CARL VON OSSIETZKY
universität OLDENBURG

Sommersemester 2020

Objektorientierte Modellierung und Programmierung

Dr. Christian Schönberg

CARL VON OSSIETZKY universität OLDENBURG

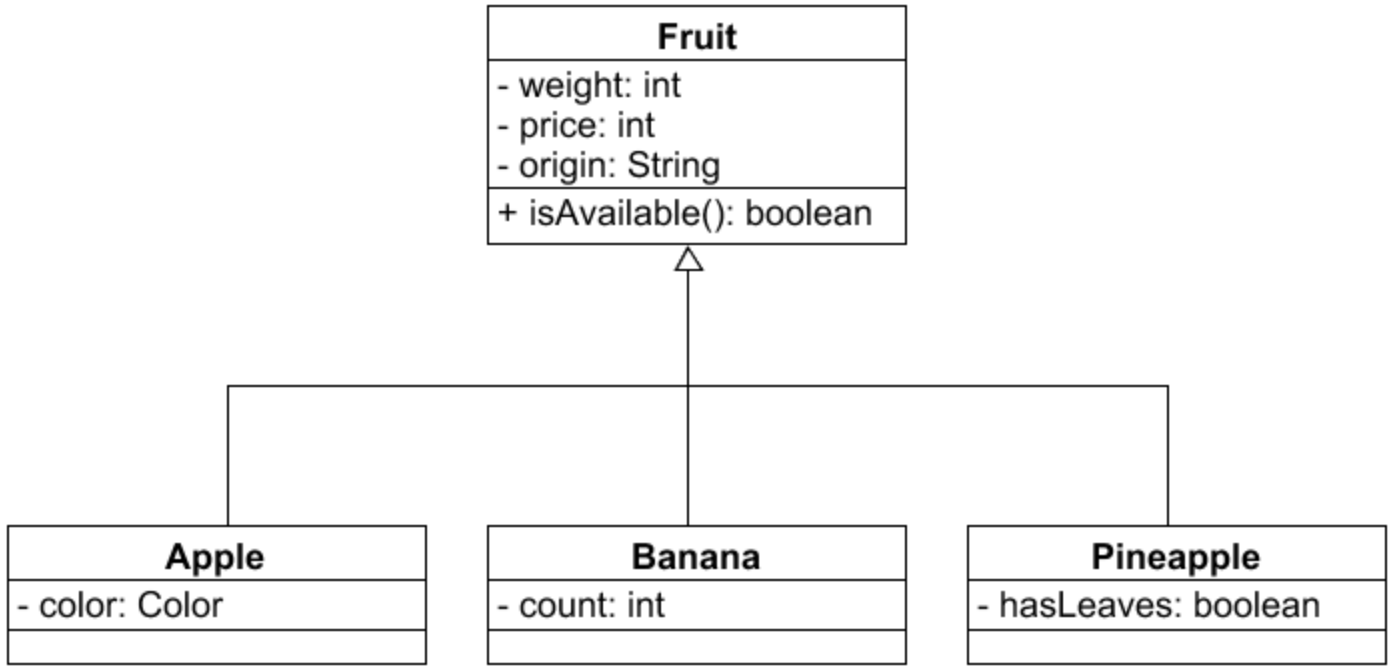
Interfaces und Mehrfachvererbung

CARL VON OSSIETZKY universität OLDENBURG

Lernziele

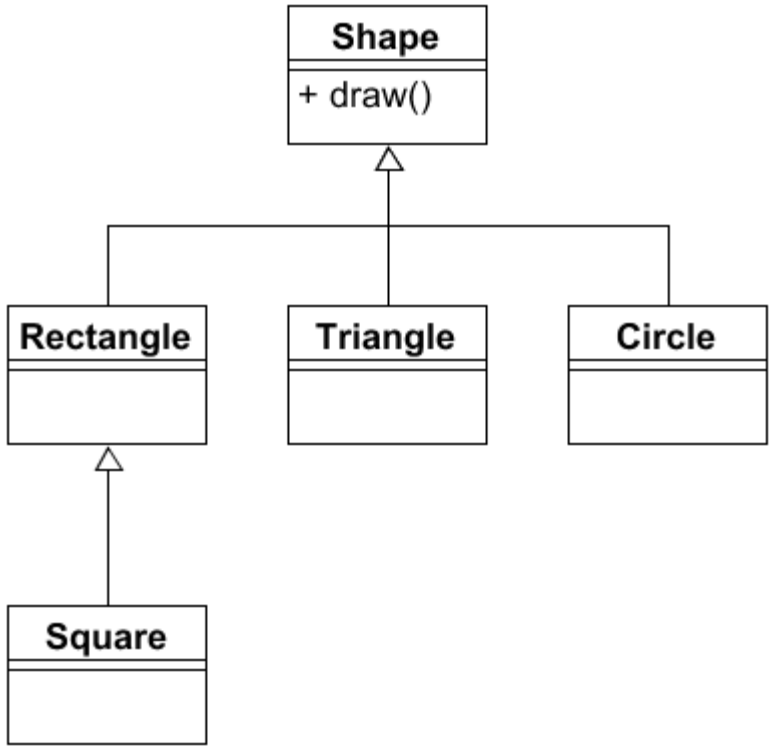
- Abstrakte Klassen
- Interfaces
- Lösungsalternativen
- Strategy-Pattern

