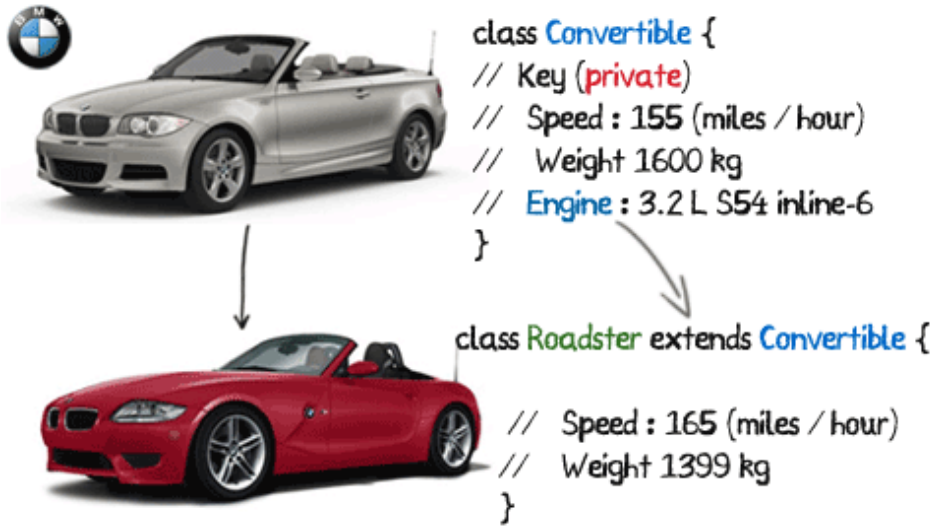
Grundlagen der Programmierung

VL13: Vererbung

Prof. Dr. Samuel Kounev
M.Sc. Norbert Schmitt

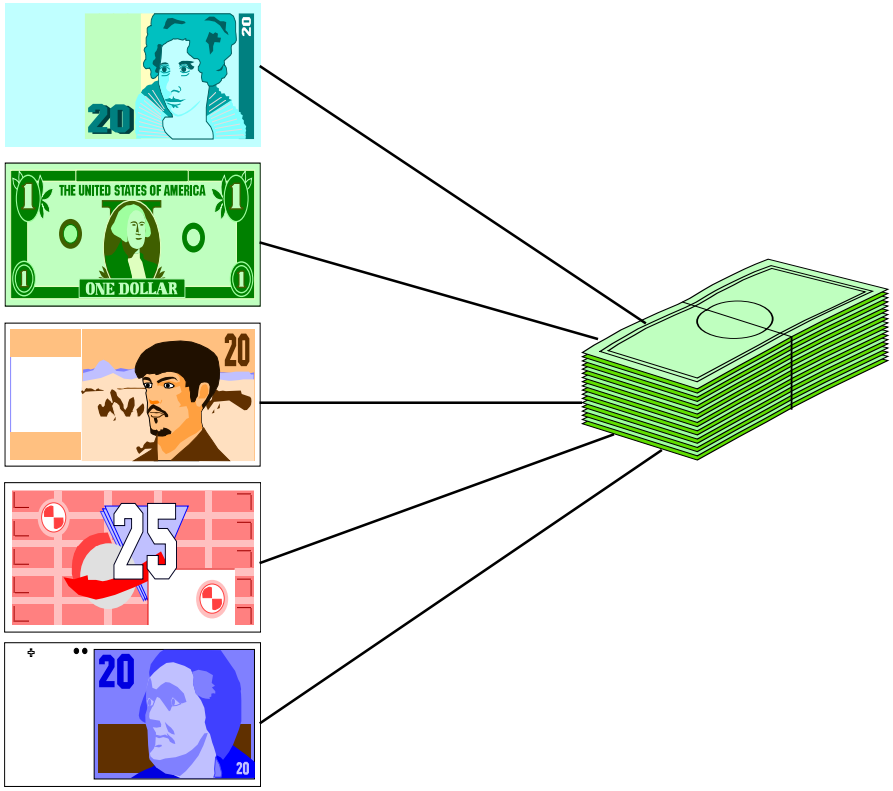


Inhalt

- **Aufgabenstellung**
- Ein erster Klassenentwurf
- Abstrakte Klassen und Vererbung
- Vererbung in der Praxis

Aufgabenstellung

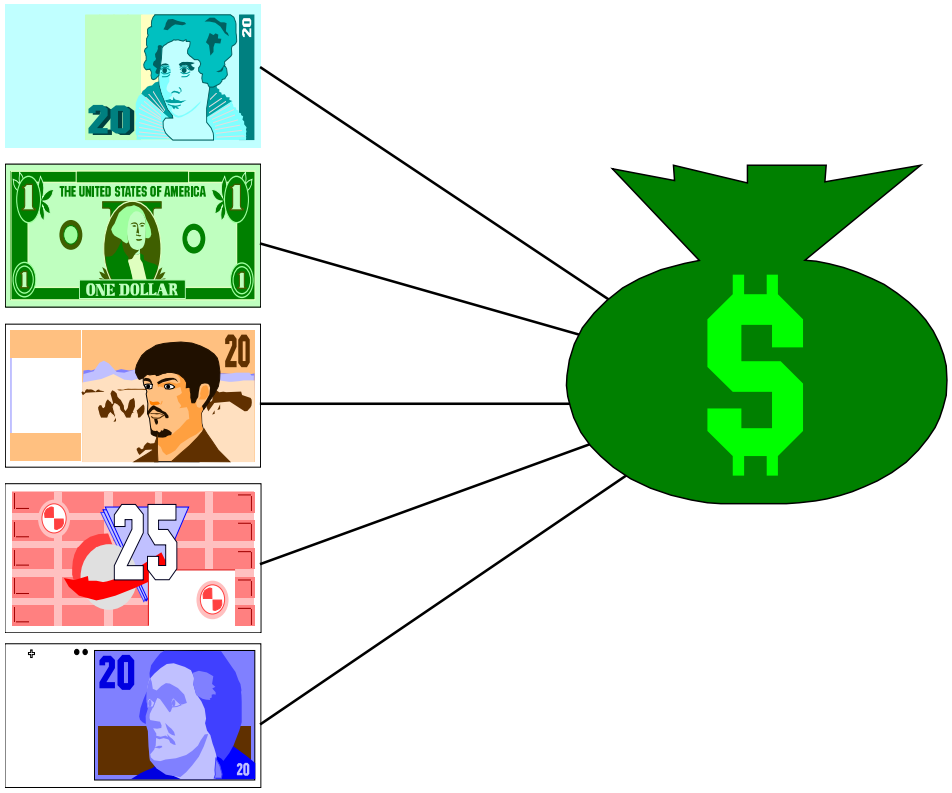
Eine internationale Hotelkette lässt für die Finanzbuchhaltung ein neues Softwaresystem entwickeln. Das Unternehmen ist in vielen Ländern vertreten und muss deshalb mit diversen Währungen umgehen.



Aufgabenstellung

Da die Firma ihren Hauptsitz in den vereinigten Staaten hat, wird der amerikanischen Dollar als Referenzwährung verwendet.

Sämtliche Vermögenswerte in den verschiedenen Ländern werden auf dieser Basis miteinander verrechnet.



Aufgabenstellung

Nehmen wir einmal an, verschiedene Posten dieser Aufstellung sind in einer Tabelle zusammengefasst

Währung	Betrag	Wert in US-$
$	17.300.000	?
€	1.400.000	?
£	3.000.000	?
...	...	...

Es gilt nun, diese Vermögen in die Referenzwährung umzurechnen!

Aufgabenstellung

Nehmen wir einmal an, verschiedene Posten dieser Aufstellung sind in einer Tabelle zusammengefasst

Währung	Betrag	Wert in US-$
$	17.300.000	17.300.000
€	1.400.000	1.391.788
£	3.000.000	1.852.072
...	...	...

Es gilt nun, diese Vermögen in die Referenzwährung umzurechnen!

Aufgabenstellung

Leider ist der Wechselkurs der verschiedenen Währungen nicht fest; die Devisenkurse ändern sich an den verschiedenen Börsen im Minutentakt. Wie können wir jedoch in Java dafür sorgen, dass die Umrechnung stets nach dem aktuellen Kurs automatisch erledigt wird?

Währung	Betrag	Wert in US-$
$	17.300.000	17.300.000
€	1.400.000	1.391.788
£	3.000.000	1.852.072
...	...	...

