Objektorientierte Programmiertechnik

Kapitel 11 – Schnittstellen (Interfaces)

Übersicht

Inhalte

✓ Konzept

Implementierung

Interface vs. Abstrakte Klasse

Cloneable

Konzept I

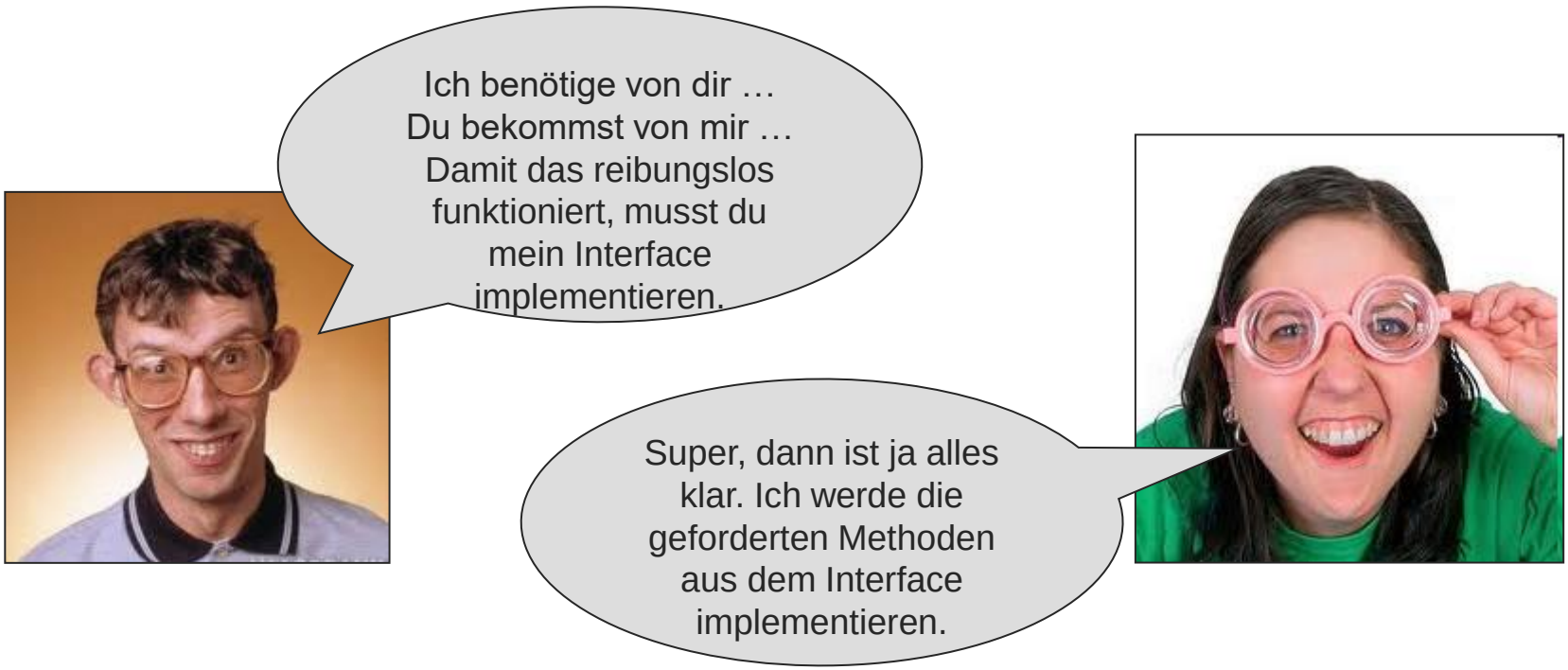
- Eine Schnittstelle (ein Interface) ist ein Sprachmittel für den Entwurf. Eine Klasse beinhaltet dagegen – sofern sie nicht abstrakt ist – den Entwurf und die Implementierung, d.h. die Methodenrümpfe:

