

Programmiertechnik II

Weitere Konzepte

Mehrfachvererbung

- Mehrfachvererbung: mehrere Basisklassen
 - Vereinigung von Zustand und Verhalten
 - Was passiert bei doppelten Methoden oder Attributen?
 - C++: bei Zugriff muss mittels der Basisklasse qualifiziert werden
 - o->Base1::member = value1;
 - o->Base2::member = value2;
 - Und was ist, wenn die Methoden virtuell sind?
 - C++: Für jede virtuelle Methode muss es einen “unique final overrider” geben
 - Implementierung: Layout der Basisklassen wird hintereinander gespeichert
 - Viele Probleme:
 - Konvertierung zwischen Klasse und Basisklasse erfordert Pointer-Korrektur
 - Virtuelle Basisklassen: Zustand soll nur einmal vorkommen
 - ...

Interfaces

- Motivation: Ein Objekt soll in verschiedenen Rollen auftreten
 - Mehrfachvererbung gilt aber als problematisch
- Lösung: Eine Klasse darf eine Basisklasse besitzen, aber beliebig viele Schnittstellen (Interfaces) implementieren
 - Polymorphie: Konvertierung von Klassen in eine implementierte Schnittstelle ist typrichtig
- Implementierung: wie virtuelle Methoden
 - Exemplare verweisen auf mehrere virtuelle Methodentabellen (eine pro Schnittstelle)

Properties

- Kapselung: Implementationsdetails des Objekts sollen versteckt und Invarianten bewahrt werden
 - Zugang zu Zustand nur über Zugriffsmethoden
 - aber: Objekte haben oft “öffentlich bekannten Zustand”
- Lösung 1: Zugriffsmethoden (getter/setter)
- Lösung 2: Properties
 - Nutzer greifen auf Properties zu wie auf Attribute
`p.name = p.name + “(Jr.)”`
 - Property-Zugriff wird automatisch in Methodenruf umgesetzt
- Implementierungsstrategie:
 - C#: Kurznotation: Compiler ersetzt Property-Zugriffe “sofort”
 - Python: Dynamisch: bei Zugriff wird Property-Methode gesucht

Templates/Parametrisierte Typen (*generic types*)

- Kein OO-Konzept im eigentlichen Sinne
- Implementierungsstrategie:
 - C++: Duplizieren des Template-Körpers für jeden Parametertypen
 - Java, C# (?): Parametrisierung zur Laufzeit (Erweiterung des Byte-Code-Formats um “generische” Byte-Codes)

Ausnahmebehandlung

- Kein OO-Konzept im eigentlichen Sinne
- Kernkonzepte:
 - Auslösen und Abfangen (Behandeln) von Ausnahmen
 - Hierarchien von Ausnahmen
- Weitere Konzepte
 - Wiederaufnahme nach beendeter Behandlung (Dylan)
 - Stack-Unwinding
 - Automatische Destruktorrufe für lokale Variablen (C++)
 - finally-Blöcke (Java, Python, C#)
 - Deklaration möglicher Ausnahmen für Funktionen
 - C++: optional
 - Java: vorgeschrieben

Ausnahmebehandlung (2)

- Exception-Sicherheit (safety): Alle Operationen funktionieren “richtig” auch falls eine Ausnahme auftritt
 - Commit or rollback: Operationen gelingen entweder vollständig oder scheitern vollständig
 - Rollback ist nicht immer implementierbar
 - Einhaltung von Invarianten
 - Temporär allozierte Ressourcen werden auch bei einer Ausnahme wieder freigegeben (Invariante: Menge allozierter Ressourcen wird von Methode nicht verändert)