Inhalt

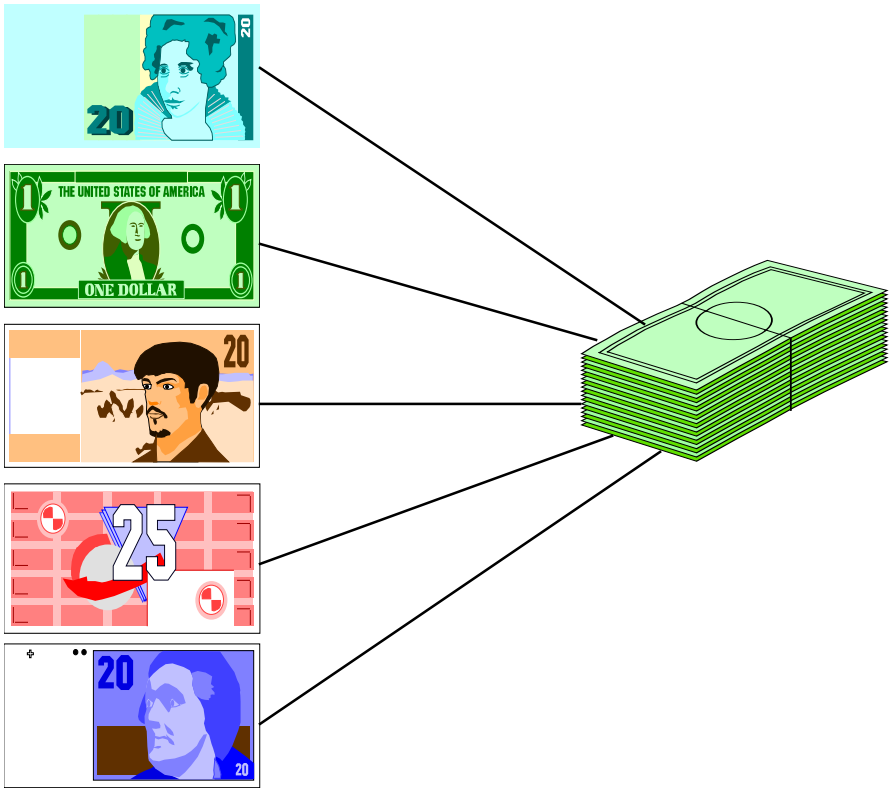
- Aufgabenstellung
- **Ein erster Klassenentwurf**
- Abstrakte Klassen und Vererbung
- Vererbung in der Praxis

Ein erster Klassenentwurf

Werfen wir erneut einen Blick auf verschiedene Währungen, die wir in dem vorherigen Diagramm kennengelernt haben

Wir stellen fest, dass jede dieser Währungen

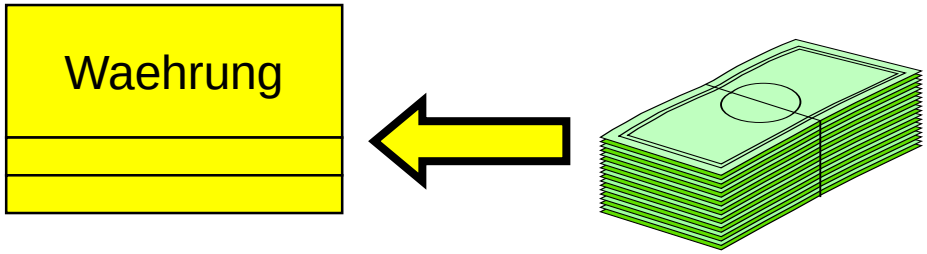
- gültiges Zahlungsmittel ist
- in US-$ umgerechnet werden kann, wobei sich
- diese Berechnung je nach Währungsart unterschiedlich gestalten kann



Ein erster Klassenentwurf

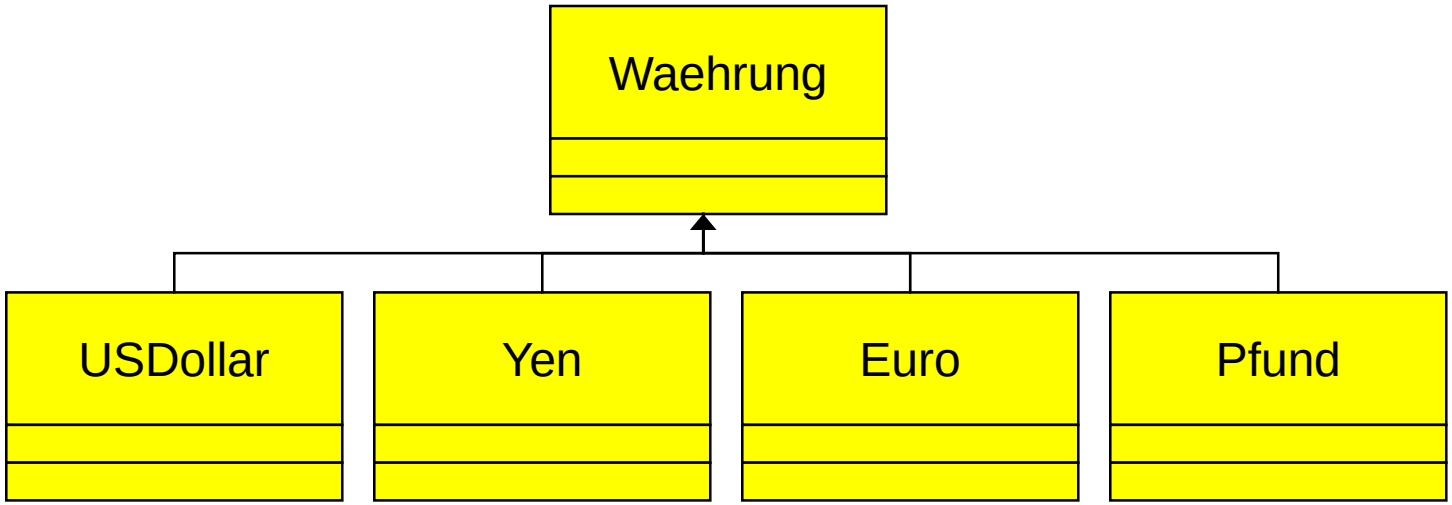
Wenn wir also die verschiedenen Währungen betrachten, so stellen wir bei ihnen gewisse Gemeinsamkeiten fest. Wir machen uns diesen Umstand zunutze und fassen diese in einer Klasse namens **`Waehrung`** zusammen.

Man bezeichnet diesen Vorgang des Abstrahierens und Zusammenfassens gemeinsamer Eigenschaften auch als Generalisierung.



Ein erster Klassenentwurf

Von dieser Klasse **`Waehrung`** ausgehend, werden wir die verschiedenen anderen konkreteren Währungen entwerfen



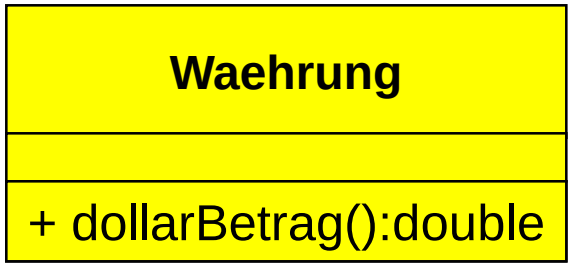
Inhalt

- Aufgabenstellung
- Ein erster Klassenentwurf
- **Abstrakte Klassen und Vererbung**
- Vererbung in der Praxis

Abstrakte Klassen und Vererbung

Instanzen der Klasse **`Waehrung`** sollen einen bestimmten Geldbetrag symbolisieren. Der Wert dieses Geldbetrages soll in der Referenzwährung (US-Dollar) ausgegeben werden. Wir sehen hierzu eine entsprechende Methode namens **`dollarBetrag`** vor.

Da **`Waehrung`** eine sehr allgemeine Struktur repräsentiert (und kein spezielles Zahlungsmittel wie etwa Dollar, Rubel oder Yen), haben wir keine Ahnung, was für einen Wert die Methode **`dollarBetrag`** konkret zurückgeben sollte.

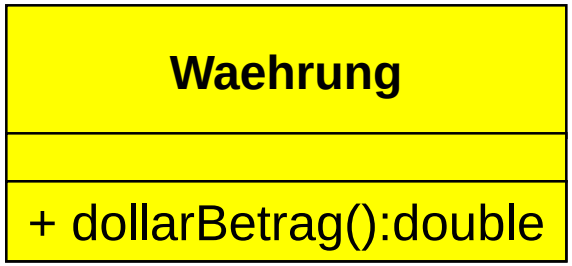


Abstrakte Klassen und Vererbung

Wir beginnen nun mit der Umsetzung unserer Ideen und realisieren die Klasse **`Waehrung`**.

Instanzen der Klasse **`Waehrung`** sollen einen bestimmten Geldbetrag symbolisieren. Der Wert dieses Geldbetrages soll in der Referenzwährung (US-Dollar) ausgegeben werden. Wir sehen hierzu eine entsprechende Methode namens **`dollarBetrag`** vor.

Weitere Details sind für die Klasse (im Moment) nicht vorgesehen, da alle zusätzlich denkbaren Eigenschaften (wie etwa das Verfahren zur Umrechnung bzw. der Wechselkurs) in den verschiedenen speziellen Währungen unterschiedlich modelliert werden müssen.