  - Sie ist eine besondere Form einer Klasse, die ausschließlich abstrakte Methoden und Konstanten enthält
  - grob vergleichbar mit den Headern aus C
  - Anstelle von `class` wird das Schlüsselwort `interface` verwendet
  - ein Interface enthält keinen Konstruktor (kann auch nicht instantiiert werden)

```
interface PunktSchnittstellen
{
   public int getX();         // Eine Methode, um den x-Wert abzuholen
   public void setX (int i);// Eine Methode, um den x-Wert zu setzen
}
```

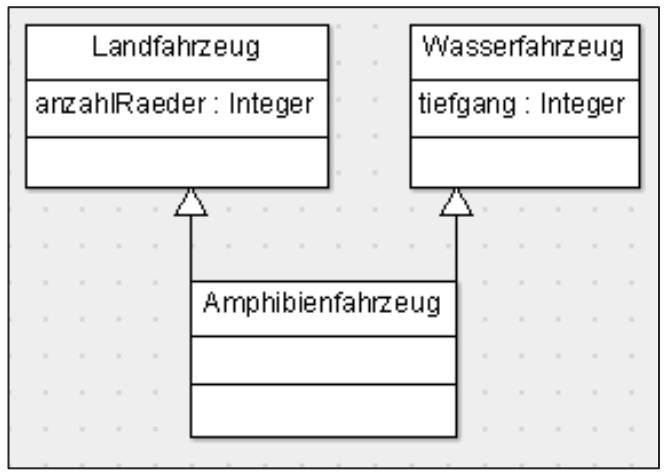
Konzept II

- Wozu braucht man Interfaces?
  - Ein Interface ist ein Sprachmittel für den Entwurf (ohne Implementierung)
  - Entwickler können sich mittels Interfaces über die Aufruf-Schnittstellen austauschen, ohne sich mit der eigentlichen Implementierung beschäftigen zu müssen.

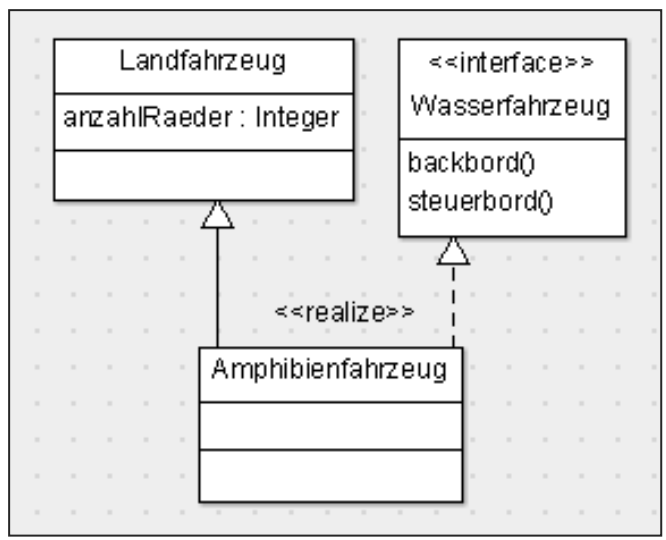


Konzept III

- Wozu braucht man Interfaces noch?
  - In Java gibt es keine Mehrfachvererbung



  - Möchte man Methoden aus mehr als einer Klassendeklaration erben, bieten Interfaces als Ersatzkonstrukt genau diese Möglichkeit.



Übersicht

Inhalte

- ✓ Konzept
- ✓ Implementierung

Interface vs. Abstrakte Klasse

Cloneable

Implementierung I

- Beispiel Fortbewegungsmittel

```
public interface Fortbewegungsmittel{

   public int kapazitaet();
   public double kilometerPreis();
}
```

```
public class Auto implements Fortbewegungsmittel{
    private String name;
    private int erstzulassung;
    private int anzahlSitze;
    private double spritVerbrauch;
    private double spritPreis;

    public int kapazitaet(){
        return anzahlSitze;
    }

    public double kilometerPreis(){
        return spritVerbrauch*spritPreis/100;
    }
}
```

Implementierung II

`implements`

- Die Implementierung eines Interfaces wird durch das Schlüsselwort `implements` bei der Klassendefinition angezeigt.

