MERKBLATT Vererbung

Klassenhierarchie: Vater und Sohn

- Im Gegensatz zur realen Welt weist die Oberklasse[1] weniger Fähigkeiten auf, als die abgeleitete Klasse[2] ➔ Generalisierung.
- Die Unterklasse (Subklasse) stellt also eine Verfeinerung) der Oberklasse (Super- / Basisklasse) dar ➔ Spezialisierung. Es entsteht eine "... *ist-ein*-Beziehung ..."!

Merkmale der Vererbung (bezogen auf Java)

- Es werden alle Methoden und Eigenschaften, die nicht `private` deklariert sind, durch die abgeleitete Klasse übernommen. ➔ Substitutionsprinzip
- Methoden und Eigenschaften die den Modifizierer `private` aufweisen, werden inhaltlich vom „Sohn“ gebraucht, sind aber für ihn nicht sichtbar! (besser `protected`)
- Der Code von abgeleiteten Klassen ist nur einmal physisch im Speicher abgelegt.
- Jedes Objekt erhält seinen eigenen Datenbereich im Speicher
- Eine mit `final` modifizierte Methode wird nicht mehr vererbt.
- Eine Änderung der Oberklasse erzwingt meistens auch eine Anpassung der abgeleiteten Klassen. (keine lose Kopplung mehr!)
- Die Vererbung ist eine mit Vorsicht zu verwendende Technik: Ihre Möglichkeiten und die Sinnhaftigkeit werden oft überschätzt!

Konstruktor / Destruktor

- Jedes Objekt ruft beim Erzeugen seinen (Standard-)Konstruktor auf.
- Implizit (oder besser *explizit* mit `super()`) ruft der Konstruktor *als erstes* den Standard-Konstruktor[3] seiner Oberklasse auf. (... bis zur obersten Klasse *object*!)
- Fehlt ein Konstruktor, wird trotzdem *implizit* der Standard-Konstruktor der Oberklasse aufgerufen.
- Konstruktoren können überladen werden, indem ihre Parameter variieren. Solche überladenen Konstruktoren sollten möglichst mit dem Standard-Konstruktor verkettet werden: `this()`.

```
public MyWindow extends Frame{

public MyWindow(String titel){

  super(titel);      // Aufruf des Konstruktors der Oberklasse

}

public MyWindow(String titel, int color){

..this(titel);       // Aufruf des eigenen Konstruktors

  setBackground(color);

}
```

- Konstruktoren kennen keinen Rückgabewert und dürfen auch keinen `void` Modifizierer aufweisen. Sie werden auch nicht vererbt!
- Java kennt keine Destruktoren, da nicht mehr benötigte Objekte durch die Garbage Collection entfernt werden.

- (Siehe auch *Singleton* und *Fabriken* zur Erzeugung von Objekten: JIK 6.5.11)

[1] auch als Oberklasse, Basisklasse, Superklasse oder Parent bezeichnet
[2] auch als Unterklasse, Subklasse oder Child bezeichnet
[3] Konstruktor ohne Parameter

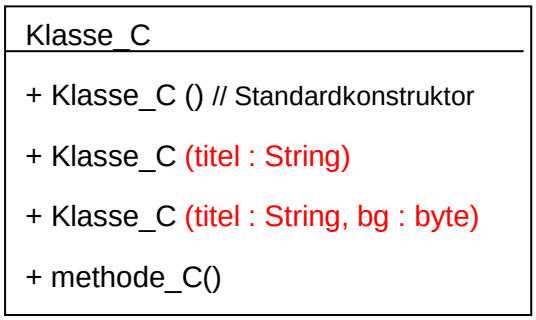
MERKBLATT Vererbung

Methoden-Polymorphie

Durch die Polymorphie ergeben sich bei OO-Sprachen Möglichkeiten, die sich mit der klassischen Programmierung nicht – oder nur auf Umwegen – realisieren lassen.

Überladen = statische Polymorphie

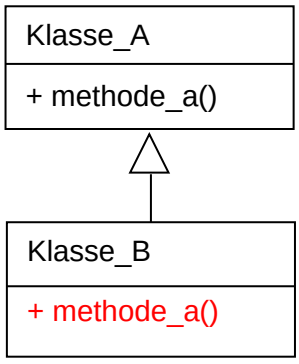
- Je nach Situation kann eine Methode unterschiedliche Parameter für ihre Aufgabe erhalten.
- Die Methoden müssen sich durch die Parameterliste unterscheiden! ➔wird oft bei Konstruktoren verwendet
- Die statische Bindung[4] (=Zuordnung) der entsprechenden Methode erfolgt hier bereits bei der Übersetzung.
- Die Instanziierung eines Objektes mit `objC1 = new Klasse_C("2. Konstruktor")` führt zu einem andern Ergebnis als dies bei `objC2 = new Klasse_C()` der Fall ist.



- Das Überladen kann innerhalb einer Klasse oder auch von der Unterklasse zur Oberklasse erfolgen!
- Beim Überladen die Methoden mit mehr Parametern mit denjenigen, die weniger Parameter haben, verketten. (Bei Konstruktoren mit `this(...)`)

Überschreiben = dynamische Polymorphie

- In einer abgeleiteten Klasse wird eine Methode benötigt, die sinngemäss wohl die gleiche Aufgabe zu lösen hat, wie jene der Oberklasse. Diese Aufgabe muss aber anders ausprogrammiert werden. Oft wird zusätzlich zuerst dieselbe Methode der Oberklasse mit `super._()` aufgerufen.
- Bei der dynamischen Bindung[5] – die zur Laufzeit erfolgt – wird immer die vorhandene Methode der tiefsten Hierarchiestufe verwendet!



Bei einem Aufruf `objB.methode_a()` wird real auch der Code der Methode in der Klasse `Klasse_B` ausgeführt und nicht jener der Klasse `Klasse_A`!

```
public class Klasse_A{
:
  public methode_a(){
    :
  }/*END methode_a*/
}/*END class Klasse_A*/

public class Klasse_B extends Klasse_A
:
  public methode_a(){
    :
  }/*END methode_a*/
}/*END class Klasse_B*/
```

- Das Überschreiben erfolgt immer von der Unterklasse zur Oberklasse!
  ➔ ein Punkt, warum Vererbung auch gefährlich sein kann!
- Die überschriebene Methode des „Vaters“ wird über `super.methode_a()` ausgeführt.

[4] =statische Bindung: vergleichbar dem Kompilieren bei klassischen Umgebungen
[5] =dynamische Bindung: vergleichbar dem Linken bei klassischen Umgebungen