

Objekt-Orientierte Programmiersprachen

Martin Gasbichler, Holger Gast

19.1.2006

Plan

- Vergleich: Smalltalk vs. Objective-C
- Aspekte klassenbasierter Sprachen

Smalltalk vs. Objective-C

- Klassen als Objekte
- Klassendeklarationen
- Selektoren als first-class values
- Objekt-Erzeugung
- Implementierung: Eigene VM vs. Einbettung in C

Klassen als Objekte

- Smalltalk: Everything is an object
 - Klassen *sind* Objekte
 - Klassen sind Instanzen von Klassen
 - Brauchen Metaklassen → Zirkelschluss
 - Klassennamen sind globale Variablen
- Objective-C: Klassen gehören zum Typ `Class`
 - `Class` erbt nicht von `NSObject`
 - Jede Klasse ist Instanz einer internen, nicht zugänglichen Metaklasse
 - Klassen antworten nicht auf `class`-Nachricht (eigentlich)
 - Klassennamen sind keine Variablen

Self-Nachrichten an Klassen in Objective-C

```
@interface Foo{}
+(void)f;
+(void)g;
@end
@implementation Foo
+(void)f{
    [self g];
}
+(void)g{
    printf("I'm Foo\n");
}
@end
```

```
int main() {
    [Foo f];
    [Bar f];
}
```

```
@interface Bar : Foo{}
+(void)g;
@end
@implementation Bar
+(void)g{
    printf("I'm Bar\n");
}
@end
```

```
I'm Foo
I'm Bar
→ Metaklassen erlauben Überschreiben
von Klassenmethoden
```

Selektoren als first-class values

- Smalltalk+Objective-C: Methodennamen für Aufruf berechnen
- Methode `perform:` (bzw. `performSelector:`) führt Aufruf durch
- Smalltalk: Methodename ist Datentyp `Symbol`
- Objective-C: Methodename ist Datentyp `SEL`

Objekt-Erzeugung

- Smalltalk+Objective-C:
 - Klassenmethode erzeugt neues (leeres) Objekt
 - Initialisierung durch weitere Methode
- Konvention: Hauptarbeit der Initialisierung
 - Objective C: [[C alloc] init...] → Instanzmethode
 - Smalltalk: C new... → Klassenmethode

Implementierung: Eigene VM vs. Einbettung in C

- Smalltalk
 - Implementierung: VM + Images, Integrierte Umgebung
 - Vorteil: Debugging mit detaillierten Ablaufinformationen
 - Vorteil: Garbage Collection
 - Nachteil: Geschwindigkeit
 - Nachteil: Keine stand-alone Applikationen
- Objective-C
 - Einbettung in C + mächtiges Laufzeitsystem
 - Vorteil: Einfachste Wiederverwendung von C-Bibliotheken
 - Vorteil: Geschwindigkeit steuerbar durch Kopplung (C-Funktionen oder Methoden)
 - Nachteil: Typprobleme von C erhalten (Segmentation faults)
 - Nachteil: Kein Garbage Collector

Aspekte klassenbasierter Sprachen

. . . anhand der behandelten Sprachen

- Klassendeklarationen
- Objekte sind Instanzen von Klassen
- Vererbung für Wiederverwendung von Code
- Klassen definieren Typen
- Protokoll
- Abstrakte Klassen
- Auswahl der Methodenimplementierung
- Klassenmethoden und Instanzmethoden
- Klassen als Werte

Klassendeklarationen

- Java,C++: Vollständig im Programmtext
- Objective-C: Hauptsächlich im Programmtext
 - @interface + @implementation Direktiven
 - Categories fügen Methoden zu bestehenden Klassen hinzu (Link-Time)
 - Verändern der Methoden zur Laufzeit möglich (z.B. Posing)
- Smalltalk: Vollständig dynamisch
 - Nachricht subclass: an Superklasse
 - Enthält Name, Instanzvariablen, Klassenvariablen (und mehr)
 - Methoden interaktiv definieren im Browser

Objekte sind Instanzen von Klassen

- Klassen beschreiben Objekte
 - Definieren Instanzvariablen
 - Implementierung der Methoden
- Klassen erzeugen neue Objekte
 - Java, C++: Allokation + Konstruktor (-Funktion)
 - Objective-C, Smalltalk: Allokation + Initialisierungsmethoden

Vererbung für Wiederverwendung von Code

- Methoden + Variablen aus der Superklasse übernehmen
- Gezielte Änderungen und Erweiterungen für neue Aufgaben
- `super` für Zugriff auf überschriebene Methoden
 - Java, C++: `super` impliziert statische Bindung
 - Smalltalk, Objective-C: dynamic dispatch, Suche beginnt in Superklasse (Klassenhierarchie & Methoden können sich dynamisch ändern!)
- Mechanismus Vererbung
 - Java, C++, Objective-C: Bei der Klassendeklaration
 - Smalltalk: `subclass:` Nachricht an Superklasse

Klassen definieren Typen

Objekt hat genau dann (dynamischen) Typ K ,
wenn es Instanz der Klasse K ist.

