Zusammenfassung und Ausblick

Martin Wirsing

in Zusammenarbeit mit
Michael Barth, Fabian Birzele und Gefei Zhang

http://www.pst.informatik.uni-muenchen.de/lehre/WS0506/infoeinf/

WS 05/06

Zusammenfassung: Objekt-orientierte Programmierung und Software-Entwicklung

Objekt-orientierte Programmierung und Software-Entwicklung

- Ein **Objekt** besitzt
  - lokalen Zustand (Attribute)
  - Methoden zur Änderung des Zustandes
- **Modulare Programmierung** durch Bildung von Klassen, Kapselung
- Unterstützung von Wiederverwendung und Änderung von Programmen durch **Vererbung**
- Robuste Programmierung durch **selbstdefinierte Ausnahmen**
- Software-Entwurf mit **UML-Klassendiagrammen**
- Systematische **(Blackbox-) Tests** mittels JUnit

Zusammenfassung: Objektorientierte Programmierung und Software-Entwicklung

Grundlagen der Informatik

- Syntax von Programmiersprachen (EBNF)
- Parameterübergabemechanismen
- Komplexität

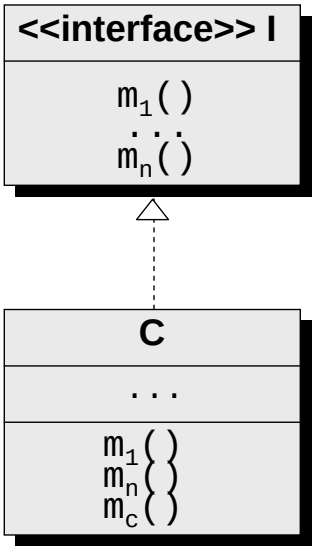
Zusammenfassung: Objektorientierte Programmierung und Software-Entwicklung

In der Vorlesung NICHT behandelt:

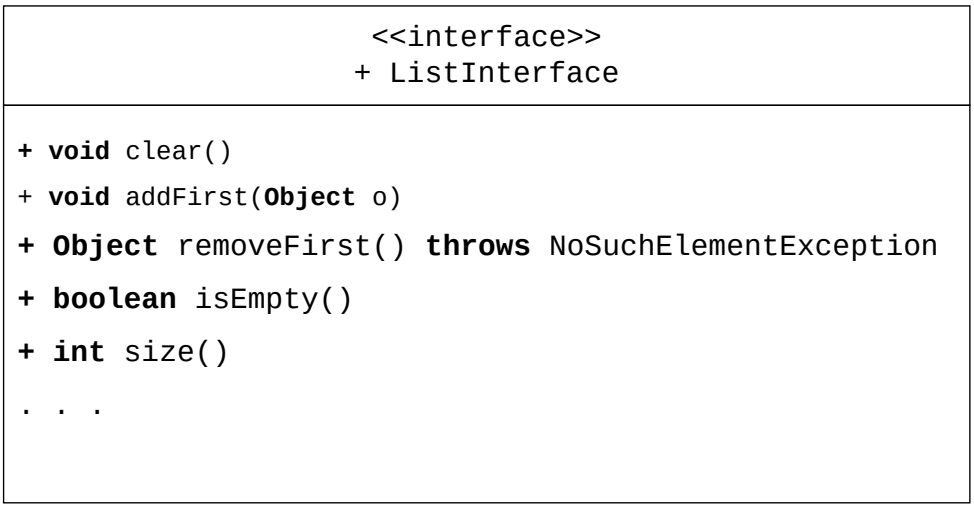
- **Einsatz von Schnittstellen** zur modularen Programmierung
- Abstrakte Klassen und Pakete
- **Nebenläufige Programmierung**
- **Systematische OO-SW-Entwicklung** mit UML

Schnittstellen

- Eine **Schnittstelle** in Java (Schlüsselwort „**`interface`**“) deklariert eine Menge von Methoden (ohne Angabe eines Rumpfs) und Konstanten (**aber** keine Attribute). Man nennt eine Methode ohne Rumpf „abstrakte Methode“. Im Gegensatz zu Klassen ist Mehrfachvererbung erlaubt, d.h. eine Schnittstelle kann Erbe mehrerer Schnittstellen sein.
- Eine Klasse `C` **implementiert** eine Schnittstelle `I`, wenn alle Methoden der Schnittstelle in `C` mit ihrer exakten Funktionalität implementiert werden, und zwar durch „öffentliche“ Methoden.



