

```

public class Student {
    public enum Studienfach {WI, TI, SEB};
    private Studienfach studifach;
    public Student (Studienfach sf){
        this.studifach = sf;
    }
}

```

Methode in anderer Klasse

```

Student studi = new Student(Student.Studienfach.WI);
=====

```

```

public enum Studienfach {
    WI (int semester){
        String inEnglish(){
            return "economics";
        }
    },
    TI (int semester),
    SEB (int semester);
    public abstract String inEnglish();
    public Studienfach(int semester) {
        ...
    }
};

```

## Vererbung (Generalisierung, Konkretisierung)

Konstruktoren werden (im Gegensatz zu Methoden) nicht vererbt.

Hat die Superklasse einen nicht trivialen Konstruktor, muss auch die Subklasse einen Konstruktor haben, der den der Superklasse aufruft.

Soll die Methode einer Superklasse nicht überschrieben werden dürfen, so muss sie in der Superklasse mit `final` gekennzeichnet sein.

Muss die Methode einer Superklasse überschrieben werden, so muss sie als `abstract` gekennzeichnet sein. Damit wird die Superklasse insgesamt `abstract`.

```
package vererbung;

public class StartVererbung {
    public static void main(String[] args) {
        SuperKlasse superklasse = (SuperKlasse) new SubKlasse();
        superklasse.sagwas();
    }
}
```

Problems @ Javadoc Declaration Console

<terminated> StartVererbung [Java Application] C:\Programme\Java\jdk1.6.0\_14\bin\javaw.exe (16.10.2009 08:  
Bin die Subklasse

Auch bei einem "Upcast" erinnert sich der Compiler, was die ursprüngliche Methode ist und ruft die spezifischere Methode (also die der Subklasse) auf. Es stehen aber nur die Methoden zur Verfügung, die in der Superklasse definiert sind.

```
package vererbung;

public class StartVererbung {
    public static void main(String[] args) {
        SubKlasse klasse = (SubKlasse)new SuperKlasse(2);
        klasse.sagwas();
    }
}

<terminated> StartVererbung [Java Application] C:\Programme\Java\jdk1.6.0_14\bin\javaw.exe (16.10.2009 08:45:34)
Exception in thread "main" java.lang.ClassCastException: vererbung.SuperKlasse ca
at vererbung.StartVererbung.main(StartVererbung.java:5)
```

Ein Downcast führt zu einer ClassCastException. dies lässt sich durch eine Typprüfung vermeiden:

```
if(klasse instanceof SubKlasse){
    SubKlasse subklasse = (SubKlasse)klasse;
    klasse.sagwas();
} else {
    System.out.println("Glück gehabt, ClassCastException vermieden");
}
```

# Modifier

## Sichtbarkeit:

- public: Attribute/Methoden/Klassen überall sichtbar
- private: nur innerhalb umgebender Klasse sichtbar
- protected: In eigenem Package und erbenden Klassen sichtbar

## Überschreibbarkeit

- final: Methoden nicht überschreibbar, Attribute nicht änderbar
- abstract: Abstrakte Methoden müssen von erbenden überschrieben werden

## Sonstige

### static:

Methoden: Alle Instanzen der Klasse nutzen die gleiche Methode (Klasse muss nicht instanziiert werden).

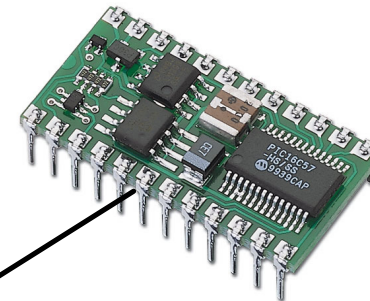
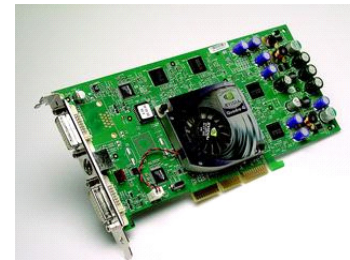
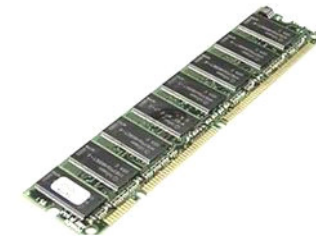
Attribute: Alle Instanzen der Klasse nutzen den gleichen Wert

# Interfaces

Mit Interfaces (Schnittstellen) kann man sicherstellen, dass alle implementierenden Klassen, genau über diese Schnittstelle verfügen, d.h. genau diese Methoden implementieren.

Gründe für eine  
Komponentenorientierung

- Wartbarkeit
- Austauschbarkeit
- Wiederverwendbarkeit
- Testbarkeit



Schnittstelle spezifiziert hier

- Position der Pins
- Anzahl der Pins
- Spannungspegel

Diese Klasse Computer hat keine Abhängigkeit mehr zu einer implementierenden Klasse (speziellen Grafikkarte):

```
package interfaces;

public class Computer {
    private IGraphikkarte karte;

    public Computer(IGraphikkarte karte) {
        this.karte = karte;
    }

    public void zeigeBildan(){
        karte.machbild();
    }
}
```

Damit ist der Computer völlig von einer konkreten Grafikkarte entkoppelt!

"Dependency Injection"

```
package interfaces;

public interface IGraphikkarte {
    void machbild();
}
```

Nvidia implementiert die Schnittstelle IGraphikkarte

```
package interfaces;

public class Nvidia implements IGraphikkarte{
    public void machbild(){
        System.out.println("Ich bin die Nvidia und mach ein super Bild!");
    }
}
```

# Interfaces und abstrakte Klassen

## Gemeinsamkeiten

- Zwingen erbende/instanzierende Klasse, Methoden zu implementieren
- Beide können nicht instanziiert werden
- beide können abstrakte Methoden enthalten (bei Interfaces müssen diese aber nicht als solche gekennzeichnet werden)

## Unterschiede

- Abstrakte Klasse können Methoden implementieren, Interfaces nicht
- unterschiedliche Schlüsselworte (extends, implements)
- Nur von einer abstrakten Klasse kann geerbt werden, es können aber beliebig viele Interfaces implementiert werden.