Programmieren:

Vererbung

Inhaltsverzeichnis:

1. Was ist Vererbung

Von der Klasse Person gibt es mehrere Spezialisierungen. Student ist also eine solche Spezialisierung, denn ein Student ist eine spezielle Person. Ein Angestellter ist auch eine spezielle Person. So wird Student mit Person vererbt, Angestellter auch mit Person vererbt (Erbungspfeil in UML).

2. Anwendung

Um Verwechslungen zwischen „hat als Teil“ und Verallgemeinerung und Spezialisierung zu vermeiden, muss man sich folgende Fragen stellen:

Was tut eine Spezialisierung?

- Sie fügt zur Verallgemeinerung Attribute und Methoden hinzu.
- Sie kann die Methoden der Verallgemeinerung ändern (überschreiben).

Was tut eine Spezialisierung nicht?

- Sie nimmt nichts von der Verallgemeinerung weg, denn das würde dem Test widersprechen: da eine Spezialisierung eine Verallgemeinerung ist, hat sie alle Eigenschaften der Verallgemeinerung, alle Attribute und Methoden.
- Wenn sie ein Attribut oder eine Methode nicht hat, so liegt vermutlich keine Vererbung vor, jedenfalls keine zwischen diesen beiden Klassen. Möglicherweise gibt es aber eine gemeinsame Oberklasse der beiden Klassen.

3. Realisierung

Deklaration von Vererbten Klassen:

```
class Fahrzeug {...};              //Basisklasse, Vater-/Elternklasse
class Strassenfahrzeug : public Fahrzeug {...};      //vererbte Klasse
                                                     //von Fahrzeug
class Wasserfahrzeug : public Fahrzeug {...};
class Auto : public Strassenfahrzueg {...};          //vererbte Klasse
                                                     //von Strassenfahrzeug
class Schiff : public Wasserfahrzeug {...};
```

4. Vorgehensweise zur Erstellung einer Kind-Klasse

1. Neue Attribute festlegen
2. Neue Methoden / Operatoren festlegen
3. Überschreiben von Methoden / Operatoren der Elternklasse
4. Konstruktoren / Destruktoren

5. Sichtbarkeit von Attributen und Objekten

public

public Methoden (sollte für Attribute nicht verwendet werden) sind öffentlich, d.h. sie sind ausserhalb der Klasse sichtbar. Das bedeutet, die Methoden können an beliebiger Stelle in einem Programm aufgerufen werden.

protected

protected Methoden (auch protected sollte für Attribute nicht verwendet werden) sind nur innerhalb der Klasse selbst und innerhalb von abgeleiteten Klassen (Kindklasse) sichtbar, nicht aber ausserhalb der Klasse.

private

private Methoden und Attribute sind ausschliesslich in der Klasse selbst sichtbar. Sie sind auch für abgeleitete Klassen nicht sichtbar.

6. Sichtbarkeit bei der Vererbung

Die Schlüsselworte public, private und protected werden auch bei der Festlegung der Oberklasse verwendet. z.B

```
class Auto : public Strassenfahrzeug { ... }
```

Mit dieser Verwendung der Schlüsselworte kann die Sichtbarkeit der Komponenten der Oberklasse eingeschränkt werden:

- public bedeutet keine Einschränkung, d.h.
  - public Komponenten der Oberklasse sind auch in der abgeleiteten Klasse public
  - protected Komponenten der Oberklasse sind auch in der abgeleiteten Klasse protected
- protected beschränkt die Sichtbarkeit in der abgeleiteten Klasse public, d.h.
  - public Komponenten der Oberklasse sind in der abgeleiteten Klasse protected.
- private beschränkt die Sichtbarkeit in der abgeleiteten Klasse auf höchstens private, d.h.
  - public und protected Komponenten der Oberklasse sind in der abgeleiteten Klasse private

Regel: Vererbung bedeutet ein „ist ein“ und wird mit public Vererbung realisiert!

7. Konstruktoren

Das Erzeugen von Objekten einer abgeleiteten Klasse erfordert mehr Schritte:

- es wird Speicherplatz für das „Objekt“ der Oberklasse angelegt
- es wird Speicherplatz für die zusätzlichen Attribute der abgeleiteten Klasse angelegt
- dann wird – falls vorhanden – ein Konstruktor der Oberklasse aufgerufen, der dynamische Attribute anlegt und Attribute initialisiert
- dann wird – falls vorhanden – ein Konstruktor der abgeleiteten Klasse aufgerufen, der weitere dynamische Attribute anlegt und Attribute initialisiert.