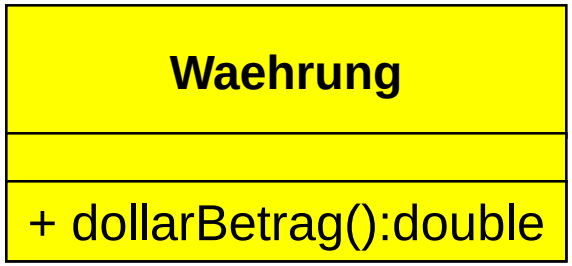


Abstrakte Klassen und Vererbung

Wenn wir unsere Klasse jedoch derart allgemein halten wollen, stoßen wir bald auf ein Problem:

Da **`Waehrung`** eine sehr allgemeine Struktur repräsentiert (und kein spezielles Zahlungsmittel wie etwa Dollar, Rubel oder Yen), wissen wir nicht, was für einen Wert die Methode **`dollarBetrag`** konkret zurückgeben sollte.

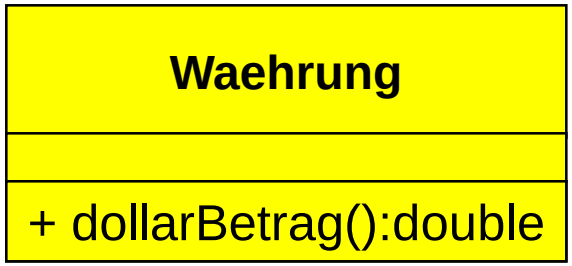
Auch macht es im Moment nur wenig Sinn, die Klasse **`Waehrung`** zu instantiieren. Diese steht schließlich für das allgemeine Konzept „Geld“ und nicht speziell für eine konkrete Realisierung...



Abstrakte Klassen und Vererbung

Wir lösen dieses Problem, indem wir die Klasse als abstrakt deklarieren.

Im Gegensatz zu unserer bisherigen Klassendefinition können wir hier (ähnlich wie in UML) eine Ansammlung von Instanzmethoden definieren, ohne die Methoden explizit ausformulieren zu müssen (man spricht in diesem Zusammenhang auch von der Definition einer **Schnittstelle** – bitte trotz gleicher Wortwahl nicht verwechseln mit dem später einzuführenden ähnlichen Konstrukt **`interface`**).

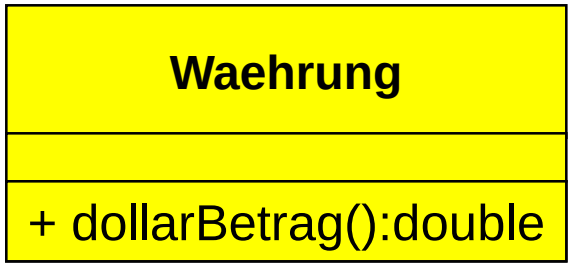


Abstrakte Klassen und Vererbung

Wir lösen dieses Problem, indem wir die Klasse als abstrakt deklarieren.

Im Gegensatz zu unserer bisherigen Klassendefinition können wir hier (ähnlich wie in UML) eine Ansammlung von Instanzmethoden definieren, ohne die Methoden explizit ausformulieren zu müssen (man spricht in diesem Zusammenhang auch von der Definition einer **Schnittstelle** – bitte trotz gleicher Wortwahl nicht verwechseln mit dem später einzuführenden ähnlichen Konstrukt **`interface`**).

Natürlich würde es wenig Sinn machen, eine derart „unvollständige“ Klasse zu instantiieren. Aus diesem Grund ist es auch nicht möglich, aus abstrakten Klassen direkt Objekte zu bilden.

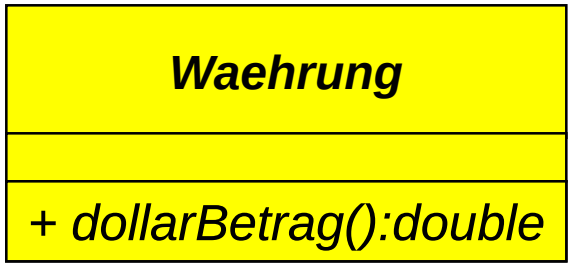


Abstrakte Klassen und Vererbung

Wir übertragen diese Überlegungen nun auf unser Klassenmodell. Wenn wir in UML eine Klasse als abstrakt markieren wollen, schreiben wir ihren Namen in *Kursivschrift*.

Auch unsere Methode **`dollarBetrag`** soll abstrakt gehalten (also nicht konkret ausformuliert) werden. Wir setzen sie deshalb ebenfalls *kursiv*.

Es stellt sich allerdings noch die Frage, wie sich dieses Modell in der Sprache Java umsetzen lässt...



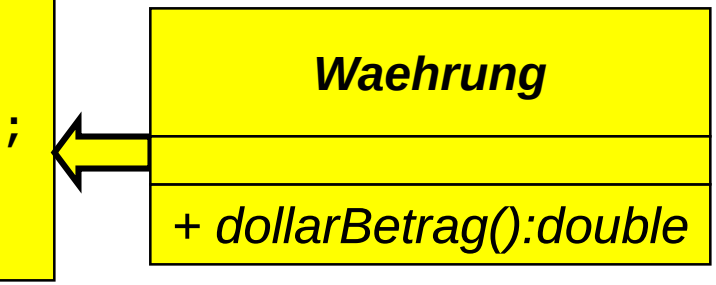
Abstrakte Klassen und Vererbung

Java

Werfen wir also einen Blick auf die Realisierung in Java.

Da die Methode **`dollarBetrag`** nun abstrakt ist, muss diese nicht weiter ausformuliert werden. Wir beenden sie statt mit dem Rumpf ({...}) mit einem einfachen Semikolon.

```
public abstract class Waehrung {

  public abstract double dollarBetrag();

}
```



Abstrakte Klassen und Vererbung

Java

Nicht jede Methode, die innerhalb einer abstrakten Klasse definiert wird, muss zwangsläufig auch abstrakt formuliert werden.

Andererseits muss jedoch jede Klasse, die eine abstrakte Methode enthält, zwangsläufig auch abstrakt sein!

```
public abstract class Waehrung {

  public abstract double dollarBetrag();

}
```

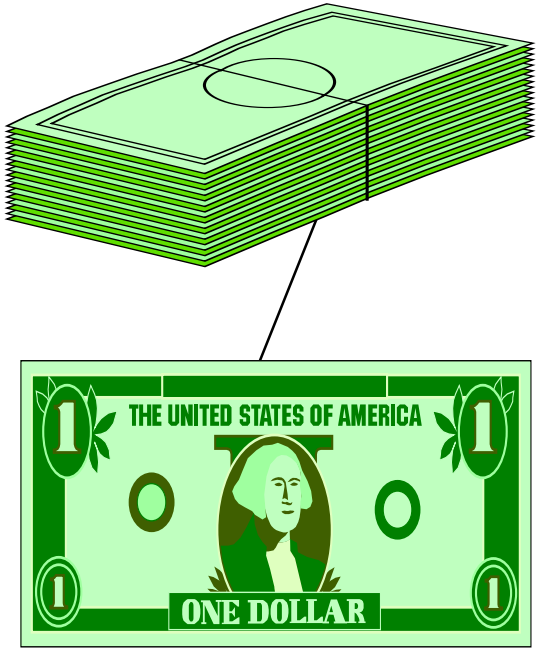
Waehrung
+ dollarBetrag():double

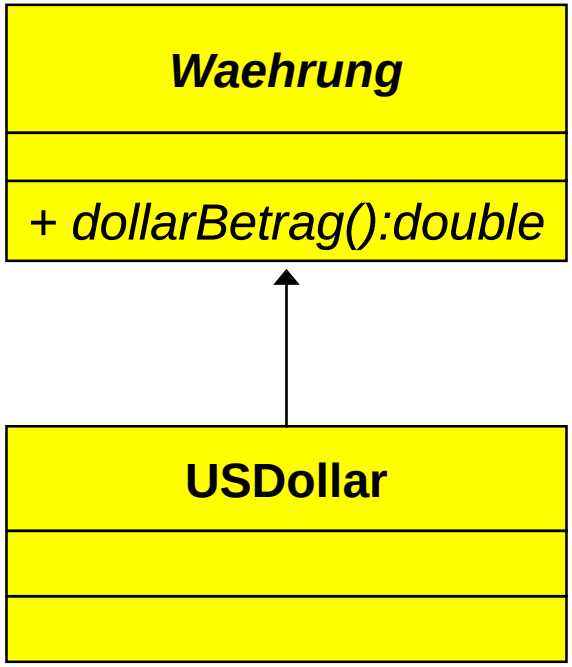
Abstrakte Klassen und Vererbung

Java

Im nächsten Schritt wollen wir unsere abstrakte Klasse nutzen, um eine konkrete Währung zu realisieren.

Hierzu beginnen wir mit dem einfachsten Fall - dem amerikanischen Dollar.



