Tagesprogramm

Zusammenfassung und Ausblick

Programmstruktur im Kleinen

widersprüchliche Ziele: 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)

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, meist nicht verfügbar)

Untertypen

U ist Untertyp von T wenn jede Instanz von U überall verwendbar wo Instanz von T erwartet

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)

Aufgabe: Statische und dynamische Sprachen

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

Was spricht für/gegen statische/dynamische Sprachen?

Zeit: 2 Minuten

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

Danke für Ihre Mitarbeit

Viel Freude und Erfolg

bei Abgabegespräch und Prüfung,
beim Programmieren
und im echten Leben