Tagesprogramm

Programmierparadigmen

Basiskonzepte der objektorientierten Programmierung

Berechnungsmodell

formaler Hintergrund (Funktionen, diverse Logiken, Algebren, Automaten, etc.)

praktische Realisierung entscheidend:

- Kombinierbarkeit
- Konsistenz
- Abstraktion
- Systemnähe
- Unterstützung und Beharrungsvermögen

Struktur im Kleinen

strukturierte Programmierung: Sequenz, Auswahl, Wiederholung

Umgang mit **Querverbindungen** durch Seiteneffekte

→ Seiteneffekte verbieten (funktional)

→ Querverbindungen sichtbar machen (objektorientiert)

First-Class-Entities

→ hoher Aufwand, lohnt sich nur für wichtigste Konzepte

Modularisierungseinheiten

Modul

Objekt

Klasse

Komponente

Namensraum

Parametrisierung

dynamisches Befüllen der Löcher (zur Laufzeit)

- Konstruktor
- Initialisierungsmethode
- zentrale Ablage

statisches Befüllen der Löcher

- Generizität
- Annotationen
- Aspekte

Ersetzbarkeit

A durch B ersetzbar wenn B überall verwendbar wo A erwartet

Schnittstellen von A und B spezifizierbar durch

- Signatur
- Abstraktion
- Zusicherungen
- überprüfbare Protokolle

Aufgabe: Modularisierungseinheiten

Warum können Module und Klassen nicht zyklisch voneinander abhängen?

A: Zur Erreichung eines hohen Grads an Wiederverwendung.

B: Weil es Übersetzungseinheiten sind.

C: Weil sich diese Übersetzungseinheiten auf andere Übers.einheiten beziehen.

D: Zwecks Ersetzbarkeit.

Aufgabe: Ersetzbarkeit und Parametrisierung

Warum ist Ersetzbarkeit keine Form der Parametrisierung?

A: Weil Ersetzbarkeit keine Wiederverwendung unterstützt.

B: Weil Ersetzbarkeit keine Löcher füllt.

C: Weil für Ersetzbarkeit alle Details von Anfang an definiert sein müssen.

D: Weil Ersetzbarkeit nicht mit Generizität kompatibel ist.

Typisierung

statische versus dynamische Typprüfungen

= Einschränkungen versus Freiheit

= diktierte Disziplin versus freiwillige Disziplin

statische Typen verbessern Lesbarkeit

→ Sicherheit durch Lesbarkeit, nicht durch Typprüfungen

Abstraktion und Typen

struktureller Typ $\longrightarrow$ nur Signatur

nominaler Typ $\longrightarrow$ Typname ermöglicht Abstraktion

abstrakter Datentyp (ADT)

$=$ nominale Schnittstelle einer Modularisierungseinheit

$$\left.\begin{array}{r}\text{Objekt aus realer Welt}\\ \text{Softwareobjekt}\end{array}\right\}\ \text{gemeinsamer Name}$$

Denken in Konzepten

Untertypen

Ein Typ U ist Untertyp eines Typs T wenn jede Instanz von U überall verwendbar ist wo eine Instanz von T erwartet wird.

für strukturelle Typen eindeutig

für ADT in der Verantwortung der Programmierer

Objekt

kapselt Variablen und Methoden zu Einheit

Eigenschaften:

- Identität
- Zustand
- Verhalten

Softwareobjekt (ADT) simuliert, abstrahiert reales Objekt

Klasse

beschreibt Struktur der Objekte

Konstruktoren zur Initialisierung aller Objekte

gleiche Klasse $\longrightarrow$ selbe Schnittstellen und selbes Verhalten

jedes Objekt ist Instanz genau einer Klasse

Untertypen und Vererbung

Untertypen bedingen Ersetzbarkeit

→ Variable hat **deklarierten** und **dynamischen Typ**

→ dynamisches Binden

Vererbung = Übernehmen von Code aus Oberklasse

Praxis: Vererbung als Hilfsmittel für Untertypen

Erfolg in der OO-Programmierung

gezielter Einsatz von **Erfahrung** erhöht Erfolgsaussichten

OOP = viele Möglichkeiten zur **Faktorisierung**

→ erleichtert gezielten Einsatz von Erfahrung

→ ermöglicht Wiederverwendung durch Ersetzbarkeit

→ überfordert Anfänger

OOP gut für große, langlebige Programme

OOP schlecht für komplexe Algorithmen

Verantwortlichkeiten einer Klasse

definiert durch drei w-Ausdrücke, „ich“ ist Objekt der Klasse:

was ich weiß	(Zustand der Objekte)
was ich mache	(Verhalten der Objekte)
wen ich kenne	(sichtbare Objekte, Klassen)

wer Klasse entwickelt ist zuständig für Änderungen in den Verantwortlichkeiten

Klassen-Zusammenhalt

Klassen-Zusammenhalt (class cohesion)

= Grad der Beziehungen zwischen den Verantwortlichkeiten der Klasse

hoch, wenn

Variablen und Methoden eng zusammenarbeiten

und durch Klassenname gut beschrieben

Klassen-Zusammenhalt soll hoch sein

→ Hinweis auf gute Faktorisierung

→ verringert Wahrscheinlichkeit für nötige Änderungen

Objekt-Kopplung

Objekt-Kopplung (object coupling)

= Abhängigkeit der Objekte voneinander

stark, wenn

- viele sichtbare Methoden und Variablen
- viele Nachrichten im laufenden System
- viele Parameter in Methoden

Objekt-Kopplung soll schwach sein

→ Hinweis auf gute Kapselung

→ weniger unnötige Beeinflussung bei Änderungen