durch Initialisierung mit Funktion(sobjekt)en
- Technik ist für Anfänger ungeeignet

Objektorientierte Programmgerüste

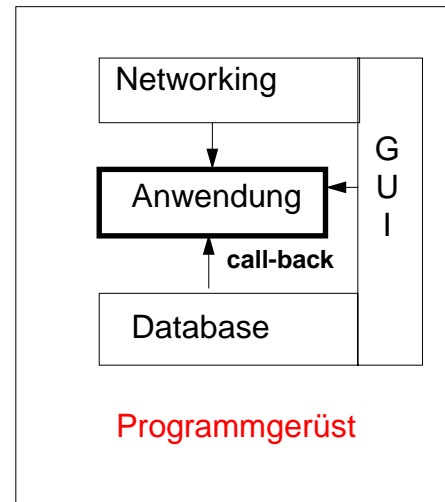
Programmgerüst (Framework):

Integrierte Sammlung von **Bausteinen, die zusammenarbeiten**, um eine **wiederverwendbare, erweiterbare Architektur** für eine **Familie verwandter Aufgaben** zur Verfügung zu stellen.

[siehe Folie 4.3]

Charakteristika:

- „fast-fertige“ Anwendung mit **geplanten Erweiterungsstellen**
- **anwendungsspezifisch**
- Umkehr der Ablaufsteuerung, **call-back-Routinen**, „don't call us, we call you“
- **komplexe Beziehungen** zwischen den Bausteinen werden **ohne Zutun des Benutzers** wiederverwendet



Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 418

Ziele:

Wiederholung des Begriffes Programmgerüst

in der Vorlesung:

Verweis auf Folie OOP-4.3

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 4

Entwicklung mit Programmgerüsten

Die Anwendung

- **spezialisiert Klassen** des Programmgerüstes,
- füllt **Erweiterungsstellen** des Programmgerüstes aus:
- implementiert **call-back-Routinen**, die vom Programmgerüst aufgerufen werden
- implementiert **Reaktionen auf Ereignisse**, die über das Programmgerüst ausgelöst werden
- setzt **Algorithmen, Klassen**, die das Programmgerüst anbietet, zu **Schnittstellen** ein, die das Programmgerüst vorgibt.

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 419

Ziele:

Gegenüberstellung von Bibliotheken

in der Vorlesung:

- Kriterien erläutern und anwenden

nachlesen:

Budd: Understanding OOP with Java, Abschnitt 4

Programmgerüst am Beispiel des Java AWT

Was macht **Java AWT** zum **typischen Beispiel** für ein **Programmgerüst**?

1. Der **Anwendungsbereich** Benutzungsoberflächen ist **komplex und vielfältig variierbar**.
2. Java AWT **deckt den Anwendungsbereich vollständig ab**;
man kann damit vollständige Benutzungsoberflächen entwickeln
- nicht nur einzelne Teilaufgaben erledigen
3. Einzelne **Komponenten** des Java AWT sind **spezialisierbar**:
bieten umfangreiche wiederverwendbare Funktionalität;
Benutzer ergänzt sie durch vergleichsweise kleine spezifische Ausprägung;
siehe Frame als Beispiel für das Konzept **Spezialisierung** (OOP-2.9)
4. **komplexes Zusammenwirken** der Komponenten liegt fast vollständig in der **wiederverwendbaren Funktionalität**.
Voraussetzung: **wohl-definierte Software-Architektur** des Java AWT;
siehe OOP-4.23

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 420

Ziele:

Typische Eigenschaften von Programmgerüsten am Beispiel

in der Vorlesung:

Erläuterungen zu

- dem Anwendungsbereich GUI,
- dem Konzept Spezialisierung durch Erinnerung an OOP-2.9, 2.10,
- dem komplexen Zusammenwirken werden auf OOP-4.23 fortgesetzt

nachlesen:

Poetzsch-Heffter, Kap. 5

Komplexes Zusammenwirken im Programmgerüst

Beispiele für komplexes Zusammenwirken der AWT-Komponenten:

- **Ereignissteuerung:**
Durch Maus, Tastatur ausgelöste Ereignisse verursachen Aufrufe von Call-back-Routinen.
siehe: Konzept eingebettete Agenten OOP-2.27, Entwurfsmuster Observer OOP-3.7a
- **Anordnung von Komponenten** (LayoutManager):
Bedienelemente werden auf bestimmten Behälterflächen nach wählbaren Strategien angeordnet;
siehe: Entwurfsmuster Strategy OOP-3.8

Die **Mechanismen für das Zusammenwirken** liegen (fast) **vollständig** in der **wiederverwendbaren Funktionalität** der Komponenten.