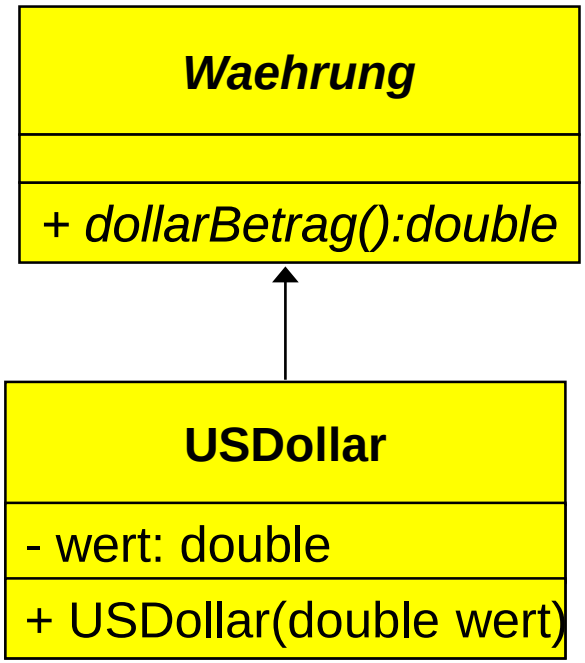


Abstrakte Klassen und Vererbung

Java

Instanzen der Klasse **`USDollar`** sollen einen Geldbetrag in der amerikanischen Währung repräsentieren. Um diesen Wert im Objekt hinterlegen zu können, definieren wir eine (**`private`**) Instanzvariable namens **`wert`**.

Die Belegung der Instanzvariable mit ihrem Wert wollen wir (wie schon in der Klasse **`Student`** geschehen) im Konstruktor vornehmen. Wir nehmen einen entsprechenden Eintrag in unserem UML-Modell vor.



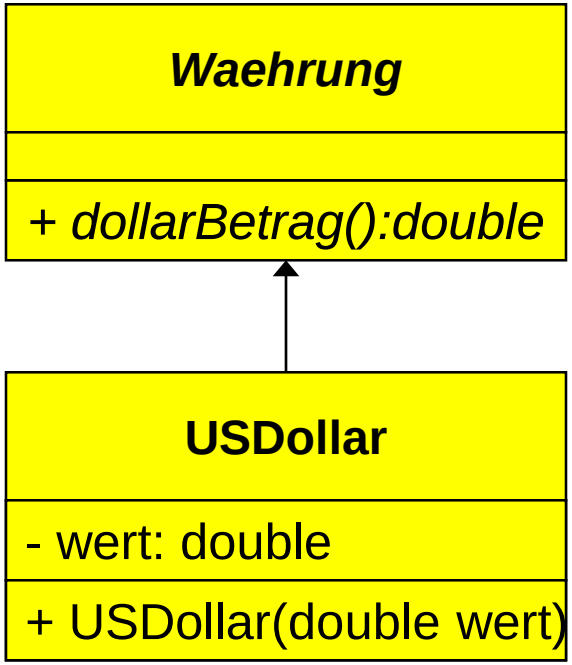
Abstrakte Klassen und Vererbung

Java

Wir werden nun versuchen, das Klassenmodell in Java umzusetzen.

Der unten angegebene Quelltext setzt hierbei jenen Teil des Modells um, den wir mit unseren bisherigen Kenntnissen in Java formulieren können. Auf die Beziehung zwischen den Klassen **`Waehrung`** und **`USDollar`** sind wir hierbei noch nicht eingegangen...

```
public class USDollar {

  private double wert;

  public USDollar(double wert) {
    this.wert = wert;
  }



}
```



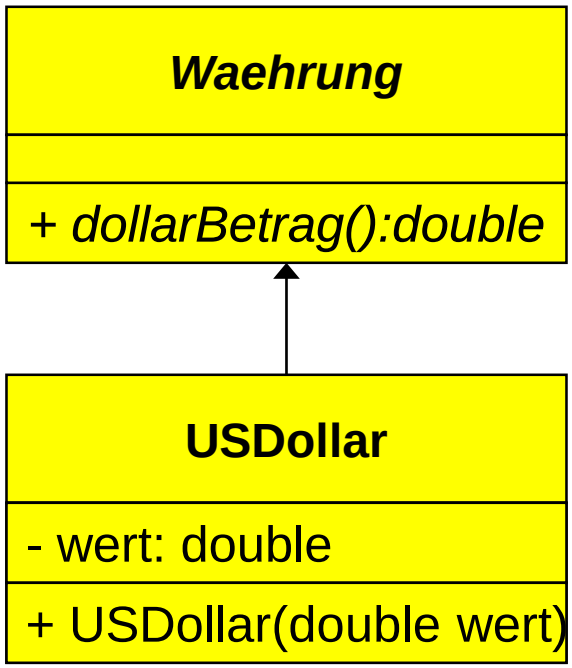
Abstrakte Klassen und Vererbung

Java

Die Klassen **`Waehrung`** und **`USDollar`** stehen hierbei in einer „ist-ein“-Beziehung. Man sagt, **`Waehrung`** ist **Superklasse** von **`USDollar`**; **`USDollar`** ist **Subklasse** von **`Waehrung`**.

Eine andere Sprechweise ist auch, die Klasse **`USDollar`** erweitert (engl. **extends**) die Klasse **`Waehrung`**.

```
public class USDollar extends Waehrung {

  private double wert;

  public USDollar(double wert) {
    this.wert = wert;
  }



}
```

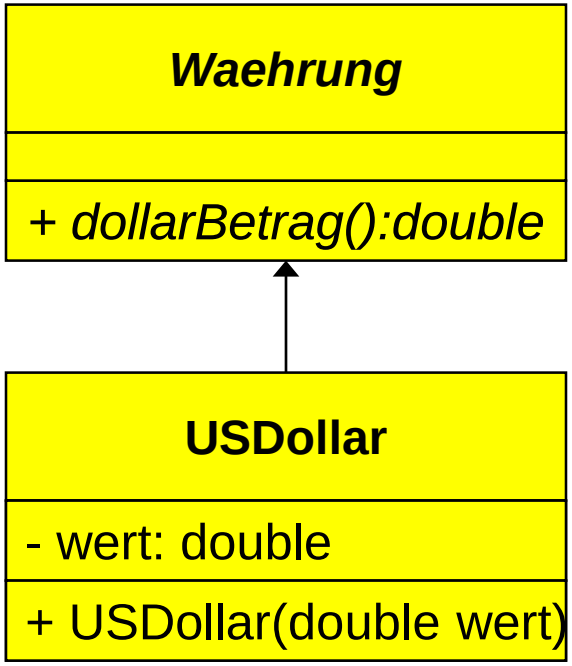


Abstrakte Klassen und Vererbung

Java

Die Formulierung („**USDollar** erweitert/extends **Waehrung**“) motiviert die Schreibweise, in der die Beziehung zwischen den beiden Klassen im Java-Programm formuliert wird. Das Schlüsselwort „**extends**“ dient als Mittel, um eine Verwandtschaft unter Klassen auszudrücken.

```
public class USDollar extends Waehrung {

  private double wert;

  public USDollar(double wert) {
    this.wert = wert;
  }



}
```

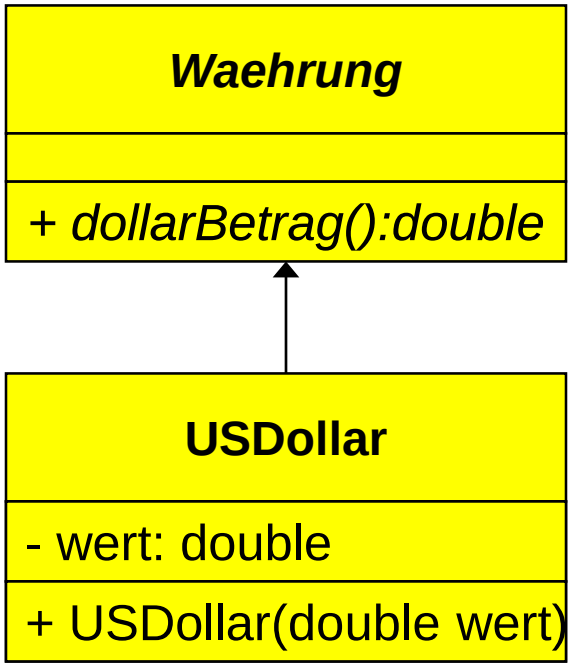


Abstrakte Klassen und Vererbung

Java

Die Klasse **`USDollar`** erbt von ihrer Superklasse (**`Waehrung`**) sämtliche Eigenschaften (also alle Variablen und alle Methoden). Eine Instanz der Klasse **`USDollar`** besitzt somit also automatisch auch eine Methode **`dollarBetrag`**, da diese für die Superklasse **`Waehrung`** vereinbart wurde.

```
public class USDollar extends Waehrung {

  private double wert;

  public USDollar(double wert) {
    this.wert = wert;
  }



}
```

