Whitebox-Vererbung vs. Blackbox-Vererbung

- Begriffsbestimmung
- Vererbung
    - öffentliche Vererbung
    - private Vererbung
- Zusammenfassung

Begriffsbestimmung

Whitebox- oder **Glassbox**-Testverfahren

- auch Strukturtest-Verfahren genannt
- basieren auf der Kontrollstruktur des zu testenden Programms
- die *Implementierung* des Programms muss also *bekannt* sein

Blackbox-Testverfahren

- auch funktionale Testverfahren genannt
- die Testfälle und Testdaten werden allein aus der Programmspezifikation abgeleitet
- die *Implementierung* des Programms bleibt für den Tester *unsichtbar*

Whitebox-Vererbung

- Vererbung der *Implementierung*

Blackbox-Vererbung

- Vererbung der *Schnittstelle*

Vererbung

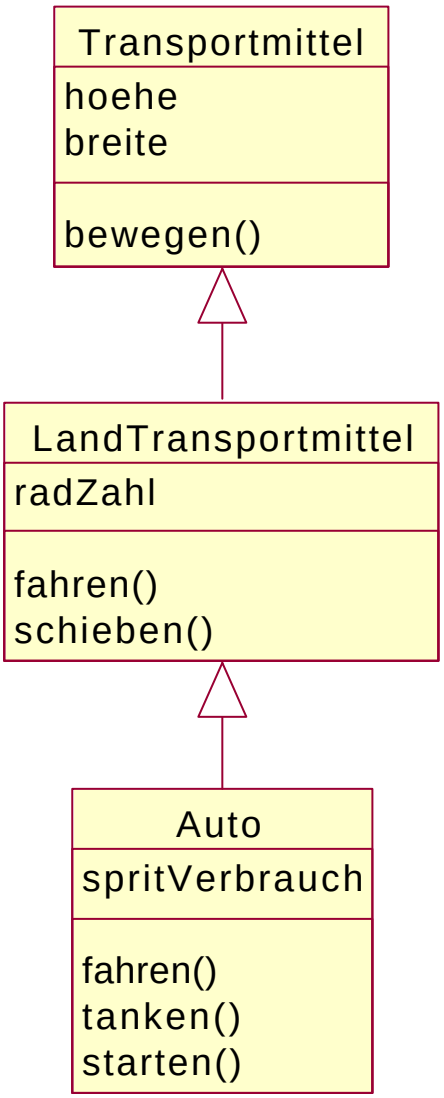
Eine **Unterklasse** *erbt* von der **Oberklasse**

- die Eigenschaften (Daten) und
- das Verhalten (die Operationen)

Wenn eine Oberklasse bekannt ist, brauchen in einer zugehörigen Unterklasse nur die *Abweichungen* beschrieben zu werden.

Alles andere kann *wiederverwendet* werden, weil es in der Oberklasse bereits vorliegt.

Beispiel:



Vererbung

Beispiel (Fortsetzung): C++-Code

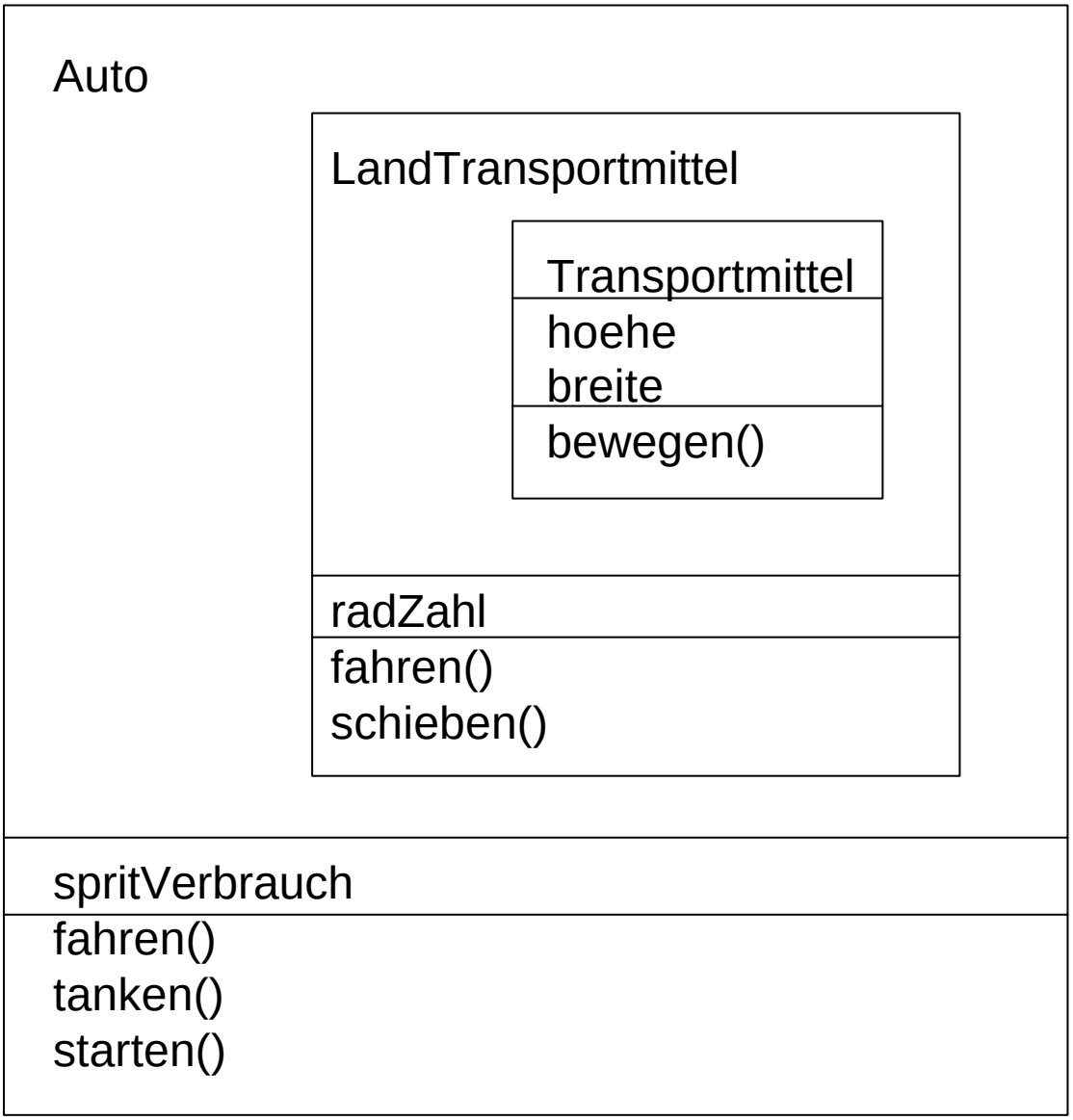
```
class Transportmittel
{
  public:
    bewegen();
  private:
    double hoehe, breite;
};

class LandTransportmittel
  : public Transportmittel
{
  public:
    fahren();
    schieben();
  private:
    int radAnzahl;
};

class Auto : public LandTransportmittel
{
  public:
    fahren();
    tanken();
    starten();
  private:
    double spritVerbrauch;
};
```

Vererbung

Beispiel (Fortsetzung): Einschluss von Subobjekten



Öffentliche Vererbung

Öffentliche Vererbung bedeutet in C++ "ist ein".

```
class Abgeleitet : public Basis
```

Abgeleitet ist eine *Erweiterung* von Basis:
Jedes Objekt vom Typ Abgeleitet ist auch
ein Objekt vom Typ Basis, aber nicht umgekehrt.

Beispiel:

```
class Person
{ ... public: void dance(); ... };

class Student : public Person
{ ... public: void study(); ... };

Person p;         // p ist eine Person
Student s;        // s ist ein Student

p.dance();        // ok: p ist eine Person
s.dance();        // ok: s ist ein Student, und
                  // ein Student ist eine Person

s.study();        // ok
p.study();        // Fehler: p ist kein Student
```

Öffentliche Vererbung

Öffentliche Vererbung besteht aus zwei Komponenten:

- Vererbung von Funktions-*Schnittstellen* (Blackbox-Vererbung)
- Vererbung von Funktions-*Implementierungen* (Whitebox-Vererbung)

Beispiel:

```
class Shape
{
  public:
    virtual void draw() const = 0;
    virtual void error(const char *msg);
    int objectId() const;
  ...
};

class Rectangle : public Shape { ... };

class Ellipse : public Shape { ... };
```

Drei unterschiedliche Deklarationen:

- `draw` ist eine rein virtuelle Funktion,
- `error` eine "normale" virtuelle Funktion und
- `objectId` eine nicht-virtuelle Funktion.