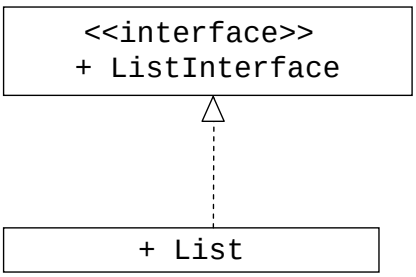
Beispiel: Schnittstelle für Listen (UML)



Beispiel: Schnittstelle für Listen (Java)

```
public interface ListInterface
{/** Removes all of the elements from this list. */
  public void clear();

  /** Inserts the given element at the beginning of this list. */
  public void addFirst(Object o);

 /** Removes and returns the first element from this list. */
  public Object removeFirst() throws NoSuchElementException;

/** Returns true if this list contains no elements. */
  public boolean isEmpty();

/** Returns true if this list contains the specified element. */
  public boolean contains(Object o);

/** Returns the number of elements in this list. */
  public int size();
. . .
}
```


Beispiel: Implementierung von Listen




```
public class List implements ObjectListInterface
{ private ListElem anchor;
  ...
}
```

Nebenläufige und verteilte Systeme

- Programme, wie wir sie in der Vorlesung geschrieben haben, arbeiten sequentiell, d.h. Aktivitäten werden nacheinander ausgeführt.
- Unter einem **System** versteht man eine von seiner Umgebung abgegrenzte Anordnung von Komponenten.
- Können in einem System mehrere Aktivitäten gleichzeitig stattfinden, man von einem **parallel ablaufenden (nebenläufigen) System**.
- Ist das System aus räumlich verteilten Komponenten aufgebaut, spricht man von einem **verteilten System**.

Beispiel: Banktransaktion

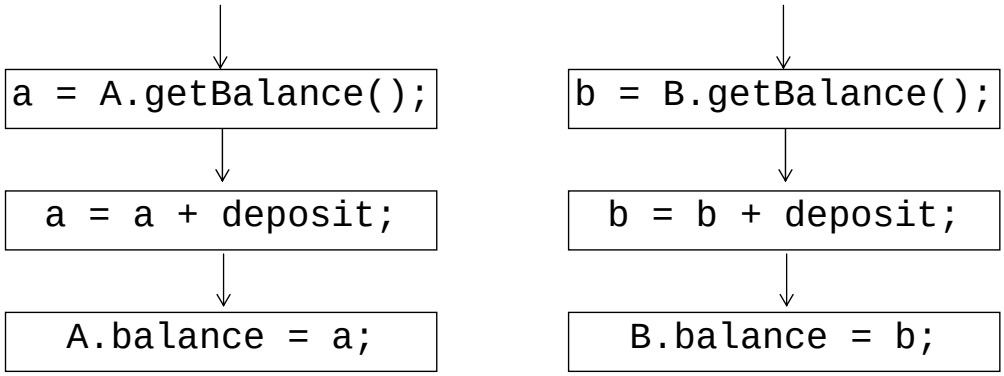
Einzahlung auf ein Konto: Der Stand eines Kontos A wird abgefragt, *eine Summe „deposit“ eingezahlt*, der neue Wert des Kontos berechnet und der neue Kontostand eingetragen:

```
a = A.getBalance();      // erfragte Kontostand a von Konte A
        ↓
  a = a + deposit;       // erhöhe a
        ↓
  A.balance = a;         // setze Kontostand auf neuen Wert von a
```

Bemerkung: Auch reale Bankautomaten und Computerprogramme laufen in solchen Sequenzen ab, die man „Thread“ (Kontrollfluß) nennt. Das obige Prigramm ist ein „**single-threaded**“ Programm.