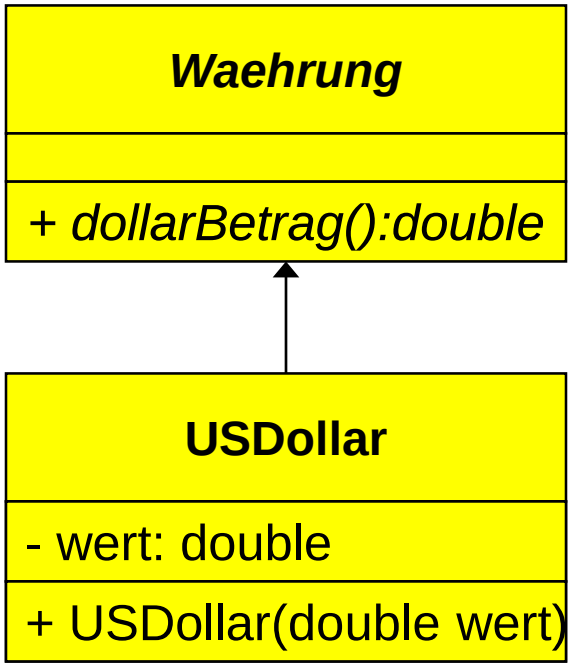


Abstrakte Klassen und Vererbung

Java

Nun befinden wir uns jedoch in der Situation, daß **`Waehrung`** eine abstrakte Klasse ist. Die Methode **`dollarBetrag`** ist ebenfalls abstrakt formuliert, so dass unsere nicht-abstrakte Klasse **`USDollar`** eine abstrakte Methode besitzt. Der Compiler honoriert dieses Verhalten mit einem Übersetzungsfehler!

```
public class USDollar extends Waehrung {

  private double wert;

  public USDollar(double wert) {
    this.wert = wert;
  }



}
```



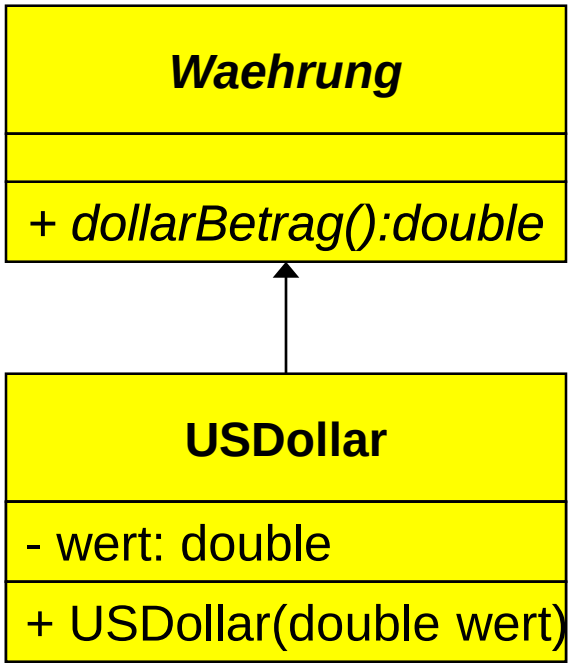
Abstrakte Klassen und Vererbung

Java

Anstatt einer reinen Schnittstellenvorgabe spendieren wir der Klasse also eine gültige Methode inklusive eines funktionsfähigen Rumpfes.

Man bezeichnet diesen Vorgang auch als **Überschreiben** (override)

```
public class USDollar extends Waehrung {

  private double wert;

  public USDollar(double wert) {
    this.wert = wert;
  }

  public double dollarBetrag() {
    return wert;
  }
}
```



Abstrakte Klassen und Vererbung

Java

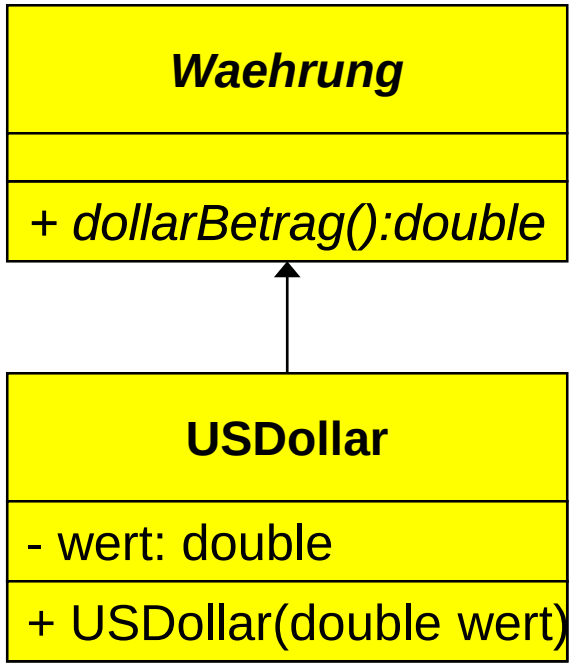
Wir können unsere Klasse **`USDollar`** nun wie gewohnt in Programmen einsetzen. Um beispielsweise ein Objekt „im Werte“ von $0.25 zu erzeugen, genügt die Programmzeile

```
USDollar quarter = new USDollar(0.25);
```

Wegen der Beziehung zwischen den Klassen können wir eine Instanz der Klasse **`USDollar`** aber auch als ein beliebiges Waehrungsobjekt behandeln. Die Zeile

```
Waehrung quarter = new USDollar(0.25);
```

wäre somit ebenfalls korrekt!



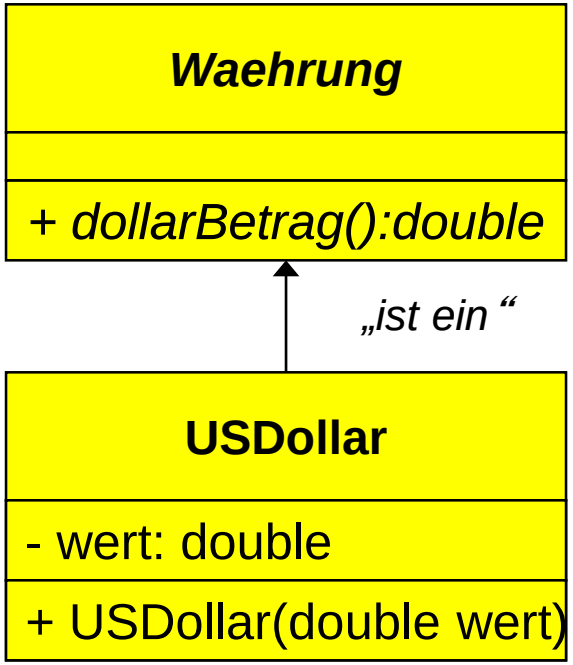
Inhalt

- Aufgabenstellung
- Ein erster Klassenentwurf
- Abstrakte Klassen und Vererbung
- **Vererbung in der Praxis**

Vererbung in der Praxis

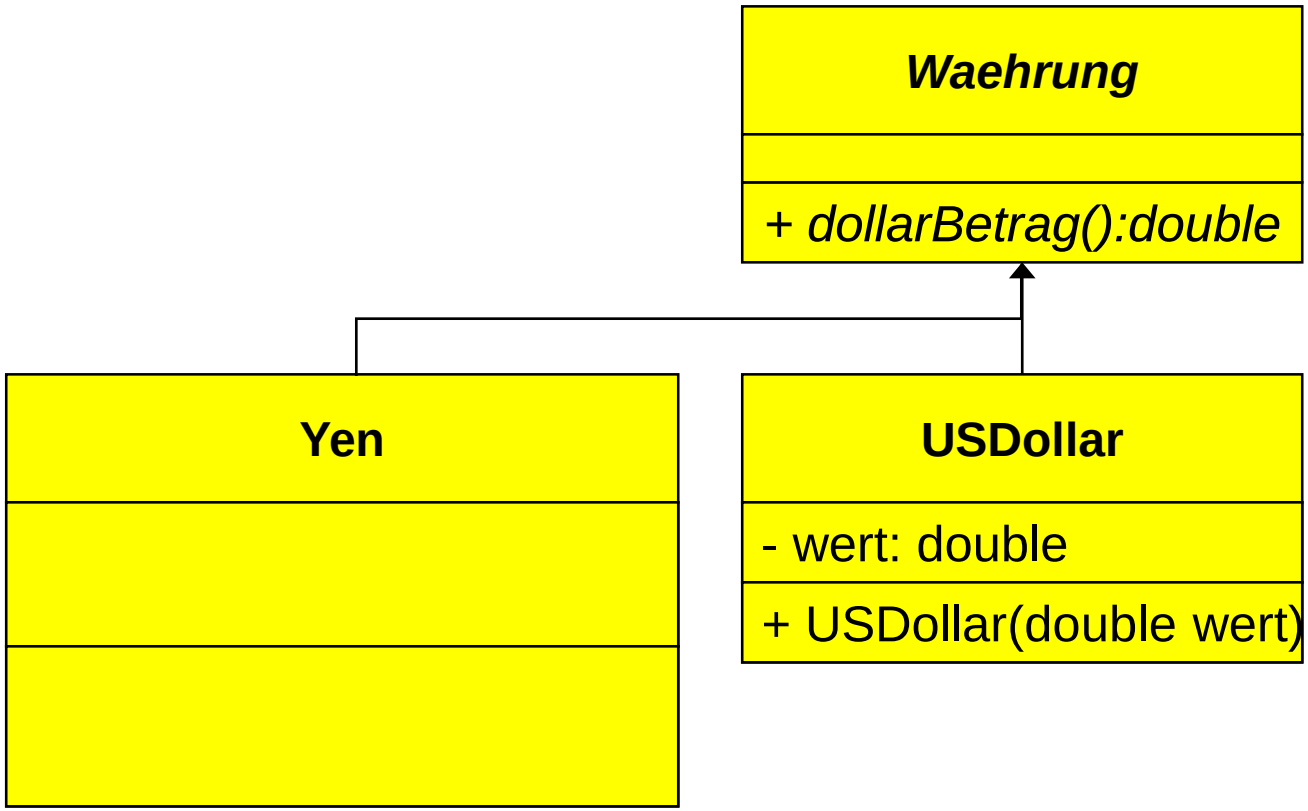
Wir haben mit der Klasse **`USDollar`** eine Subklasse unserer Klasse **`Waehrung`** geschaffen. Wegen der verwandtschaftlichen Beziehung zwischen den beiden Klassen („**`USDollar extends Waehrung`**“) haben wir festgestellt, dass ein Objekt der Subklasse **`USDollar`** auch als ein Objekt der Superklasse **`Waehrung`** aufgefasst werden kann.