- Klassen können eine beliebige Anzahl von Interfaces implementieren (jeweils Komma-getrennt)

```
public class Oldtimer implements Fortbewegungsmittel, Sammlerstueck{
    …
}
```

- Was passiert, wenn zwei gleichzeitig implementierte Interfaces gleichlautende Methodensignaturen enthalten?
  - Es kann in der entsprechenden Klasse nur eine Methode implementiert werden.
  - Diese Methode muss beide „Verträge“ mit den Interfaces erfüllen

Aufgabe 11.01

Aufgabe 11.01 Interface Musikinstrument

Schreiben Sie das Interface Musikinstrument mit der Methode `spieleInstrument()`. Implementieren Sie dieses Interface in den beiden Klassen `Trommel` und `Trompete`. Das Musikinstrument soll hierbei beim Spielen eine entsprechende Ausgabe auf dem Bildschirm machen. So soll z.B. eine Trommel am Bildschirm „Trommel, Trommel“ ausgeben. Zum Testen der Klassen soll die Methode `main()` in der Klasse `Musikantenstadl` mehrere Musikinstrumente erzeugen und abspielen.

Übersicht

Inhalte

- ✓ Konzept
- ✓ Implementierung
- ✓ Interface vs. Abstrakte Klasse

Cloneable

Interface vs. Abstrakte Basisklasse I

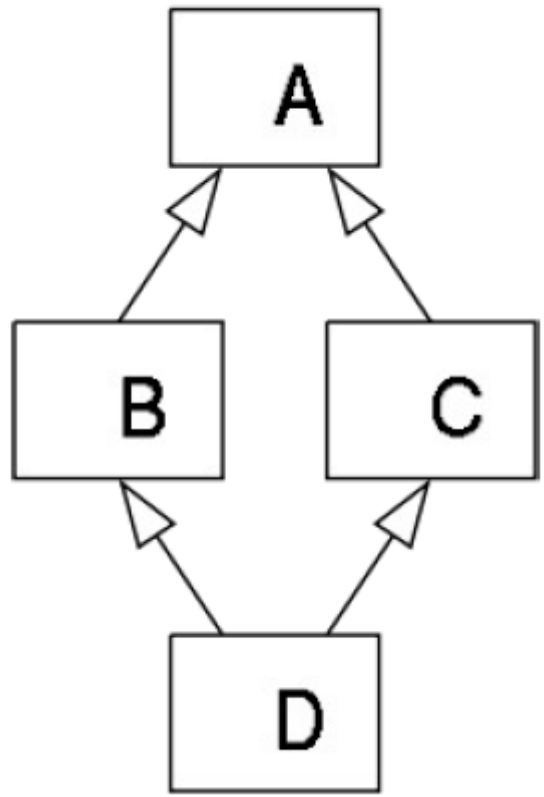
- Abstrakte Basisklassen können Variablen, Konstanten, implementierte und abstrakte Methoden enthalten.

- Schnittstellen können nur Konstanten und abstrakte Methoden enthalten.
  - Seit Java 8 auch default- und statische Methoden
  - Dadurch beherrscht Java eine gewisse Fähigkeit zur Mehrfachvererbung

Mehrfachvererbung

Was sind eigentlich die Probleme bei der Mehrfachvererbung, die dazu geführt haben, dass dieses Feature als so schwierig gilt?