8. Überschreiben von Methoden und virtual

Der Aufruf der geerbten Methode in der überschriebenen Methode

- ist notwendig, um auf die Attribute der Oberklasse zuzugreifen
- ist sinnvoll, denn die Vererbung wird wirklich genutzt, nämlich die vorhandene Methode
- garantiert, dass die überschriebene Methode das gleiche Verhalten wie die geerbte Methode hat

Bei überschriebenen Methoden legt der Compiler bei Funktionsaufrufen bereits beim Übersetzen fest, welche Funktion verwendet werden soll (statisches Binden). Er tut dies auf Grund des Typs der Parameter. Das ist in einigen Fällen gar nicht erwünscht.

Damit der Compiler dynamisch bindet muss man ihn dazu auffordern. Man muss für jede Methode, die dynamisch gebunden werden soll, dies festlegen. Das geschieht, indem man in der Deklaration der Oberklasse und in der abgeleiteten Klasse die Methode durch das Schlüsselwort `virtual` kennzeichnet.

```
public:
     virtual void print( int i);
```

Regeln:

- Eine nichtvirtuelle Funktion einer Oberklasse gibt eine Schnittstelle und eine obligatorische Implementierung vor, die in abgeleiteten Klassen nicht überschrieben werden soll.
- Eine virtuelle Funktion einer Oberklasse gibt eine Schnittstelle und eine Standardimplementierung vor, die in abgeleiteten Klassen überschrieben werden kann. Andernfalls wird die Standardimplementierung benutzt.

9. Destruktoren und virtual

Wenn ein Objekt einer abgeleiteten Klasse vernichtet wird, erfolgt ein Ablauf ähnlich wie beim Erzeugen, allerdings in umgekehrter Reihenfolge:

- zunächst wird der Destruktor der abgeleiteten Klasse aufgerufen, um dynamisch angelegte Attribute freizugeben
- danach wird für die Oberklasse der Destruktor aufgerufen
- dann wird der Speicherplatz der normalen Attribute der abgeleiteten Klasse freigegeben
- anschliessend wird der Speicherplatz der Oberklasse freigegeben

delete bewirkt einen Aufruf des Destruktors, auf Grund der statischen Bindung wird der Destruktor der Vaterklasse aufgerufen. Falls die Kindklasse einen eigenen Destruktor hätte, würde dieser nicht aufgerufen. Es gibt ein Speicherleck. Dieses Speicherleck wird vermieden, indem man den **Destruktor virtuell** macht.

Regeln:

- Wenn eine Klasse Basisklasse für abgeleitete Klassen ist, sollte der Destruktor virtuell sein. Es empfiehlt sich, die Destruktoren der abgeleiteten Klassen ebenfalls mit dem Schlüsselwort `virtual` zu kennzeichnen.
- Das Vorhandensein von virtuellen Funktionen ist ein (hinreichender, aber nicht notwendiger) Hinweis darauf, dass die Klasse als Basisklasse gedacht ist.

10. Polymorphe Pointer

```
Strassenfahrzeug* mein Fahrzeugpark[4];
meinFahrzeugpark[0] = &meinRennVelo;        //von Klasse Velo
meinFahrzeugpark[1] = &meinMountainBike;    //von Klasse Velo
meinFahrzeugpark[2] = &meinOpel;            //von Klasse Auto
meinFahrzeugpark[3] = &meinFiat;            //von Klasse Auto
```

Da die Methode `drucke()` virtuell gemacht wurde, wird die Methode von jeder Klasse (Auto und Velo) separat abgerufen, anstatt immer die `drucke()`-Funktion von Strassenfahrzeug.

