OOP

Wiederholung

Berechnungsmodell

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

praktische Realisierung entscheidend:

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

Struktur im Kleinen

widersprüchlich: flexibel, sicher, verständlich

→ breites Spektrum an Sprachen, Evolution nichtlinear

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 (Übersetzungseinheit, zyklenfrei)

Objekt (zur Laufzeit)

Klasse (Modul und Muster für Objekterzeugung)

Komponente (komplexere Initialisierung)

Namensraum

Parametrisierung

Befüllen der Löcher zur Laufzeit:

- Konstruktor
- Initialisierungsmethode
- zentrale Ablage

Befüllen der Löcher zur Übersetzungszeit:

- Generizität
- Annotationen
- Aspekte

Aufgabe: Modularisierung und Parametrisierung

Such Sie in Gruppen zu zwei bis drei Personen Antworten auf folgende Fragen:

1. Was unterscheidet Module, Objekte, Klassen, Komponenten voneinander?
2. Welche Eigenschaften haben unterschiedliche Formen der Parametrisierung?
3. Hängt die Parametrisierung von der Art der Modularisierungseinheiten ab?

Zeit: 2 Minuten

Ersetzbarkeit

A durch B ersetzbar wenn B überall verwendbar wo A erwartet

Schnittstellen von A und B spezifizierbar durch

- Signatur (einfach, wenig aussagekräftig)
- Abstraktion (intuitiv, nicht immer zuverlässig)
- Zusicherungen (manchmal komplex, recht zuverlässig)
- überprüfbare Protokolle (sehr komplex, nicht verfügbar)

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
→ ohne Deklarationen vom Compiler prüfbar

für ADT in der Verantwortung der Programmierer
→ explizite Deklaration von Untertypbeziehungen

Zusicherungen als Ergänzung
→ müssen in Untertypbeziehungen berücksichtigt werden

Gestaltungsspielraum für Typen

Typen zeigen Grenzen auf:

- kovariante Probleme bei Untertypen
- rekursive Datentypen erfordern induktive Konstruktion
- Typinferenz mächtig, aber nicht mit Untertypen

Typen bieten ungeahnte Möglichkeiten:

- praktisch alle Eigenschaften statisch propagierbar
- aber derzeit nur indirekt unterstützt (z.B. Annotationen)

Erfolg in der OO-Programmierung

gezielter Einsatz von **Erfahrung** erhöht Erfolgsaussichten

OOP = viele Möglichkeiten zur **Faktorisierung**

$\longrightarrow$ erleichtert gezielten Einsatz von Erfahrung

$\longrightarrow$ ermöglicht Wiederverwendung durch Ersetzbarkeit

$\longrightarrow$ überfordert Anfänger

OOP gut für große, langlebige Programme

OOP schlecht für komplexe Algorithmen

Lernziele

Programmierparadigmen	verstehen
Ersetzbarkeit und Wiederverwendung (Schwerpunkt)	erschaffen
Generizität, dynamische Typinformation	anwenden, analysieren
Ausnahmen, Nebenläufigkeit, Annotationen, Aspekte	anwenden, analysieren
ausgewählte Softwareentwurfsmuster	verstehen, anwenden
Modelle, Abstraktionen und Softwarelösungen (auch für unvollständig spezifizierte Probleme)	erschaffen
Selbstorganisation im Team	erschaffen

erinnern verstehen anwenden analysieren evaluieren erschaffen

Danke für Ihre Mitarbeit

Viel Freude und Erfolg

bei Abgabegespräch und Prüfung
beim Programmieren
und im echten Leben