Öffentliche Vererbung

Die Deklaration einer **nicht-virtuellen** Funktion bedeutet, dass abgeleitete Klassen *sowohl die Schnittstelle als auch eine obligatorische Implementierung* erben.

```
int objectId() const;
```

Geerbte nicht-virtuelle Funktionen sollten *niemals* überschrieben werden.

Beispiel:

```
class Basis { ... public: void mf(); ... };

class Abgeleitet : public Basis
{ ...
  public:
    void mf();        // überdeckt Basis::mf
  ...
};

Abgeleitet x;
Abgeleitet *pAbgeleitet = &x;  // Zeiger auf x
Basis *pBasis = &x;  // Zeiger auf x

x.mf();              // ruft Abgeleitet::mf auf
pAbgeleitet->mf();   // ruft Abgeleitet::mf auf
pBasis->mf();        // ruft Basis::mf auf
```

Öffentliche Vererbung

Die Deklaration einer **einfachen virtuellen** Funktion bedeutet, dass abgeleitete Klassen *die Schnittstelle* der Funktion *wie auch eine Standardimplementierung* erben.

```
class Shape
{
  public:
    virtual void error(const char *msg);
  ...
};
```

Öffentliche Vererbung

Es ist manchmal gefährlich, für virtuelle Funktionen auch eine Implementierung bereitzustellen.

```
class Airport { ... };    // rep. Flughäfen

class Airplane {
  public:
    virtual void fly(
      const Airport& destination)
    {
      // Standardcode für das Anfliegen
      // des angegebenen Flughafens
    }
};

class ModelA : public Airplane { ... };

class ModelB : public Airplane { ... };

class ModelC : public Airplane {
  // Modell C wird völlig anders geflogen,
  // aber keine fly-Funktion deklariert!
  ...
};

Airport SFO(...);
Airplane *pa = new ModelC;
pa->fly(SFO);    // ruft Airplane::fly() auf!
```

Öffentliche Vererbung

Lösung des Problems:

```
class Airplane {
  public:
    virtual void fly(const Airport& destination) = 0;
};

void Airplane::fly(const Airport& destination) {
  // Standardcode für das Anfliegen
  // des angegebenen Flughafens
}

class ModelA : public Airplane {
  public:
    virtual void fly(const Airport& destination) {
      Airplane::fly(destination);
    }
  ...
};

class ModelB : public Airplane {
  public:
    virtual void fly(const Airport& destination) {
      Airplane::fly(destination);
    }
  ...
};

class ModelC : public Airplane {
  public:
    virtual void fly(const Airport& destination) {
      // wie ein Modell C den Zielflughafen anfliegt
    }
  ...
};
```

Öffentliche Vererbung

Die Deklaration einer **rein virtuellen** Funktion bedeutet, dass abgeleitete Klassen *ausschließlich die Schnittstelle* der Funktion erben.

```
class Shape
{
  public:
    virtual void draw() const = 0;
  ...
};
```

Beispiel:

```
Shape *ps = new Shape;      // Fehler: Shape
                            // ist abstrakt

Shape *ps1 = new Rectangle;   // ok
ps1->draw();    // ruft Rectangle::draw auf

Shape *ps2 = new Ellipse;     // ok
ps2->draw();    // ruft Ellipse::draw auf
```

Öffentliche Vererbung

Reine Schnittstellenvererbung oder **Blackbox-Vererbung** können wir durch *öffentliches Erben von abstrakten Basisklassen* erreichen.

Beispiel:

```
class Person
{
  public:
    virtual ~Person();
    virtual const char *name() const = 0;
    virtual const char *address() const = 0;
    virtual const char *phone() const = 0;
};

class Student : public Person
{
  ...
};
```

Private Vererbung

Private Vererbung bedeutet in C++ "ist implementiert mit".

```
class Abgeleitet : private Basis
```

Abgeleitet erbt ausschließlich die *Implementierung* von Basis, nicht jedoch die Schnittstelle.