Motivation

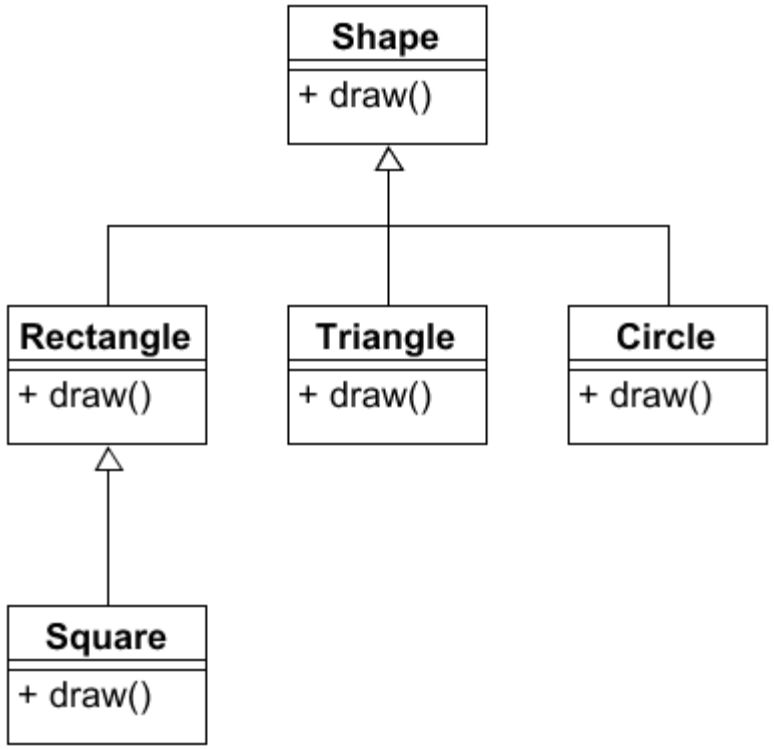



```
Fruit fruit = new Fruit();
```

Motivation (2)



Motivation (2)



CARL VON OSSIETZKY universität OLDENBURG

Abstrakte Klassen

Definition: Abstrakte Klasse

- Eine abstrakte Klasse ist eine Klasse, die als **`abstract`** deklariert ist
- Eine abstrakte Klasse kann **nicht** instanziiert werden
- Eine abstrakte Klasse kann spezialisiert werden
  → es können Unterklassen definiert werden
- Eine abstrakte Klasse kann abstrakte Methoden enthalten
- Abstrakte Methoden sind Methoden, die als **`abstract`** deklariert sind und die **keine** Implementierung haben
- Klassen, die abstrakte Methoden enthalten, müssen abstrakt sein

Beispiel: Abstrakte Klasse

```java
public abstract class Shape {

    public abstract void draw();

}
```

