

Mehr zur Vererbung

- Vererbung erlaubt es, von einer Klasse Unterklassen abzuleiten
 - Attribute und Methoden werden vererbt
 - Weitere Attribute und Methoden können ergänzt werden
 - Geerbte Methoden können überschrieben werden
- Vorgehensweise:
 - Spezialisierung vorhandener Klassen
 - “Herausziehen” von gemeinsamen Eigenschaften bereits vorhandener Klassen

Mögliches Problem

Betrachte die Klassenhierarchie auf dem Bild rechts:

- Die Klasse **GeometrieObjekt** kann alle Attribute und Methoden bereitstellen, die sich auf alle Arten von geometrischen Objekten anwenden lassen
- Ein Objekt vom Typ **GeometrieObjekt** ergibt jedoch wenig Sinn, denn ein konkretes Objekt ist entweder ein Kreis oder ein Rechteck

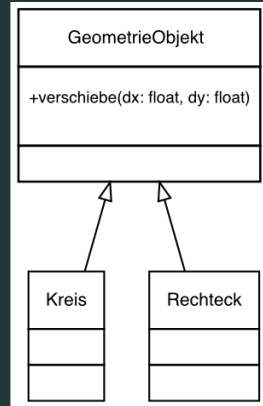


Abbildung 1: Beispielhafte Klassenhierarchie

Wie lässt sich verhindern, dass Instanzen der Klasse **GeometrieObjekt** erzeugt werden?

Abstrakte Klassen

- Eine abstrakte Klasse ist eine Klasse, von der sich keine Instanz erzeugen lässt
 - Um ihre Eigenschaften/Methoden benutzen zu können, **muss** man eine Unterklasse davon erzeugen

- Definition mit dem Schlüsselwort **abstract**:

```
abstract class GeometrieObjekt {  
    private float x, y;  
    public void verschiebe(float dx, float dy) {  
        x += dx;  
        y += dy;  
    }  
}
```

Darstellung in UML

- Klassenname wird *kursiv* geschrieben
- Zur Verdeutlichung: Zusätzliche Angabe von **{abstract}** vor oder nach den Namen

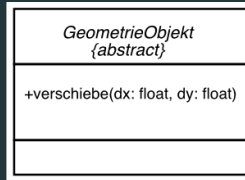


Abbildung 2: Abstrakte Klasse in UML

Aufgabe 1

- Erstelle in BlueJ die **abstrakte** Klasse `GeometrieObjekt` und die Unterklassen `Kreis` und `Rechteck`
 - Ergänze weitere sinnvolle Attribute und Methoden in **allen** Klassen
- Erstelle eine Klasse `Main` mit einer Klassenmethode `main`
 - Erzeuge Objekte vom Typ `Kreis` und `Rechteck`
 - Rufe die geerbte Methode `verschiebe(float dx, float dy)` auf diesen Objekten auf
 - Versuche auch, ein Objekt vom Typ `GeometrieObjekt` zu erzeugen

- Mit abstrakten Klassen kann man Methoden vorgeben, die für alle Unterklassen “nützlich” sind
- Manchmal will man hingegen nur “erzwingen”, dass eine Unterklasse eine bestimmte Methode selbst implementiert, ohne die Implementierung bereits vorzugeben

Interfaces

- Ein Interface enthält eine oder mehrere **abstrakte Methoden**, also Methoden ohne Implementierung
 - Es werden also nur Rückgabetyp, Methodenname sowie Art und Anzahl der Parameter festgelegt
- Eine Klasse kann ein oder mehrere Interfaces implementieren
 - Innerhalb der Klasse müssen die Methoden, die das Interface vorgibt, implementiert werden

Implementierung in Java

Definition eines Interfaces:

```
interface Flugfaehig {  
    public void fliege(int d);  
}
```

Benutzung:

```
class Flugzeug implements Flugfaehig {  
    @Override  
    public void fliege(int d) {  
        // ...  
    }  
}
```

interface Schlüsselwort, das die Definition eines Interfaces einleitet
{ bzw. } Block, der die Definition des Interfaces umschließt

implements Schlüsselwort, das angibt, dass eine Klasse ein Interface implementiert. Mehrere Interfaces werden durch Komma getrennt

@Override optionale Annotation, die angibt, dass die folgende Methode eine schon vorhandene überschreibt

- Prinzipieller Aufbau wie ein Klassendiagramm
- Das Schlüsselwort **<<interface>>** kennzeichnet ein Interface

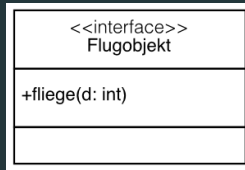


Abbildung 3: Interface in UML

Aufgabe 2

- Erstelle in BlueJ das Interface `Flugfaehig` mit der Methode `fliege(int d)`
- Erstelle eine Klasse `Flugzeug` und eine Klasse `Vogel`, die beide dieses Interface implementieren
 - Denke Dir einige sinnvolle zusätzliche Attribute (und ggf. Methoden) für beide Klassen aus
 - Prüfe, was passiert, wenn Du die Methode `fliege(int d)` in diesen Klassen **nicht** implementierst

- Interfaces können keine “vollwertigen” Attribute enthalten
 - Falls man doch welche definiert, so sind diese automatisch Konstanten
- Interfaces können auch Methoden mit vollständiger Implementierung enthalten
 - Entweder definiert als `static` oder als `default`

Wann nimmt man was? (1)

- Einfache Unterscheidung (für uns in der Regel ausreichend):
 - Abstrakte Klassen, wenn man die Implementierung von Methoden bereits vorgeben will
 - Interfaces, wenn man nur vorgeben will, welche Methoden es geben soll, ohne sie zu implementieren
- Aaaaaaber:
 - Abstrakte Klassen können auch Methoden ohne Implementierung enthalten (definiert als **abstract**)
 - Interfaces können auch Methoden mit Implementierung enthalten (definiert als **static** oder **default**)

Wann nimmt man was? (2)

- Zusätzliches Kriterium:
 - Abstrakte Klassen für gemeinsame Codenutzung bei eng verwandten Klassen (wie Kreis, Rechteck, Geometrieobjekt)
 - Interfaces zum Hinzufügen bestimmter Fähigkeiten zu nicht zwingend miteinander verwandten Klassen (wie Flugfähigkeit für Flugzeuge oder Vögel oder Papierflieger...)
- Auch die Verwendung einer Kombination aus beidem ist denkbar
 - Abstrakte Klasse implementiert Interface(s)
 - Eigene Klassen erben von der abstrakten Klasse