11. Abstrakte Klassen und rein virtuelle Methoden

Abstrakte Klassen dienen lediglich dazu, Gemeinsamkeiten von mehreren abgeleiteten Klassen zu beschreiben, ohne dass die Klassen eigene Objekte haben. Dies ist immer dann der Fall, wenn die Gemeinsamkeit der abgeleiteten Klassen vor allem in ihrer Schnittstelle besteht, d.h. in den Methoden, die sie anderen Klassen zur Verfügung stellen, nicht jedoch in der Implementierung dieser Methoden.
Sobald eine Methode in der Oberklasse nicht definiert werden kann, ist die Klasse abstrakt. Eine abstrakte Klasse enthält also (mindestens) eine Methode ohne Definition (rein virtuelle Methoden).

Deklaration:

```
virtual Rückgabetyp Methodenname(Parameterliste)=0; //rein Virtuell

virtual void fahren();          //virtuell
```

Ein anderes Merkmal einer abstrakten Klasse ist, wenn der Konstruktor in protected oder private ist.

Regel:

- Eine **nichtvirtuelle** Funktion einer Oberklasse gibt eine Schnittstelle und eine obligatorische Implementierung vor, die in abgeleiteten Klassen nicht überschrieben werden soll.
- Eine **virtuelle** Funktion einer Oberklasse gibt eine Schnittstelle und eine Standardimplementierung vor, die in abgeleiteten Klassen überschrieben werden kann. Andernfalls wird die Standardimplementierung benutzt.
- Eine **rein** virtuelle Funktion einer Oberklasse gibt eine Schnittstelle vor, die in einer abgeleiteten Klasse definiert werden muss.

  ➔ siehe dazu auch Beispiel im Script Seite 3-178

12. Mehrfachvererbung

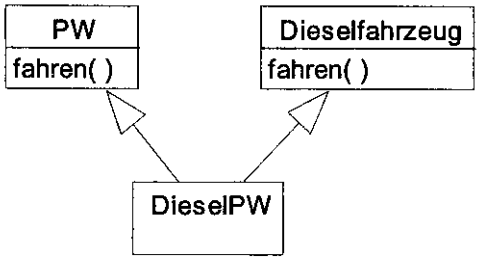
Die Bedeutung von Mehrfachvererbung ist umstritten:

- sie ist in manchen Programmiersprachen nicht vorhanden
- sie kann zu verschiedenen Problemen führen (z.B. Namenskonflikte)
- sie erlaubt das elegante Abbilden von Strukturen der realen Welt

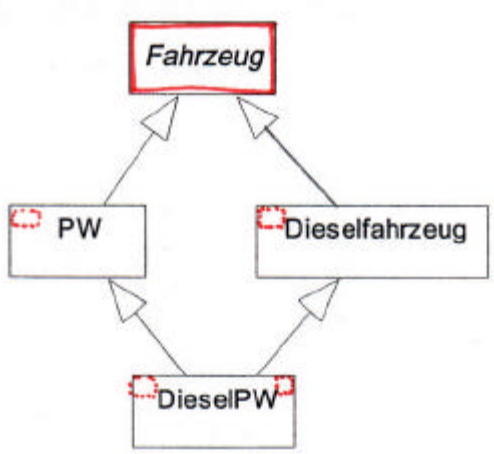
Empfehlung:

- Mehrfachvererbung nur mit Vorsicht einsetzen
- Mit den möglichen Problemen vertraut machen und sie von Anfang an vermeiden.

Beispiel:
ein DieselPW ist sowohl ein PW als auch ein Dieselfahrzeug, die Vererbung von beiden Oberklassen ist also korrekt.



Der Aufruf von `fahren()` gibt ein Compilerfehler, denn `fahren()`wird sowohl von PW als auch von Dieselfahrzeug geerbt. Daher muss die Klasse DieselPW eine eigene Funktion `fahren()` enthalten.



Hier enthält sowohl die Klasse PW als auch die Klasse Dieselfahrzeug ein Objekt Fahrzeug. Somit ist in der Klasse DieselPW das Objekt Fahrzeug doppelt enthalten. Man kann dies vermeiden, indem man dem Compiler explizit mitteilt, es solle nur ein Objekt Fahrzeug verwendet werden. Um ihm das mitzuteilen, wird die Basisklasse Fahrzeug als virtuelle Oberklasse von PW und Dieselfahrzeug deklariert.

class Fahrzeug { ... };

```
class PW : public virtual Fahrzeug { ... };
class Dieselfahrzeug : public virtual Fahrzeug { ... };

class DieselPW : public PW, public Dieselfahrzeug { ... };
```