Deadly Diamond of Death



Mehrfachvererbung

Java 8 kann Mehrfachvererbung

- seit Java 8 können Interfaces default-Methoden enthalten

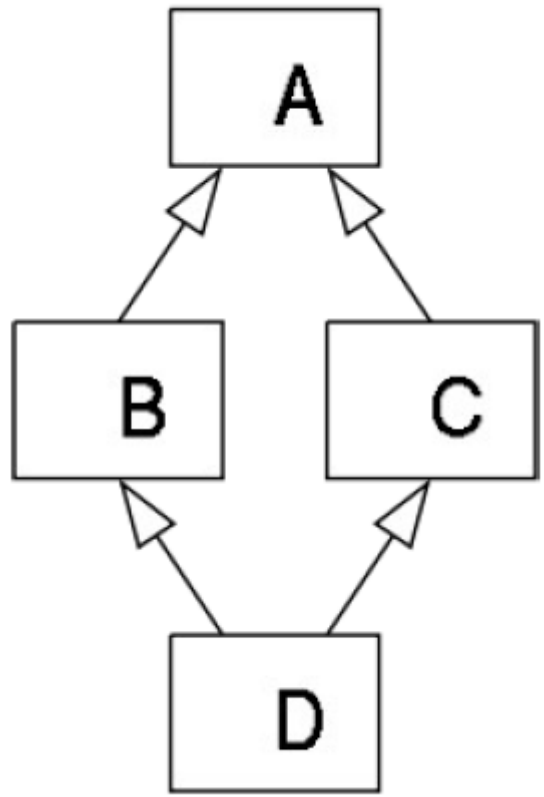
```
interface InterfaceA {

    public void saySomething();

    default public void sayHi() {
        System.out.println("Hi from InterfaceA");
    }

}
```

ggf. Demo (MehrfachVererbung.java)

Mehrfachvererbung

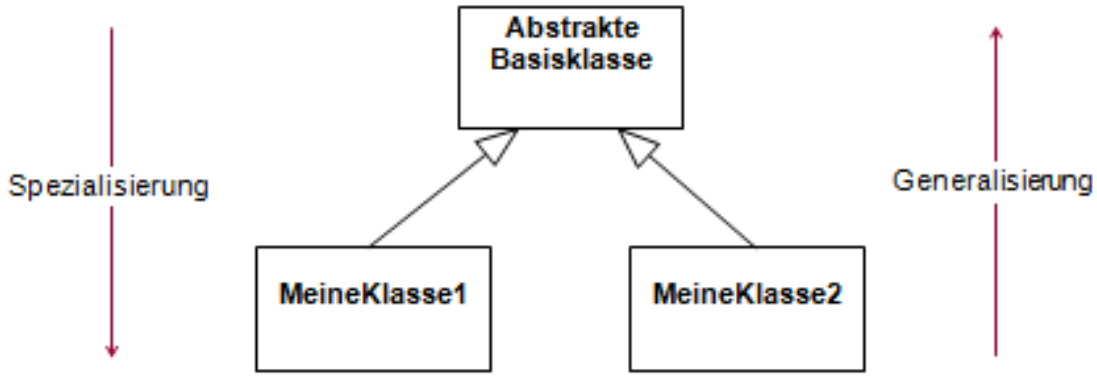
Java 8 kann Mehrfachvererbung

- Wenn D eine Klasse in Java ist, die von B und C Funktionalität erbt, dann muss mindestens einer der Typen B und C ein Interface sein, da es in Java auch weiterhin keine Mehrfachvererbung von Klassen gibt.
- Wenn aber mindestens B oder C ein Interface ist, dann muss auch A ein Interface sein, denn ein Interface kann in Java nicht von einer Klasse abgeleitet sein.
- Wenn nun A ein Interface ist, dann kann es zwar Funktionalität in Form von Default-Methoden enthalten, trotzdem hat A keine nicht statische und nicht finale Variablen. So dass sich die Frage, wie häufig die Variablen von A in D enthalten sein sollen, gar nicht stellt.