*Wir wissen allerdings noch nicht, welchen **praktischen** Nutzen wir von dieser Beziehung haben!*



Vererbung in der Praxis

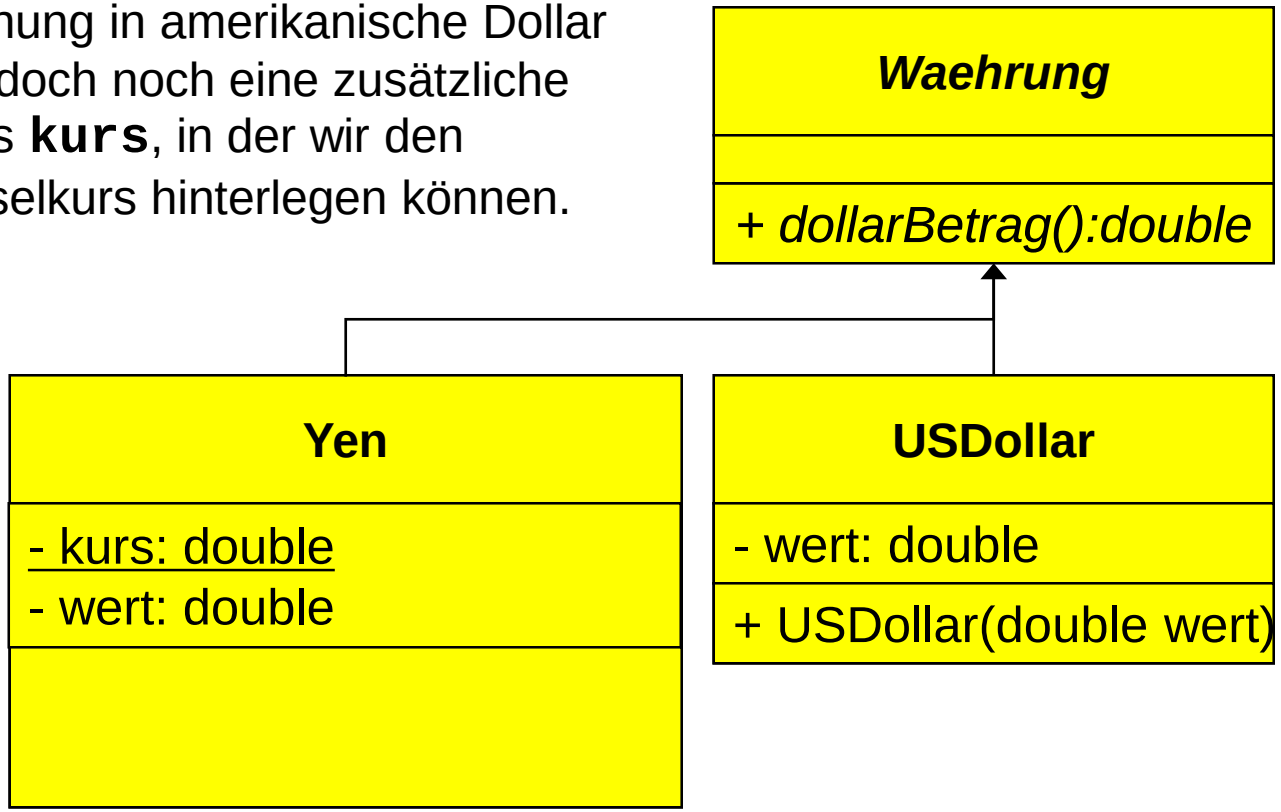
Wir werden diesen Nutzen deshalb an einem Beispiel verdeutlichen. Zuvor müssen wir allerdings noch eine zweite Subklasse von **`Waehrung`** bilden: wir modellieren das Zahlungsmittel **`Yen`**.



Vererbung in der Praxis

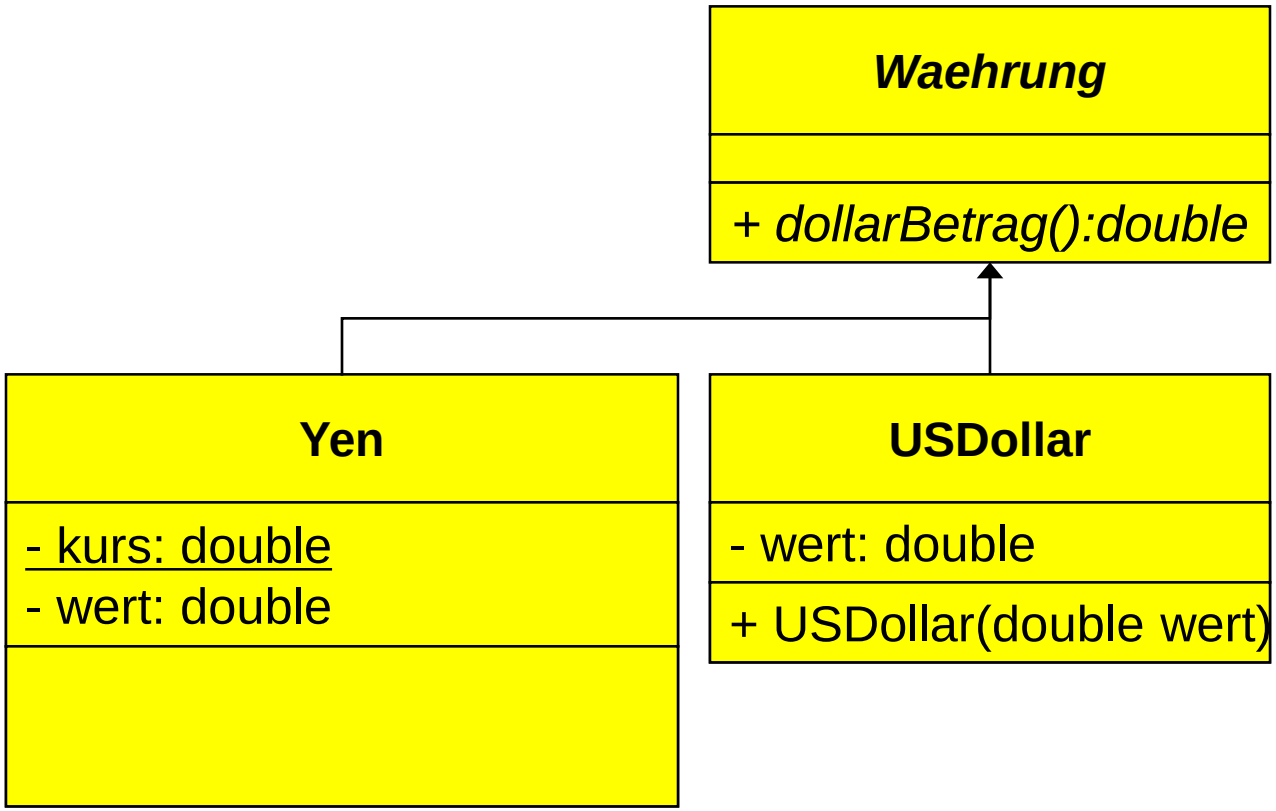
Ähnlich wie bei der Klasse **`USDollar`** wollen wir den Wert eines Währungsobjektes der Kategorie **`Yen`** in einer privaten Instanzvariablen namens **`wert`** speichern.

Für die Umrechnung in amerikanische Dollar benötigen wir jedoch noch eine zusätzliche Variable namens **`kurs`**, in der wir den aktuellen Wechselkurs hinterlegen können.