Beispiel: Banktransaktion

In einer realen Bank können Kontoeinzahlungen parallel laufen:



Bemerkung: Innerhalb eines Computers wird dies „**multi-threading**“ genannt. Ein Thread kann eine Aufgabe unabhängig von anderen Threads erfüllen. Ebenso wie zwei Bankautomaten die gleichen Konten benutzen können, können Threads Objekte gemeinsam benutzen.

Threads

- In Java wird Nebenläufigkeit durch **Threads** realisiert. Ein nebenläufiges Java-Programm besteht aus **mehreren Threads**, die über **gemeinsame Objekte** miteinander kommunizieren.
- Ein **Thread** ist ein Teil eines Programms, das unabhängig von von anderen Teilen des Programms ausgeführt werden kann. Es repräsentiert eine einzige Sequenz von Anweisungen, d.h. einen sequentiellen Kontrollfluss, der nebenläufig zu anderen Threads ausgeführt werden kann, und der Daten gemeinsam mit anderen Threads benutzt.
- Aus Betriebssystemsicht wird ein **Prozess** verstanden als eine **abstrakte Maschine, die eine Berechnung ausführt**. Da ein Thread im Allgemeinen zusammen mit anderen Threads auf der gleichen abstrakten Maschine läuft und Ressourcen mit anderen Threads teilt, ist ein Thread ein „**leichtgewichtiger Prozess**“.

Software Engineering

- Software Engineering ist die Disziplin der systematischen Software-Entwicklung,

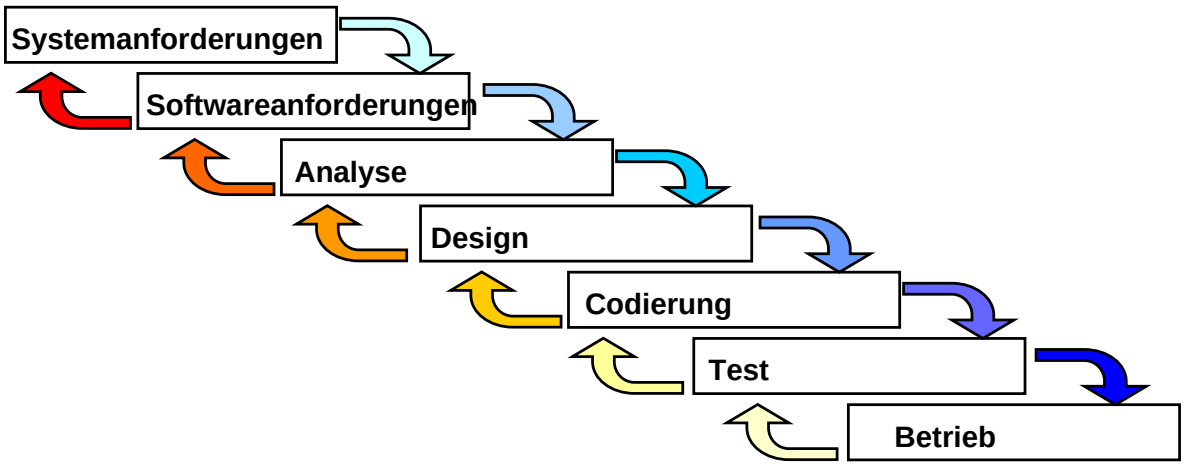
  dies bedeutet

  > die Bereitstellung und systematische Anwendung von Methoden, Verfahren und Werkzeugen zur Entwicklung, Betrieb und Wartung von Software. [***IEEE Std. 610.12 (1990)***]

- Software Engineering umfasst neben SW-Entwicklungstechniken auch Fragen der Managements, der Kosten, der Qualität und der Teamarbeit.