Benutzer brauchen sie **nicht (tiefgehend) zu kennen oder zu verstehen**.

Software-Architektur des Programmgerüsts ist Voraussetzung dafür:

- **Architekturgerüst** ist festgelegt, so dass es für alle Anwendungsvarianten passt;
- Komponenten haben **unterschiedliche Aufgaben**,
z. B. Container, Event, Listener, LayoutManager (OOP-4.24)
- **Regeln zum Zusammensetzen** einer Anwendungsvariante

Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 423

Ziele:

Hintergrund für das Zusammenwirken

in der Vorlesung:

- Erinnerung an die Themen der beiden Beispiele,
- Erläuterungen zum Begriff Software-Architektur,
- Weiteres Beispiel: Übersetzer mit wohldefinierter Architektur

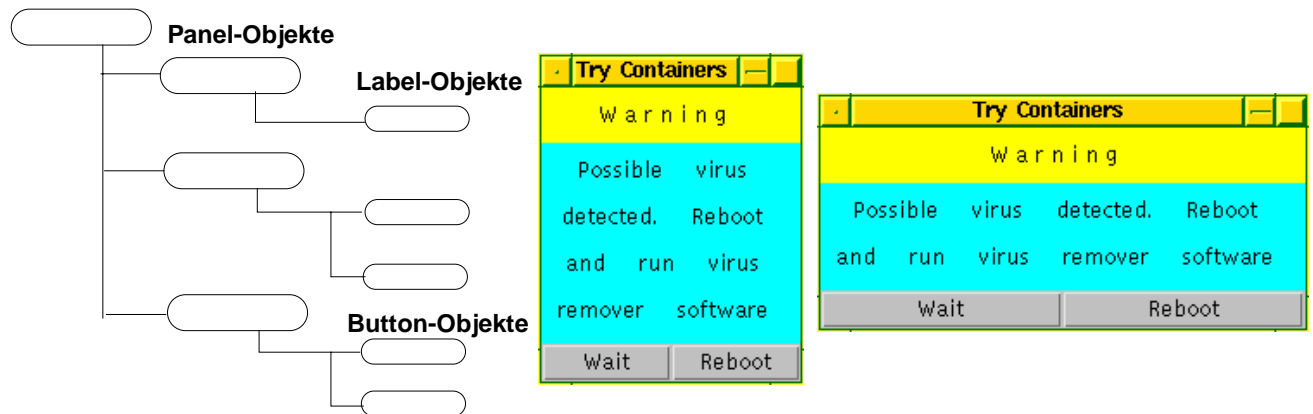
nachlesen:

Poetzsch-Heffter, Kap. 5.2.3.2, 5.2.3.5

Architekturkonzept: hierarchisch strukturierte Fensterinhalte

- Zusammengehörige Komponenten in einem Objekt einer **Container**-Unterklasse unterbringen (**Panel**, **Window**, **Frame** oder selbstdefinierte Unterklasse).
- Eigenschaften der Objekte im **Container** können dann gemeinsam bestimmt werden, z. B. Farbe, **LayoutManager**, usw. zuordnen.
- Mit **Container**-Objekten werden beliebig tiefe Baumstrukturen von AWT-Komponenten erzeugt. In der visuellen Darstellung sind sie ineinander geschachtelt.

Frame-Objekt



Vorlesung Objektorientierte Programmierung WS 2013/2014 / Folie 424

Ziele:

Beispiele für Architekturkonzepte

in der Vorlesung:

- Verschiedene Aufgaben von Komponenten: Frame und Panel als Container; LayoutManager
- Regeln zum Zusammensetzen: Bedienelemente auf Container; LayoutManager zu Container
- Voraussetzungen für automatisches Zusammenwirken
- trotzdem Freiheit und Vielfalt der konstruierbaren Anwendungen

nachlesen:

Poetzsch-Heffter, Kap. 5.2.3, 5.3.1