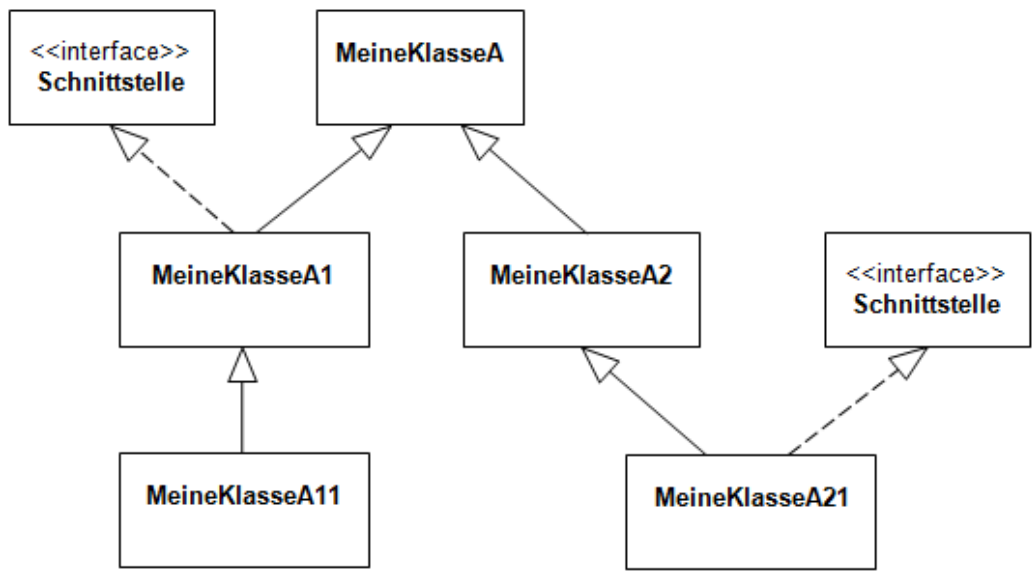


Interface vs. Abstrakte Basisklasse II

- Sowohl eine Unterklassenbildung aus einer abstrakten Basisklasse im Rahmen der Vererbung als auch eine Verfeinerung einer Schnittstelle stellt die Bildung eines Untertypen dar. Ein Objekt einer Klasse (die eine Schnittstelle implementiert) ist vom Typ seiner Klasse und vom Typ der Schnittstelle.



- Eine Schnittstelle kann von jeder beliebigen Klasse implementiert werden, ohne dass die Schnittstelle in den Klassenbaum eingeordnet werden muss.



Übersicht

Inhalte

- ✓ Konzept
- ✓ Implementierung
- ✓ Interface vs. Abstrakte Klasse
- ✓ Cloneable

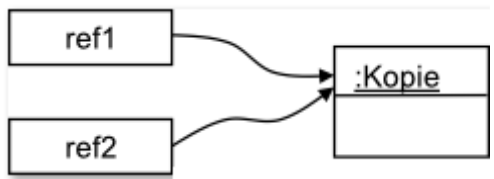
Cloneable I

- Das Interface Cloneable ist etwas untypisch, denn es besitzt weder Methoden noch Konstanten.
  - Es dient lediglich dazu, dem Compiler anzuzeigen, dass Objekte dieser Klasse sich selbst kopieren dürfen.

- Ein Objekt zu klonen bedeutet nicht anderes, als eine exakte Kopie davon zu erstellen.
  - man erhält eine Referenz auf ein neues Objekt, welches die gleichen Werte besitzt, wie die Klonvorlage.

So erstellen wir lediglich eine neue Referenz auf dasselbe Objekt (**keine Kopie!**):

```
…
Kopie ref1 = new Kopie(1);
Kopie ref2 = ref1;
…
```

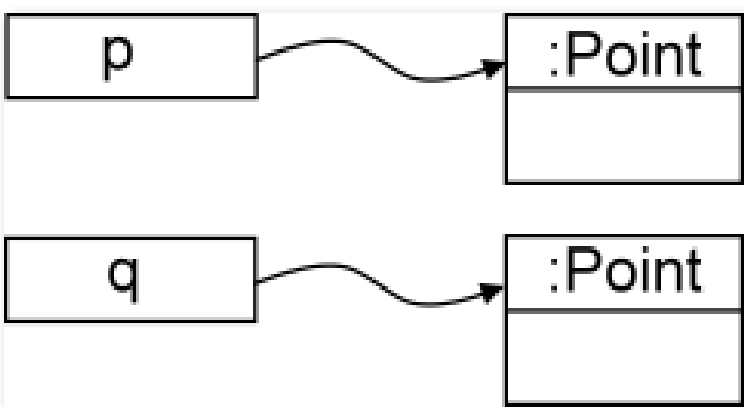


Kopie kommt auf nächster Folie!