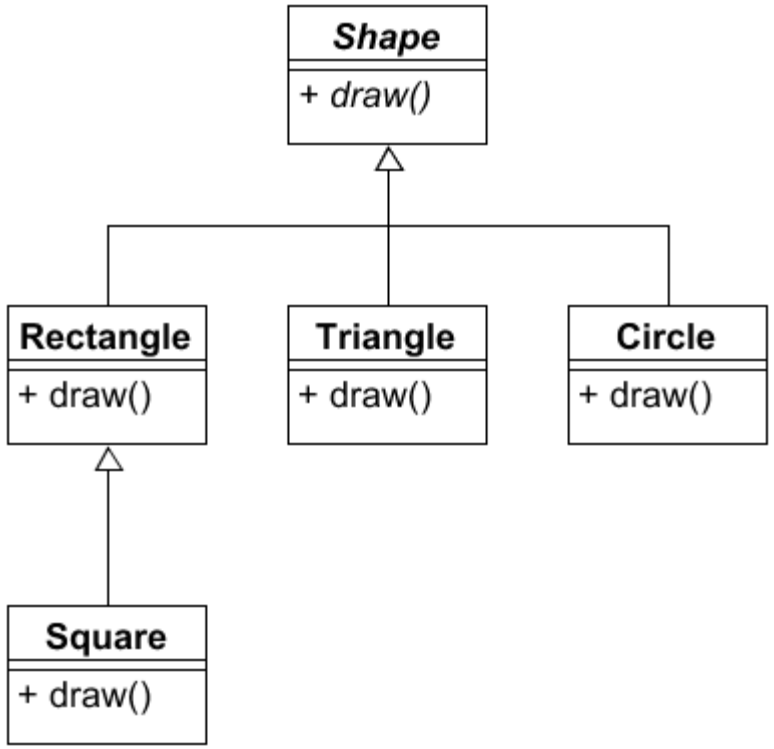
```java
public class Rectangle extends Shape {

    @Override
    public void draw() {
        // ...
    }

}
```

```java
public class Circle extends Shape {

    @Override
    public void draw() {
        // ...
    }

}
```

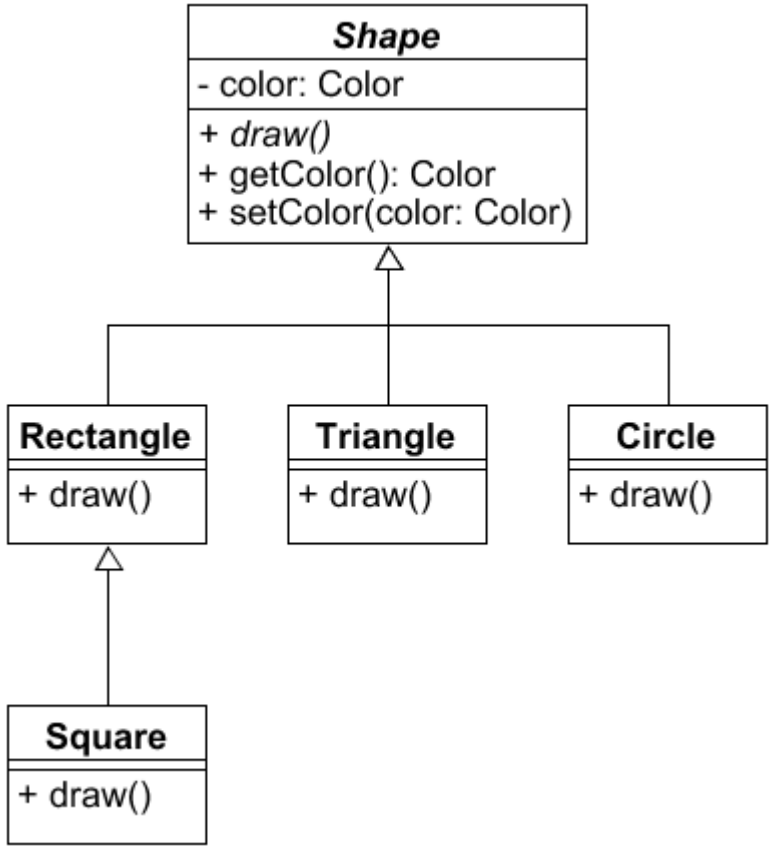
Nicht-abstrakte Unterklassen von abstrakten Klassen **müssen** die abstrakten Methoden der Oberklasse(n) überschreiben und implementieren.