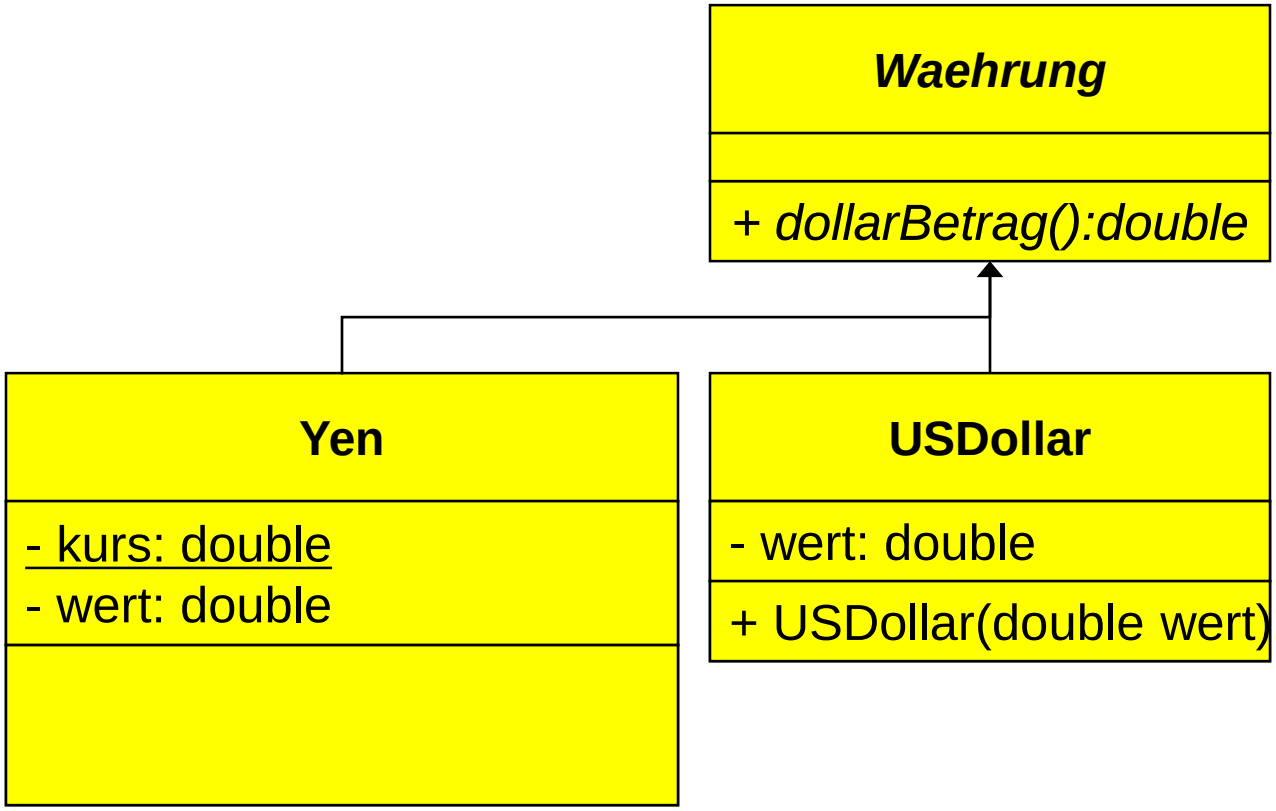
Vererbung in der Praxis

Der Wechselkurs für eine Währung zu einen bestimmten Zeitpunkt hängt nicht von einem speziellen Geldobjekt ab - er ist für alle Instanzen der Klasse **`Yen`** identisch. Es ist deshalb ratsam, die Variable **`kurs`** **statisch** anzulegen (also als Klassenvariable).



Vererbung in der Praxis

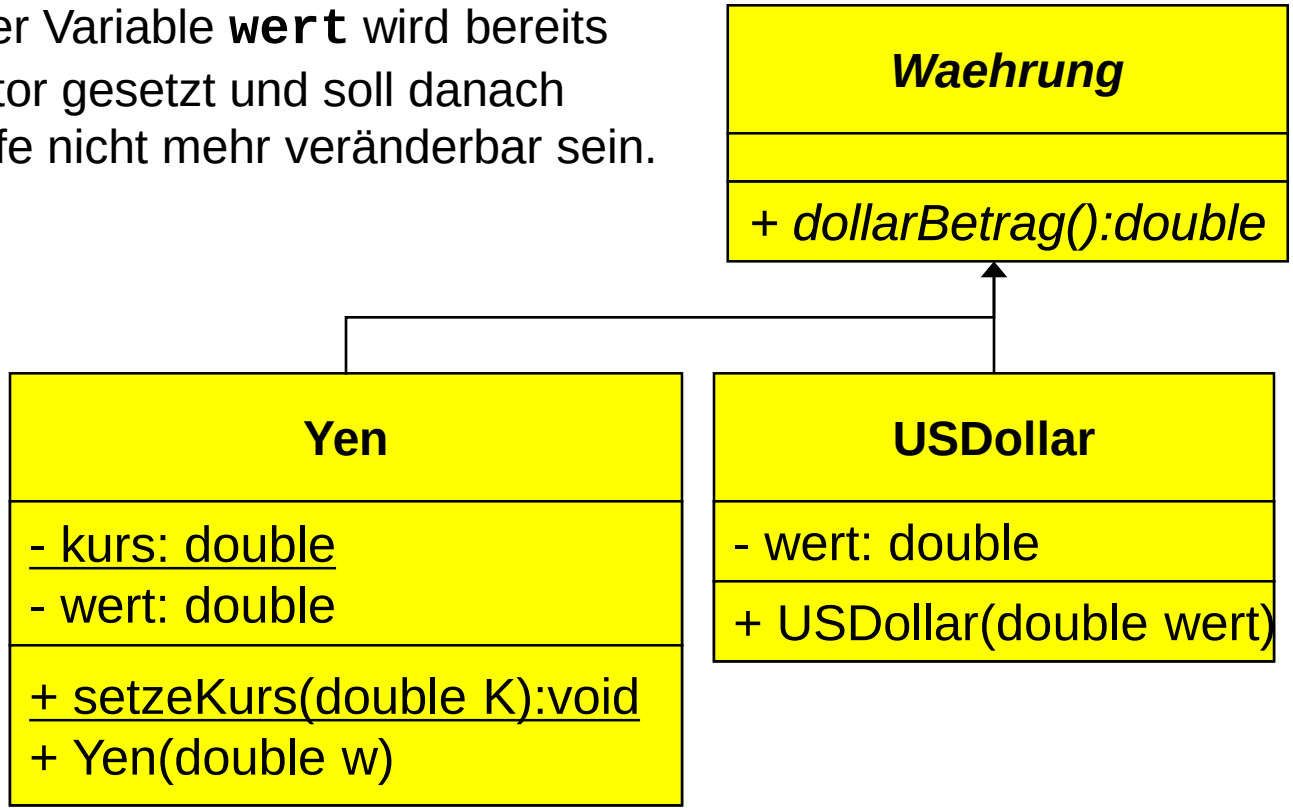
Dieses Vorgehen hat den Vorteil, dass der Kurs einer speziellen Währung global gesetzt werden kann - eine Aktualisierung kann somit stattfinden, ohne dass hierzu sämtliche Objekte überprüft werden müssen...



Vererbung in der Praxis

Natürlich benötigen wir auch eine Möglichkeit, die (**`private`**) Klassenvariable auf einen bestimmten Wert zu setzen. Hierzu definieren wir eine Klassenmethode **`setzeKurs`**, die dieses Problem in die Hand nimmt.

Der Inhalt der Variable **`wert`** wird bereits im Konstruktor gesetzt und soll danach durch Zugriffe nicht mehr veränderbar sein.