Cloneable II

- Mehr als 300 Klassen der Java-Bibliothek unterstützen ein clone(), was ein neues Exemplar mit dem gleichen Zustand zurückgibt.

```
…
java.awt.Point p = new java.awt.Point( 12, 23 );

java.awt.Point q = (java.awt.Point) p.clone();

System.out.println( q );    // java.awt.Point[x=12,y=23]
…
```



Cloneable III

- `clone()`- Methode für eigene Klassen
  - Eigene Klassen überschreiben die protected-Funktion `clone()` aus der Oberklasse `java.lang.Object` und machen sie `public`.
  - Für die Implementierung kommen zwei Möglichkeiten in Betracht:
    1. Von Hand ein neues Objekt anlegen, alle Attribute kopieren und die Referenz auf das neue Objekt zurückgeben.
    2. Das Laufzeitsystem soll selbst eine Kopie anlegen, und diese geben wir zurück

```
public class Player implements Cloneable{

    public String name;
    public int age;

    public Player clone() throws CloneNotSupportedException{
        // hiermit wird die überschriebene clone()-Methode
        // der Klasse Object aufgerufen.
        return (Player) super.clone();
    }
}
```

Aufgabe 11.02

Aufgabe 11.02 Drucker

Schreiben Sie das Interface Drucker mit den Methoden `drucke(String s)` und `getSeiten()`.

- `drucke(String s)` gibt die Zeichenkette auf der Systemausgabe aus und dekrementiert den Seitenzähler um 1.
- `int getSeiten()` gibt die Anzahl der noch verfügbaren Seiten zurück

Implementieren Sie dieses Interface `Drucker` und das Interface `Cloneable` (mit der Methode `clone()`) in der Klasse `Laserdrucker`. Standardmäßig hat ein Laserdrucker zu Beginn 100 Seiten.

Im Online-Campus finden Sie die Datei `DruckerTest.java`, welche zum Testen Ihrer Implementierung verwendet werden soll. Importieren Sie diese Datei nach Eclipse.

Aufgabe 11.03

Aufgabe 11.03 Fliegen

a) Entwickeln Sie ein Interface namens `Fliegen`, das eine Methode `fliegen()` deklariert. Diese Methode soll keinen Rückgabewert besitzen und keine Parameter erwarten.

b) Entwickeln Sie dann eine Klasse Biene, die das Interface `Fliegen` implementiert. Geben Sie in der zu implementierenden Methode aus dem Interface den String ”Summsumm” auf der Standardausgabe aus.

c) Entwickeln Sie eine Java-Klasse `Aufg_11_03` und erstellen Sie innerhalb der Methode `main()` ein Objekt basierend auf der Klasse `Biene`. Rufen Sie eine in `Aufg_11_03` neu zu definierende Methode `abflug(Fliegen f)` auf, der Sie das Objekt vom Typ Biene übergeben. Die Methode `abflug(Fliegen f)` soll keinen Rückgabewert besitzen und ein Objekt vom Typ des Interfaces `Fliegen` erwarten. Rufen Sie für den Parameter vom Typ Fliegen innerhalb der Methode `abflug(Fliegen f)` die Methode `fliegen()` auf, und kompilieren und führen Sie die Anwendung dann aus.

d) Modifizieren Sie das in Teilaufgabe a) entwickelte Interface `Fliegen` und wandeln Sie es in eine abstrakte Klasse um. Passen Sie Ihre Klasse `Biene` so an, dass die entwickelte Java-Applikation immer noch funktioniert.