Abstrakte Klassen in der UML

/text in italics/



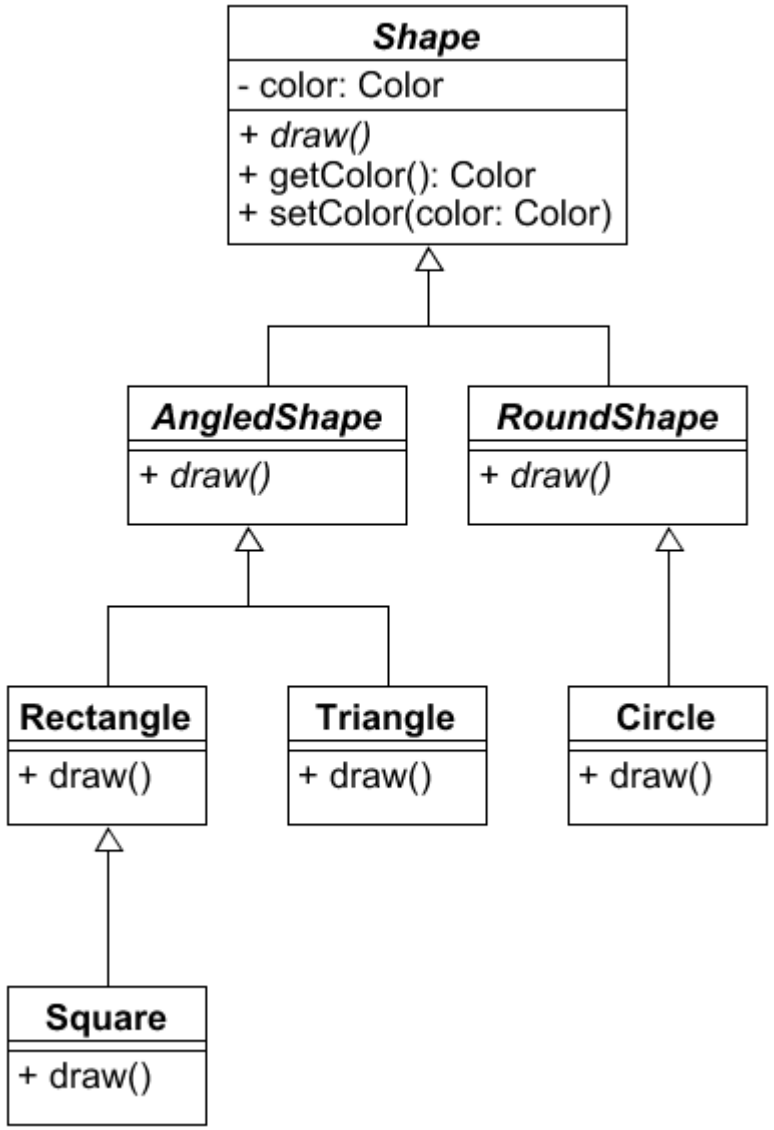
Abstrakte Klassen



Abstrakte Klassen

```
public abstract class Shape {

    private Color color;

    public Color getColor() {
        return color;
    }

    public void setColor(Color color) {
        this.color = color;
    }

    public abstract void draw();

}
```

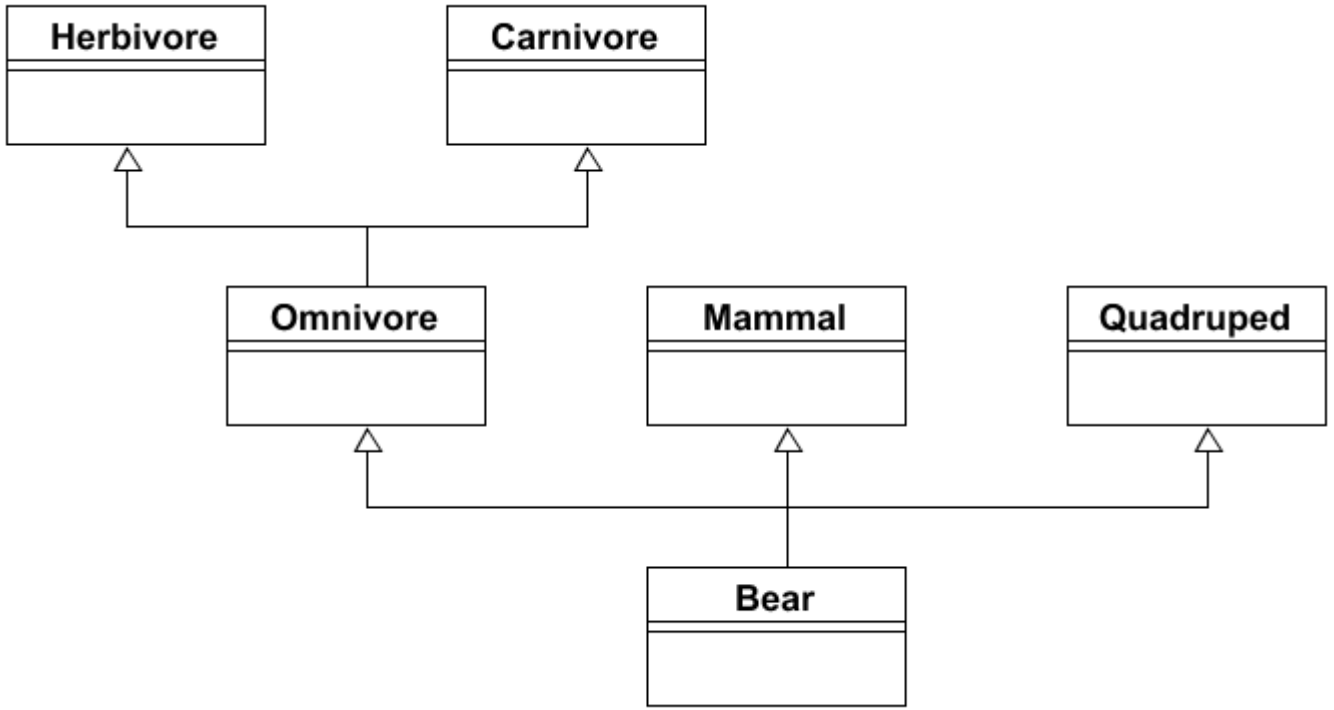
Abstrakte Klassen (2)