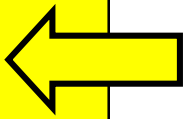


Vererbung in der Praxis

Java

```
public class Yen extends Waehrung {

  private static double kurs;
  private double wert;

  /** Konstruktor */
  public Yen(double w) {wert = w;}

  /** Kurs setzen */
  public static void setzeKurs(double K) {
    kurs = K;
  }

  /** Wert in US-Dollar */
  public double dollarBetrag() {
    return wert * kurs;
  }
}
```

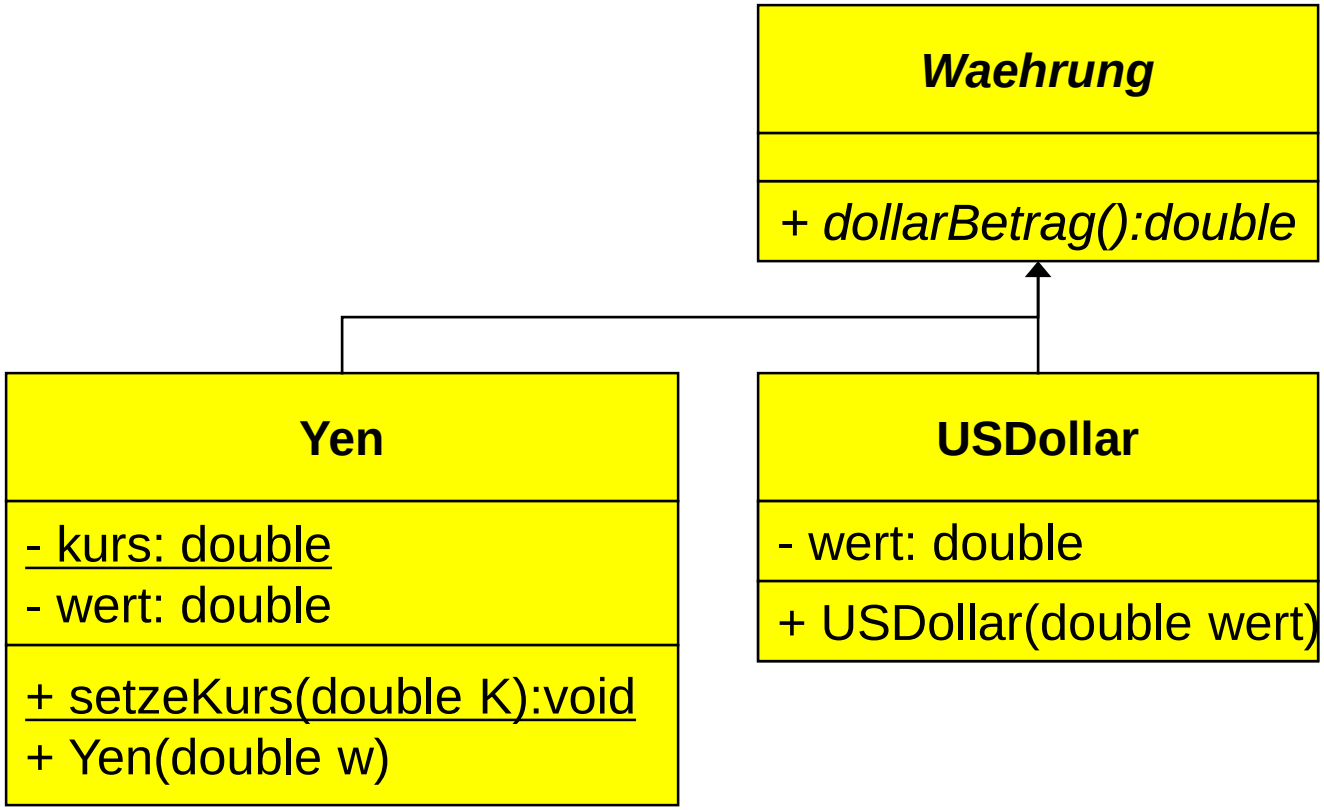
Werfen wir einen Blick auf die Umsetzung des Modells in Java.



Yen
- kurs: double - wert: double
+ setzeKurs(double K):void + Yen(double w)

Vererbung in der Praxis

Zurück zu unserem Softwareprojekt. Unsere Hotelkette möchte mit ihrer Finanzbuchhaltung unter anderem ihre **Steuererklärung** machen können. Die Firma wird in den USA veranlagt.



Vererbung in der Praxis

Nach einigen schlaflosen Nächten kommen die Steuerberater zu folgendem Schluss: Die Firma muss eine Steuer von exakt 8 Prozent auf ihr gesamtes Barvermögen zahlen.

Diese Summe soll von unserem Java-Programm automatisch berechnet und bei Fälligkeit an die Finanzbehörde überwiesen werden.

Unglücklicherweise besitzt die Firma Vermögen in den verschiedensten Devisen, so dass die Beträge zu dem entsprechenden Datum alle umgerechnet werden müssen.

Vererbung in der Praxis

Java

Hätten wir es nur mit einer Klasse von Objekten (also einer Währungsart) zu tun, wäre das Problem schnell gelöst:

```
public static double berechneSteuer(Waehrung[] geld) {
  double summe = 0;                          // der Gesamtbetrag
  for (int i=0; i < geld.length; i++)        // wird in einer Schleife
    summe += geld[i].dollarBetrag();         // summiert und danach
  double steuer = summe * 0.08;              // mit 8 Prozent
  return steuer;                             // multipliziert
}
```

Alle Geldmittel, die in dem Feld **`geld`** gespeichert sind, werden summiert - und danach 8% dieser Summe berechnet.

Vererbung in der Praxis

Java

Aufgrund ihrer verwandtschaftlichen Beziehungen können wir ohne Probleme ein „gemischtes“ Feld erstellen. Jedes Währungsobjekt lässt sich in diesem Feld unterbringen:

```
Waehrung[] geld = new Waehrung[3];
geld[0] = new USDollar(2500);
geld[1] = new Yen(200000);
geld[2] = new USDollar(20);

double steuer = berechneSteuer(geld);
```

Dank der Beziehungen zwischen Sub- und Superklasse stellt also das Array **`geld`** ein gültiges Feld von **`Waehrung`**-Objekten dar. Wir können unsere Methode **`berechneSteuer`** auf dieses Feld anwenden.

Vererbung in der Praxis

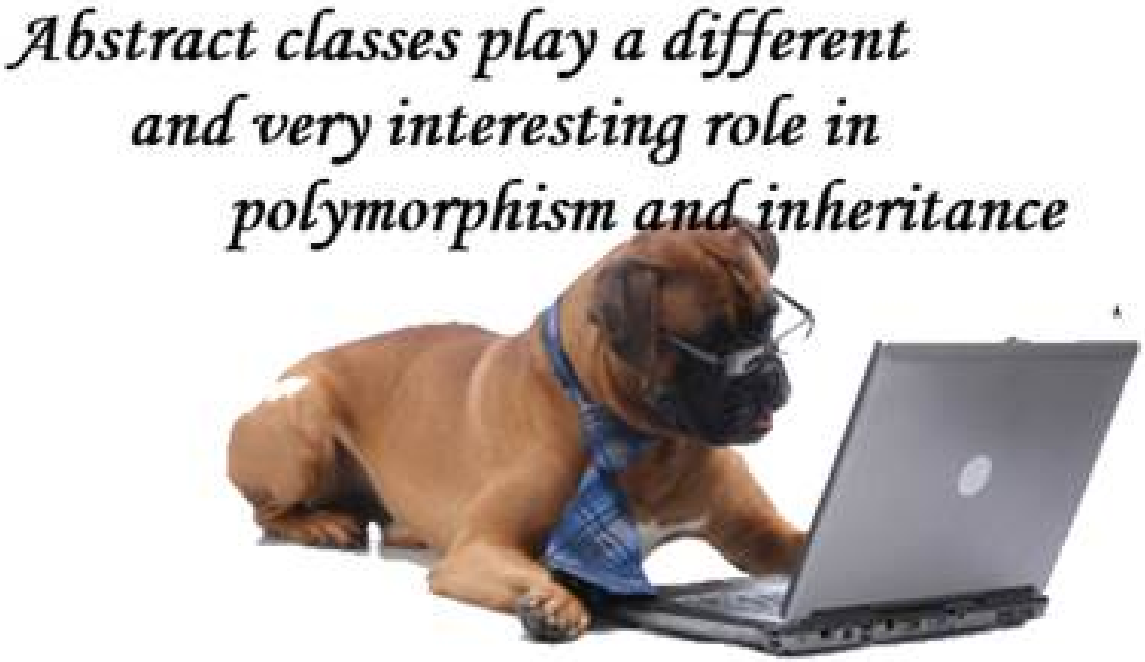
Java

`USDollar` und **`Yen`** sind zwei verschiedene Währungen mit unterschiedlichen Eigenschaften.

Trotz dieser Unterschiede haben die beiden Klassen jedoch Gemeinsamkeiten in ihrer Schnittstelle - nämlich die abstrakte Superklasse **`Waehrung`**. Obwohl beide Subklassen diese abstrakte **`Waehrung`** völlig unterschiedlich erweitert haben (vgl. die beiden Implementierungen von **`dollarBetrag`**), reichen die gemeinsamen Eigenschaften jedoch bereits aus, um Operationen von der abstrakten Klasse auf beide Kindklassen zu übertragen.

Die Methode **`berechneSteuer`** muss nur ein einziges Mal verfasst werden - egal, wieviele verschiedene Währungen wir implementieren!

Fragen?


Abstract classes play a different
and very interesting role in
polymorphism and inheritance

Danksagung

- Vorlesungsmaterialien von Prof. Dr. Detlef Seese wurden als Basis verwendet
- Unterstützung bei der technischen und inhaltlichen Gestaltung des Vorlesungsmaterials leisteten:

  Jóakim v. Kistowski

  Dietmar Ratz, Joachim Melcher, Roland Küstermann, Jana Weiner, Hagen Buchwald, Matthes Elstermann, Oliver Schöll, Niklas Kühl, Tobias Diederich