Informatik IV SS 03

11. April 2003

Kapitel 3, 7 & 11 im Buch gelesen?
`http://www.goepps.de/download/CSlernen.zip`

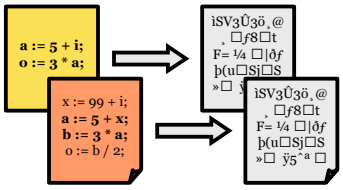
Was wir heute tun

- 20' Was sind Objekte / Klassen?
- 10' Polymorphie / Vererbung
- 10' Ereignisse
- 15' Vorbereitung Ü1/2
- 20' Was ist .NET?
- 5' Namensräume
- 10' Interfaces & Delegates
- 15' Fehlerbehandlung

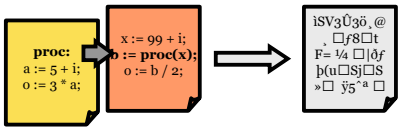
total 105 min, also keine Pause

Wie man früher programmierte

Für jede Aufgabe von Grund auf programmieren

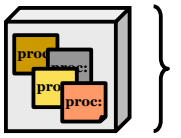


Mit Hilfe von Prozeduren Code wiederverwenden

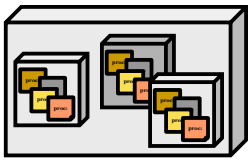


Tendenz zur Wiederverwendung

Aufgaben / Probleme werden in Teilaufgaben gesplittet & getrennt gelöst



Teilproblem



Problem mit verschiedenen Teilproblemen gelöst (selbst wieder ein Teilproblem)

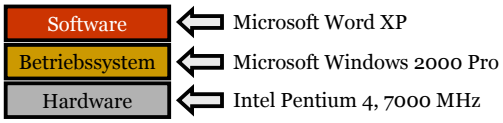
Entwicklung der Sprachen

Sprache	Stufe der Wiederverwendung
Assembler	explizit keine, Prozedurmarken möglich
Basic, C, Pascal	Prozeduraufrufe
C++	lasch objektorientiert
AOS, Eiffel, C#, Java	stark objektorientiert

Trend:
- zunehmend objektorientiert
- zunehmend abstrahiert von Hardware

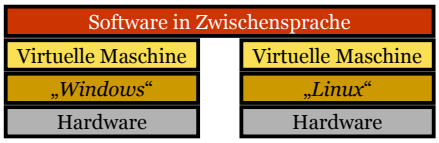
Virtuelle Maschinen

Schichtenprinzip



schön, aber Word funktioniert nicht auf Sun Solaris

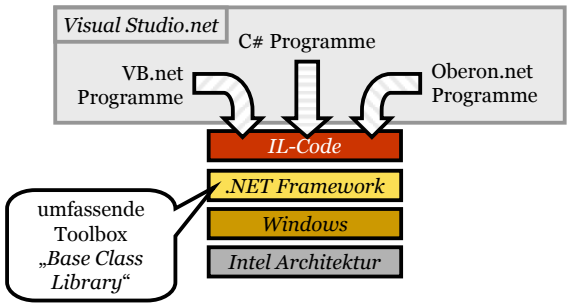
deshalb eine Software für alle Betriebssysteme, welche dafür vorgesehene Software laufenlassen kann



Was ist .NET?

→18~21

.NET ist eine *virtuelle Maschine*



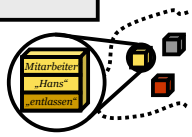
Was sind Objekte / Klassen? →51

Ein Objekt ist eine Sammlung zueinander in Beziehung stehender Informationen und Funktionen

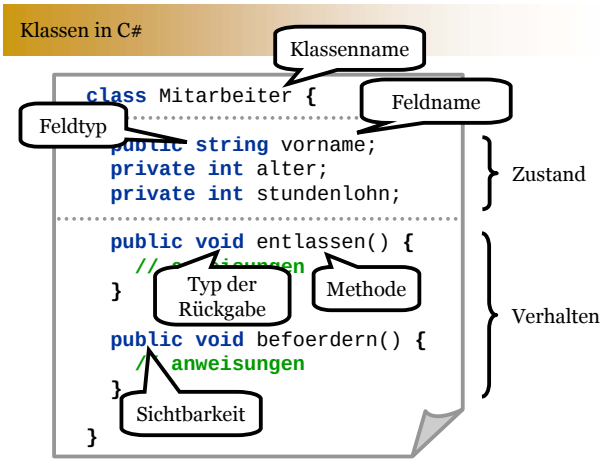
Zur Kompilierzeit werden *Klassen deklariert*