- Java, C++ (, Objective-C): Statisches Typsystem
- Objective-C, Smalltalk: Dynamische / latente Typen
- Vererbung ergibt Subtypen
 - Polymorphie
 - Ersetzbarkeit
 - Konversionen

Protokolle

Protokolle definieren unabhängig von Klassen die Schnittstellen von Objekten

→ Protokolle sind Listen der verstandenen Nachrichten

Anwendung: Statische Überprüfung von Nachrichtenausdrücken im Typchecker

- Java: Interfaces
- C++: public virtual base classes
- Objective-C: Protocols
- Smalltalk: — (keine statische Prüfung)

Abstrakte Klassen

Abstrakte Klassen stellen ihren Subklassen Methoden und Instanzvariablen zur Verfügung. Sie sind aber selbst noch nicht vollständig und sollen daher keine Instanzen erzeugen.

Umsetzung:

- Java: abstrakte Methoden und abstrakte Klassen
- C++: pure virtual methods
- Objective-C: Konvention (Dokumentation)
- Smalltalk: Konvention `“self subclassResponsibility”` in abstrakten Methoden

Auswahl der Methodenimplementierung

- Polymorphie: Verschiedene Objekte können unterschiedlich auf die gleiche Nachricht reagieren.
 - Dynamische Bindung: Objekt wählt Implementierung der Methode aus.
- ⇒ Zentraler Mechanismus zur Entkopplung in OO

Frage: Interaktion mit Klassenmechanismus?

Polymorphie

Welche Objekte behandelt die Sprache als “polymorph”?

→ Welche Objekte erlaubt die Sprache als mögliche Empfänger einer Nachricht?

- Java: Objekte mit gleicher (Super)klasse oder gleichem Interface
- C++: Objekte mit gleicher (Super)klasse (aber: Mehrfachvererbung)
- Objective-C: Breites Spektrum von Kopplung
 - Objekte mit gleicher Superklasse oder formalem Protokoll
 - Informelles Protokoll für NSObject
 - Beliebige Objekte (bei Typ id)
- Smalltalk: Beliebige Objekte (unabhängig von Klassen)

Dynamic dispatch

Auflösung von Polymorphie erfolgt zur Laufzeit durch Auswahl der Methodenimplementierung zu einer Nachricht.

- Java, C++: Statische Klassenstruktur ermöglicht vorberechnete Tabellen (Compiler kann dabei auch Überladung nach (statischen) Argumenttypen auflösen)
- Objective-C, Smalltalk:
 - Suche zur Laufzeit nach Methode für Selektor
 - Selektor ist kanonische Repräsentation des Methodennamens

Klassenmethoden und -variablen

Klassen beinhalten genau wie Objekte Daten und Operationen auf diesen Daten.

- Java, C++: `static`
- Objective-C: +-Kennzeichner + C-`static`
- Smalltalk: Klassen *sind* Objekte

Klassen als Werte

Sind Klassen in der Sprache first-class values?

- C++
 - Klassen sind keine Werte (Teilweiser Ersatz: RTTI)
 - Klassennamen werden als Namespaces verwendet, bezeichnen aber keine Werte zur Laufzeit
- Java
 - Klassen sind Laufzeitwerte vom Typ `Class`
 - `Class` ist Subklasse von `Object`
 - Klasse von `Class` ist `Class` selbst (→ keine Metaklassen)
 - ⇒ Kein `this` in Klassenmethoden
 - ⇒ Kein dynamic dispatch für Aufrufen von Klassenmethoden

Klassen als Werte (Fortsetzung)

- Objective-C
 - Klassen sind Objekte, sie antworten auf Nachrichten
 - Class ist eigener Typ, keine Subklasse von NSObject
 - Class hat selbst keine (sichtbare) Klasse, Metaklassen werden nur intern verwendet⇒ self + dynamic dispatch für Klassenmethoden
- Smalltalk
 - Class ist Subklasse von Object
 - Klassen sind vollständige Objekte, sie sind selbst Instanzen von Metaklassen⇒ self + dynamic dispatch für Klassenmethoden

Résumé

- Klassenbasierte Sprachen zeigen viele Gemeinsamkeiten bei *Objekt, Klasse, Objekt-orientierte Programmierung*
- ⇒ Unterschiede im detaillierten Verständnis
- ⇒ Konzepte werden erst bei detaillierter Betrachtung sichtbar
 - Java-Compiler
 - Vergleich mit anderen Sprachen
- Erlernen neuer Sprachen
 - Konzepte vorgestellt
 - Herangehensweise:
 - Ist ein bestimmtes Konzept vorhanden?
 - Wie ist es umgesetzt?
 - Ergeben sich Abweichungen zu bekannten Sprachen?