Software Engineering

- Eine klassische (heute vielfach kritisierte) Vorgehensweise ist das **Wasserfallmodell** mit den folgenden Phasen, die in den meisten Vorgehensweisen vorkommen:



- In der Vorlesung wurden nur Codierung sowie etwas Design und Test betrachtet.

Ausblick 1: Objekt-orientierte Programmierung und andere Programmierparadigmen

- Ein objekt-orientiertes Programm beschreibt die **Änderung lokaler Zustände** und berechnet Werte ebenfalls mittels Änderung lokaler Zustände.
- **Beispiel:**

```
class Square
{  int data;
   public int sumSquares()
      {  int s = 0; int i = 0;
         while (i < data) {i++; s =+ i*i; }
         return s;
      }
}
```

Attribut: Beschreibt **lokalen Zustand eines Square-Objekts** auf der Halde

Lokale Variablen im Keller; Keller ist global für das gesamte Programm

Vergleich mit imperativer Programmierung

- Imperative Programmierung basiert auf dem Konzept der **Anweisung.**
- Ein imperatives Programm arbeitet auf einem globalen Speicher und beschreibt, wie der **globale Speicher modifiziert** wird.
- Imperative Sprachen können **modular** sein, besitzen aber **keine Vererbung**.
- Imperative Programmiersprachen sind Pascal, C, FORTRAN.
- **Beispiel:**

```
int data;
public static int sumSquares() {
   int s = 0, i = 0;
   while (i < data){ … }; return s;
}
```

Verwendet und verändert möglicherweise globale Variablen

Vergleich mit funktionaler Programmierung

- Funktionale Programme basieren auf dem Konzept der **mathematischen Funktion**.
- Ein funktionales Programm beschreibt den **Zusammenhang von Argument und Wert einer Funktion**. Berechnungen hängen **NICHT** vom Zustand des Systems ab, sondern nur von den Werten der (aktuellen) Parameter.
- Funktionale Programmiersprachen sind SML, Haskell, Gofer, CAML.
- **Beispiel:**

```
public static int sumSquares(int data) {
   int s = 0, i = 0;
   while (i < data){ … }; return s;
}
```

Verwendet und verändert möglicherweise globale Variablen

Ausblick 2: Neue Entwicklungen in Programmiersprachen

Neue Entwicklungen in Programmiersprachen

- **Aspekt-orientierte Progammierung** (AspectJ, HyperJ, ...)

  „Einweben“ von Aspekten:

  Z.B. Logging, Zusicherungen,

  Synchronisation beim Einbetten sequentieller in nebenläufige Programme
- **Agenten-orientierte Programmierung** (Jade, Aglets, ...)

  Autonome, reaktive und proaktive Programme
- **Dienst-orientierte Programmierung**

  Programme bestehen aus Diensten, die (im Web) veröffentlicht, gesucht und dynamisch kombiniert werden können.

Ausblick 3: Neue Entwicklungen im Software Engineering

- **UML**
  - Extreme Programming & UML
  - Model-driven Architecture
    - Generieren von Code aus den Modellen
    - Modelchecking von Software Entwürfen
- **Web-Engineering und Dienstentwicklung**
  - Entwurf von Web-basierten Systemen
  - Entwicklung von Web-Diensten

Ausblick: Lehre

Einführung in die Informatik: Systeme und Anwendungen

- Relationale Datenbanksysteme
- Grundlagen der Betriebssysteme
- Grundlagen des Internet: Netze, XML

Weiterführende Veranstaltungen:

- Datenbanksysteme I und II (Vertiefung)
- Methoden der Software-Engineering (WS 06/07)
- Objekt-orientierte Software-Entwicklung . . .

Vielen Dank für Ihre Aufmerksamkeit und Mitarbeit!

Michael Barth, Fabian Birzele, Gefei Zhang und ich wünschen Ihnen

viel Erfolg in der Klausur und eine **angenehme vorlesungsfreie Zeit!**

Auf Wiedersehen!