Beispiel:

```
class Person
{ ... public: void dance(); ... };

class Student : private Person
{ ... public: void study(); ... };

Person p;         // p ist eine Person
Student s;        // s ist ein Student

p.dance();        // ok: p ist eine Person
s.dance();        // Fehler: ein Student
                  // ist keine Person
```

Private Vererbung

Beispiel: ein Template für doppelt verkettete Listen

```
template<class T>
class Liste
{
  public:
    Liste();
    Liste(const Liste&);
    virtual ~Liste();
    Liste& operator=(const Liste&);

    bool empty() const { return anzahl == 0; }
    int size() const { return anzahl; }

    // am Anfang bzw. Ende einfügen
    void push_front(const T&);
    void push_back(const T&);

    // am Anfang bzw. Ende löschen
    void pop_front();
    void pop_back();

    // am Anfang bzw. Ende lesen
    T& front();
    const T& front() const;
    T& back();
    const T& back() const;

  private:
    ...
};
```

Private Vererbung

Implementierung einer Warteschlange durch **öffentliches Erben** von der Klasse Liste

```
template<class T>
class Queue : public Liste<T>
{
  public:
    // empty und size, front und back
    // werden von Liste<T> geerbt

    // am Ende einfügen
    void push(const T& x) { push_back(x); }

    // am Anfang entnehmen
    void pop() { pop_front(); }
};

// Warteschlange verwenden
Queue<int> q;
for (int i=0; i<10; i++)
  q.push(i);

q.push_front(47);  // Wird vom C++-Compiler
q.pop_back();      // akzeptiert, ist aber
                   // ein logischer Fehler.
```

Es gilt *nicht*, dass eine Warteschlange eine `Liste` ist. Einige Operationen, die für eine Liste erlaubt sind, sind für Warteschlangen nicht erlaubt.

Private Vererbung

Implementierung einer Warteschlange durch **privates Erben** von der Klasse Liste

```
template<class T>
class Queue : private Liste<T>
{
  public:
    using Liste<T>::empty;
    using Liste<T>::size;

    // am Ende einfügen
    void push(const T& x)
    { Liste<T>::push_back(x); }

    // am Anfang entnehmen
    void pop()
    { Liste<T>::pop_front(); }

    // am Anfang bzw. Ende lesen
    using Liste<T>::front;
    using Liste<T>::back;
};

// Warteschlange verwenden
Queue<int> q;
q.push_front(47);  // Wird vom C++-Compiler
q.pop_back();      // abgewiesen.
```

Private Vererbung

Implementierung einer Warteschlange durch **Delegation**

```
template<class T>
class Queue
{
  public:
    bool empty() const { return L.empty(); }
    int size() const { return L.size(); }

    // am Ende einfügen
    void push(const T& x) { L.push_back(x); }

    // am Anfang entnehmen
    void pop() { L.pop_front(); }

    // am Anfang bzw. Ende lesen
    T& front() { return L.front(); }
    const T& front() const { return L.front();}
    T& back() { return L.back(); }
    const T& back() const { return L.back(); }

  private:
    Liste<T> L;
};
```

Zusammenfassung

Whitebox-Vererbung vs. Blackbox-Vererbung

Öffentliche Vererbung bedeutet "ist ein" (im Sinne einer Erweiterung der Basisklasse).

```
class Abgeleitet : public Basis
```

Private Vererbung bedeutet "ist-implementiert mit".

```
class Abgeleitet : private Basis
```

Delegation bedeutet "hat ein".

```
class Queue {
  ...
  private:
    Liste<T> L;
};
```

Reine Implementierungsvererbung oder **Whitebox-Vererbung** können wir durch *privates Erben* erreichen.

Zusammenfassung

Für die *öffentliche* Vererbung gilt:

Eine *rein virtuelle* Funktion bedeutet, dass nur die Schnittstelle der Funktion geerbt wird.

```
virtual void draw() const = 0;
```

Eine *einfache virtuelle* Funktion bedeutet, dass die Schnittstelle und eine Standardimplementierung der Funktion geerbt wird.

```
virtual void error(const char *msg);
```

Eine *nicht-virtuelle* Funktion bedeutet, dass die Schnittstelle plus einer obligatorischen Implementierung geerbt wird.

```
int objectId() const;
```

Reine Schnittstellenvererbung oder **Blackbox-Vererbung** können wir durch *öffentliches Erben von abstrakten Basisklassen* erreichen.

```
class Person {
  public:
    virtual ~Person();
    virtual const char *name() const = 0;
    virtual const char *address() const = 0;
    virtual const char *phone() const = 0;
};
class Student : public Person { ... };
```

Literaturhinweise

Breymann, Ulrich:
C++ Eine Einführung.
8. Auflage, Hanser, München, 2005

Gamma, Erich u. a.:
Entwurfsmuster.
Addison-Wesley, München, 1996

Meyers, Scott:
Effektiv C++ programmieren.
Addison-Wesley, München, 1998