Tagesprogramm

Ausgewählte Aspekte von Programmierparadigmen

Basiskonzepte objektorientierter Programmierung

Was Paradigmen ausmacht

Berechnungsmodell imperativ, funktional, logikorientiert

Umgang mit **Querverbindungen** durch Seiteneffekte

→ Seiteneffekte verbieten funktional und referenziell transparent

→ Querverbindungen sichtbar machen objektorientiert

First-Class-Entities

→ hoher Aufwand, lohnt sich nur für wichtigste Konzepte

Modularisierungseinheiten

Objekt	existiert erst zur Laufzeit
Klasse	Vorlage für Objekterzeugung
Modul	Übersetzungseinheit, gegenseitig referenziert, zyklenfrei
Komponente	Überseztungseinheit, keine gegenseitigen Referenzen, Deployment
Namensraum	strukturiert größere Einheiten, keine weiteren Eigenschaften

Parametrisierung

dynamisches Befüllen von Lücken (zur Laufzeit)

- Konstruktor — einfach
- Initialisierungsmethode — Initialisierung in mehreren Schritten
- zentrale Ablage — geht von erzeugtem Objekt aus

statisches Befüllen von Lücken

- Generizität — explizite Lücken, vor allem für Typen geeignet
- Annotationen — Kenntnis der Lücken nicht nötig
- Aspekte — Querschnittsfunktionalität von Kernfunktionalität getrennt

Ersetzbarkeit

A durch B ersetzbar wenn B überall verwendbar wo A erwartet

Schnittstellen von A und B spezifizierbar durch

- Signatur — statisch prüfbar
- Abstraktion — intuitiv
- Zusicherungen — informelle Verträge (Design-by-Contract)
- Protokolle — formale Verträge, noch nicht etabliert

Aufgabe: Ersetzbarkeit und Parametrisierung

Warum ist Ersetzbarkeit keine Form der Parametrisierung?

A: Weil Ersetzbarkeit keine Wiederverwendung unterstützt.

B: Weil Ersetzbarkeit keine Lücken füllt.

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

D: Weil Ersetzbarkeit nicht mit Generizität kompatibel ist.

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(innen)

Objekteigenschaften

gelten auch für andere Modularisierungseinheiten

Objekt kapselt Variablen und Methoden zu Einheit

Identität Adresse

Zustand Variableninhalte

Verhalten was Methoden machen

Softwareobjekt (ADT) simuliert und abstrahiert reales Objekt

Klasse

beschreibt Struktur der Objekte

Konstruktoren zur Initialisierung der Objekte

gleiche Klasse $\longrightarrow$ gleiche Schnittstellen und gleiches 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

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 Verantwortlichkeiten der Klasse

hoch, wenn

Variablen und Methoden eng zusammenarbeiten

und durch Klassenname gut beschrieben

Klassen-Zusammenhalt soll hoch sein

→ Hinweis auf stabile Faktorisierung Refaktorisierung eher schwierig

→ 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 ≠ stabile Faktorisierung

→ weniger unnötige Beeinflussung bei Änderungen Refaktorisierung einfacher