Eine Klasse definiert eine Menge von Objekten, die eine gemeinsame Struktur haben und ein gemeinsames Verhalten zeigen

Aus diesen Klassen kann zur Laufzeit ein *Objekt instanziert* werden



Klassen in C#



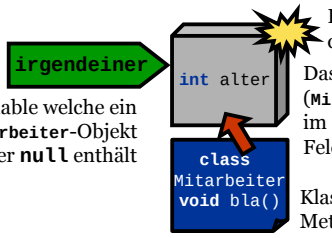
Objekte instanzieren →53

die Klasse definiert, wie alle **`Mitarbeiter`** aussehen & sich verhalten (eine Vorlage aller **`Mitarbeiter`**)

nun möchte ich einen neuen Mitarbeiter

→ **`Mitarbeiter`**-Objekt instanzieren

```
Mitarbeiter irgendeiner = new Mitarbeiter();
```



Variable welche ein **`Mitarbeiter`**-Objekt oder **`null`** enthält

Konstruktor, der das Objekt initialisiert

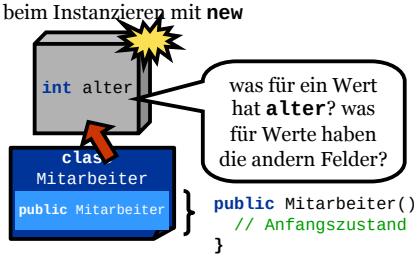
Das **`Mitarbeiter`**-Objekt (**`Mitarbeiter`**-Instanz) im Speicher, welches alle Felder enthält

Klassendefinition mit Methoden

Der Konstruktor

→88

beim Instanzieren mit **new**



der Konstruktor – spezielle Methode – wird zuerst aufgerufen:

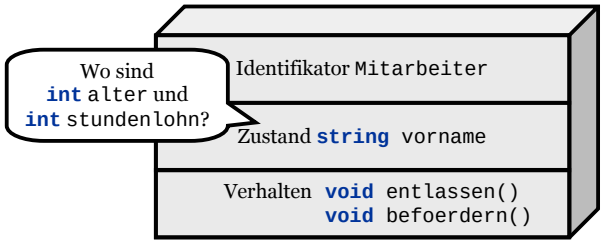
- Anfangszustand definieren

beim Löschen des Objektes Destruktor

```
public ~Mitarbeiter() { ... }
```

Sichtbarkeit

auf was kann ich zugreifen?

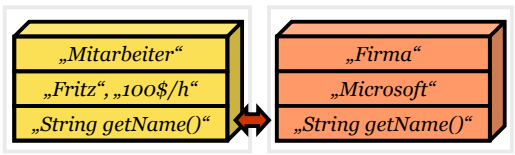


- einzelne Elemente sind unsichtbar!
- es kann nicht darauf zugegriffen werden
- wie definieren wir das?

Klassen und ihr Gültigkeitsbereich

→92

zwei Methoden im ganzen Programm mit gleichem Namen



was passiert, wenn ich **getName()** ausführe?

die beiden Klassen Mitarbeiter und Firma haben ihren eigenen *Namespace*:

- **a = einMitarbeiter.getName();**
- **b = eineFirma.getName();**

explizite Namensräume in C#

```
namespace MeineKleineWelt {
  // dinge, welche in meine welt gehören
}
```

Namespaces können auch verschachtelt werden

```
namespace MeineKleineWelt {
  namespace Ost {
    // dinge, die in meiner welt östlich liegen
  }
  namespace West {
    // dinge, welche in meiner welt westlich liegen
    class Restaurant { ... };
  }
}

a = MeineKleineWelt.West.Restaurant.getName();
```

