

Programmierwerkstatt



Objektorientierung und
Korrektheit von Programmen



Zum Inhalt

- Wir wollen Euch:
 - das Wesentliche vermitteln
 - Fehlerquellen verdeutlichen
 - Verständnis ist uns wichtig
 - „programming by coincidence“ vermeiden
- Themen dieses Vortrags sind:
 - Vererbung von Klassen
 - Sinn von Information-Hiding
 - Überschreiben und Überladen von Methoden
 - Korrektheit von Programmen (Hoare-Kalkül)
 - Debugging von Programmen



Vererbung & abstrakte Klassen

- Sinn von Vererbung
 - bessere Modellierbarkeit der realen Welt
 - Wiederverwendbarkeit
 - Übersichtlichkeit
- sparen von Schreibarbeit nicht Hauptzweck
- Abstrakte Klasse kann nicht instanziiert werden
 - mindestens eine virtuelle Methode
- Programmbeispiel (Tiere)



Overloading & Overriding

- Overloading (Überladen)
 - gleicher Methodename mit unterschiedlicher Signatur
 - unterschiedliche Parameterübergabe an „gleiche“ Methode möglich (konsistentes Verhalten)
- Overriding (Überschreiben)
 - Anpassung geerbter Methoden an modifiziertes Objektverhalten



this

- this ist eine Selbstreferenz auf das aktuelle Objekt
- Sonst gibt's nix mehr zu sagen!



Zugriffsrechte

	Klasse	Subklassen	alle	Freunde
private	X	-	-	X
protected	X	X	-	X
public	X	X	X	X



Interface & Implementierung

- Aufteilung in Header und Source-Code
- Sinn: z.B. Programme schneller verständlich
- Programme kompilieren schneller



Information-Hiding (Kapselung)

- Entkopplung von Interface und Implementierung
- Sinn von Information-Hiding
 - Wechsel/Erweiterung der Implementierung (siehe Bsp.)
 - Algorithmen und Datenstrukturen ohne Probleme austauschbar
 - „sichtbare“ Schnittstelle ändert sich nicht
- Vermeidung von unerwartetem Objektverhalten
 - nur gewollte Zugriffe auf Variablen möglich
- Programmbeispiel (Zugriffsrechte)



Get & Set-Methoden

- Information-Hiding realisiert durch spezielle Zugriffsmethoden
- Schützen die privaten Attribute einer Klasse
- Sieht nach unnötigem Aufwand auf, bringt aber was!



Beispiel: Get & Set 1/3

- **Schlechtes Beispiel:**

```
class Line {  
    public:  
        Point start;  
        Point end;  
        double length() {return start.distanceTo(end);}  
}
```

- **Attribute öffentlich einsehbar**



Beispiel: Get & Set 2/3

- Gutes Beispiel:

```
class Line{  
  private:  
    Point start;  
    Point end;  
  public:  
    void setStart(Point p) { start = p; }  
    void setEnd(Point p)   { end = p; }  
    Point getStart(void)   { return start; }  
    Point getEnd(void)     { return end; }  
    double getLength() {  
      return start.distanceTo(end);  
    }  
};
```



Beispiel: Get & Set 3/3

- Erweiterung (z.B. Caching):

```
class Line{
private:
    bool changed;
    double length;
    Point start;
    Point end;
public:
    void setStart(Point p) { start = p; changed = true; }
    void setEnd(Point p)   { end = p;   changed = true; }
    Point getStart(void)   { return start; }
    Point getEnd(void)     { return end; }
    double getLength() {
        if (changed) {
            length= start.distanceTo(end);
            changed= false;
        }
        return length;
    }
};
```



Übertriebenes Get & Set

- Wann ist die Verwendung sinnvoll?
- In großen Projekten Verwendung obligatorisch
- Hier eher nicht:

```
class Vector {  
public:  
    float x;  
    float y;  
    float z;  
} // wie ein struct
```

- Implementierung ändert sich voraussichtlich nicht



Korrektheit von Programmen

- Hoare-Kalkül
 - Beispiel an der Tafel



Assert: gewollte Bruchstelle 1/2

- Anwendung:
 - Schleifeninvarianten
 - Überprüfung von Funktionsparametern
 - Überprüfung von berechneten Ergebnissen
- Crash an der Fehlerstelle - nicht irgendwo!
 - Prinzip: Crash Early



Assert: gewollte Bruchstelle 2/2

- **Beispiel:**

```
#include "assert.h"
//im Code:
int fakultaet(int z) {
    assert(z>0); //Parameter überprüfen
    int i,r;
    while (i>0) {
        r = r*i;
        i= i-1;
        assert(r >0);
    }
    return r;
}
```



Debugging

- LIVE-Demo in Visual C++
- Haltepunkte (Breakpoints)
- Schrittweises Debuggen
- Watcher: Inhalt von Datenstrukturen
- VORSICHT: potentielle Zeitfalle



Richtlinie zum Debuggen

- Warum macht das Programm nicht was es soll?
Die Logik stimmt doch!
 - Seiteneffekte
 - falsch/nicht initialisierte Variablen
 - » Debuggen
- Was genau macht das Programm?
 - » besser in Ruhe nachdenken



Literaturempfehlung

- Andrew Hunt & David Thomas (Addison-Wesley)
„The Pragmatic Programmer“
- Visual C++ Express Edition
(<http://msdn.microsoft.com/vstudio/express/visualc/download/default.aspx>)
- Hoare Kalkül
(http://wwwswt.informatik.uni-rostock.de/deutsch/Mitarbeiter/michael/lehre/pt_2001/Hoare_akt_008.pdf)



Fragerunde

Fragen?



Zum Schluss...

- Es ist noch kein Meister vom Himmel gefallen!
- Am Anfang ist es immer schwer in eine Programmiersprache einzusteigen
- Stellt daher bitte Fragen wenn euch etwas unklar ist
 - Wir werden schließlich dafür bezahlt!
- Erreichbar sind wir
 - persönlich in den Pools oder
 - über die Mailingliste
progwerkstatt@informatik.uni-ulm.de
- Gebt bitte auch denen Bescheid die nicht da sind



...Danke für die Aufmerksamkeit!

**Viel Erfolg bei
der PI-Klausur!!!**