Theorie und Implementation objektorientierter Programmiersprachen

Slide 1

Kategorie:	PG
Dozent:	Andreas Abel (`abel@informatik.uni-muenchen.de`)
Vorkenntnisse:	OO-Sprache, am Besten JAVA
	Grammatiken, strukturelle Induktion
Schein:	ja, Anforderungen werden noch festgelegt
Web-Seite:	`http://www.tcs.informatik.uni-muenchen.de/~abel/OO/`

Geschichte objektorientierter Programmierung

Slide 2

1. Vorläufer: SIMULA 1 (1964): Sprache für Simulationen.
2. Smalltalk (1972): Erste rein objektorientierte Sprache.
3. C++ (1986): Hybridsprache: Prozedural und objektorientiert.
4. JAVA (1995): Typsichere (?) OO-Sprache für platformunabhängige Anwendungen.
5. weitere: ObjectPascal, MODULA-3, Oberon ...

Wesentliche Konzepte der Objektorientierung

Slide 3

Operationelle Konzepte:

1. Methoden und dynamische Bindung
2. Vererbung
3. Späte Bindung

Sicherheitskonzepte:

4. Datenkapselung
5. Subtyping (Interfaces, Subclasses)

Sicherheit durch formale Methoden

Slide 4

- Anforderung an moderne Programmiersprachen: zur Verlässlichkeit und Robustheit von Software beitragen.
- Vollständige Korrektheitsbeweise durch formale Methoden sind sehr aufwendig und meist zu teuer.
- Schlanke formale Methoden: beschränken sich auf automatisch testbare Eigenschaften von Programmen.
  1. Typ-Systeme
  2. Model-Checker
  3. Laufzeitüberwachungssysteme

Typ-Systeme

Slide 5

- Definition (Benjamin Pierce [1]):

  A type system is a syntactic method for automatically proving the absence of certain program behaviours by classifying phrases according to the kinds of values they compute.

- Typsicherheit:
  1. Jeder korrekte Programmausdruck hat einen Typ.
  2. Der Typ eines Ausdrucks ändert sich durch Auswertung nicht.
  3. Ein wohlgetypter Ausdruck erzeugt keine unerwarteten Laufzeitfehler.
- Slogan: *Well-typed programs cannot go wrong*.

Statische und dynamische Typprüfung

Slide 6

	Statisch	Dynamisch
Typprüfung	beim Übersetzen	zur Laufzeit
Vorteil	Fehler früher erkannt	Nur wirkliche Fehler
Nachteil	Korrekte Programme abgelehnt	Fehler "schlummern"
Sprachen	ML, Haskell C, C++, JAVA	LISP, Scheme JAVA

Bemerkung: Nicht alle Sprachen mit Typprüfung sind auch typsicher. Unsicher sind z.B. C und C++.

Typprüfung für objekt-orientierte Sprachen

Slide 7

- Typsysteme für objekt-orientierte Sprachen sind kompliziert:
- Das Typ-System von JAVA ist stellenweise unnötig restriktiv.
- Beispiel: Cloning.
- Manchmal ist das operationelle Verhalten überraschend.
- Beispiel: Equality.
- Das Typ-System von JAVA enthält fehlerhafte Regeln.
- Beispiel: Arrays.

Ziel und Inhalt der Vorlesung

Slide 8

Ziel: Beherrschung von Typsystemen für OO-Sprachen.

1. Theorie
   - Schrittweise Einführung in Typen und korrekte Typregeln.
   - Beweis von Typsicherheit für zunehmend mächtigere Systeme.
2. Implementation
   - Umsetzen der theoretischen Erkenntnisse in funktionierende Typprüfer.
   - Entwicklung eines typsicheren Interpreters für eine objektorientierte Sprache.

References

[1] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

Slide 9