Sichtbarkeit in C# →59

Sichtbarkeit gesteuert durch Modifikator

Modifikator	wo sichtbar / wie verwendbar?
`public`	überall
`internal`	nur im eigenen Projekt
`readonly`	im eigener Klasse voller Zugriff, von aussen nur lesen
`protected`	nur in eigener Klasse und in davon vererbten Klassen
`private`	nur in eigener Klasse

noch andere Modifikatoren (später)

Verschiedene Ansichten

in eigener Klasse

```
class Mitarbeiter {

  readonly string vorname;
  internal int alter;
  private int stundenlohn;
  protected int koerpergroesse;

  public void entlassen() {
    // anweisungen
  }
  public void befoerdern() {
    // anweisungen
  }

}
```

in anderer Klasse im gleichen Projekt

```
string vorname;
int alter;
void entlassen()
void befoerdern()
```

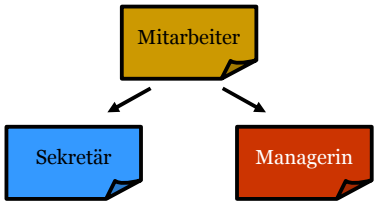
in anderer Klasse ausserhalb Projekt

```
string vorname;
void entlassen()
void befoerdern()
```

Objekte / Klassen erweitern

→207

in einer Firma gibt es verschiedene Mitarbeiter
also braucht man auch verschiedene Mitarbeiter-Klassen

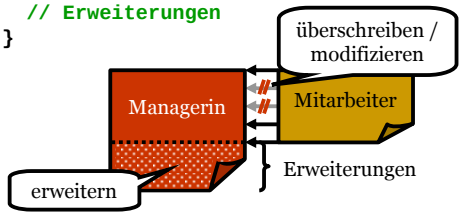


Managerinnen können auch entlassen werden!

→ Funktionalität erweitern

Funktionalität vererben

```
class Managerin : Mitarbeiter {
  // Erweiterungen
}
```



nimm die Mitarbeiter-Klasse und

- erweitere sie mit neuen Zuständen & neuem Verhalten
  ```
  new public void teileSekretaerZu( Sekretaer s );
  ```
- modifiziere bestehendes Verhalten
  ```
  override public void befoerdern();
  ```

Vererbung in C#

```
class Mitarbeiter {
  public int alter;
  public Mitarbeiter(int dasalter) { alter = dasalter; }
  public virtual void entlassen() {
    // funktioniert so und so...
  }
}

class Managerin : Mitarbeiter {
  public override void entlassen() {
    // funktioniert hier ganz anders...
  }
  public new void teileSekretaerZu( Sekretaer s );
}

Managerin maya = new Managerin( 25 );
Sekretaer hans = new Sekretaer( 32 );
maya.entlassen();
maya.teileSekretaerZu( hans );
```

Basis aufrufen

mit **base** kann man auf die Basis-Klasse zugreifen

```
class Mitarbeiter {
  public int alter;
  public Mitarbeiter(int dasalter) { alter = dasalter; }
  public virtual void entlassen() {
    // funktioniert so und so...
  }
}
class Managerin : Mitarbeiter {
  public override void entlassen() {
    // funktioniert so und so...
    base.entlassen();
    // ... und dann noch das hier
  }
  public new void teileSekretaerZu( Sekretaer s );
}

Managerin maya = new Managerin( 25 );
maya.entlassen();
```

Abstrakte Klassen

→217

die Mitarbeiterklasse hat beispielsweise eine Methode
`string getBeruf()`
wie ist die beim Mitarbeiter deklariert?
gibt es in der Firma überhaupt Mitarbeiter, die nur Mitarbeiter, aber nicht Elektriker, Managerin, ... ist?

Mitarbeiter ist also ein abstraktes Gebilde

```
abstract class Mitarbeiter {
  public int alter;
  public Mitarbeiter(int dasalter) { alter = dasalter; }
  public virtual void entlassen() {
    // funktioniert so und so...
  }
  public abstract string getBeruf();
}
```

Mitarbeiter muss jetzt durch Spezialisierung / Vererbung erweitert werden, dient also als Grundgerüst

Klassen versiegeln

gelegentlich möchte man, dass eine Klasse nicht mehr weiter abgeleitet wird

versiegelte Klassen können nicht mehr abgeleitet werden

```
sealed class Managerin {
  // ...
}
```

→ darf keine abstrakten Methoden beinhalten