CARL VON OSSIETZKY universität OLDENBURG

Interfaces

Motivation



CARL VON OSSIETZKY universität OLDENBURG

Motivation (2)

```
List list = new LinkedList();
            new DoublyLinkedList();
            new ArrayList();
            // ...
list.add("1");
list.add("2");
list.add("3");
list.remove("2");
```

Beispiel: Interface

```
public interface List {

    public void add(Object data);

    public void insert(int index, Object data);

    public Object get(int index);

    public int find(Object data);

    public Object remove(int index);

    public void remove(Object data);

}
```

Beispiel: Interface

```
public interface List {

    void add(Object data);

    void insert(int index, Object data);

    Object get(int index);

    int find(Object data);

    Object remove(int index);

    void remove(Object data);

}
```

Alle in einem Interface deklarierten Methodensignaturen müssen **`public`** sein, deshalb kann/soll der Sichtbarkeitsmodifikator hier weggelassen werden.
In der implementierenden Klasse muss er aber wieder angegeben werden.

CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Interface

```
public class LinkedList implements List {

    @Override
    public void add(Object data) { … }

    @Override
    public void insert(int index, Object data) { … }

    @Override
    public Object get(int index) { … }

    @Override
    public int find(Object data) { … }

    @Override
    public Object remove(int index) { … }

    @Override
    public void remove(Object data) { … }

}
```

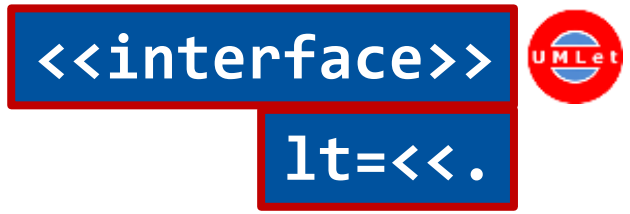
Beispiel: Interface

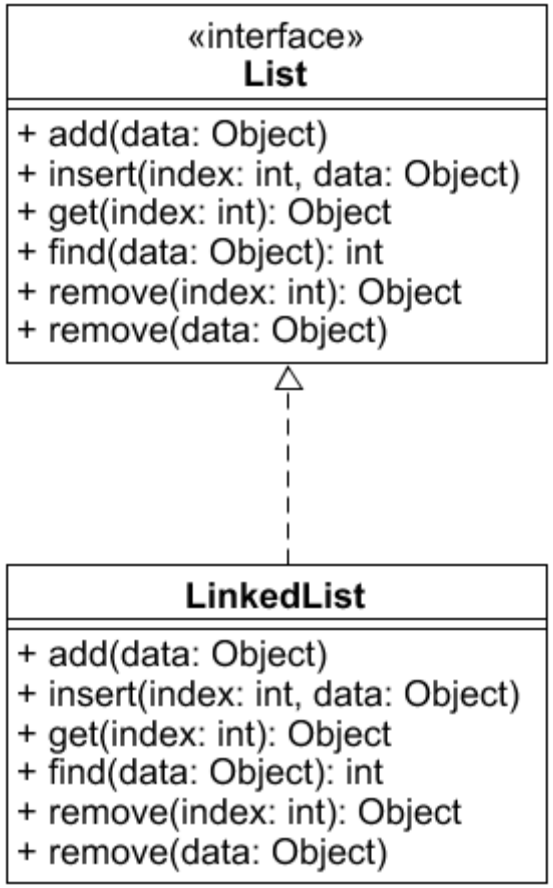
```
public class ListUser {

    public static void main(String[] args) {
        List list = ListUser.getListInstance();
        list.add("1");
        list.add("2");
        list.add("3");
        list.remove("2");
    }

    private static List getListInstance() {
        return new LinkedList();
    }

}
```

Definition: Interface

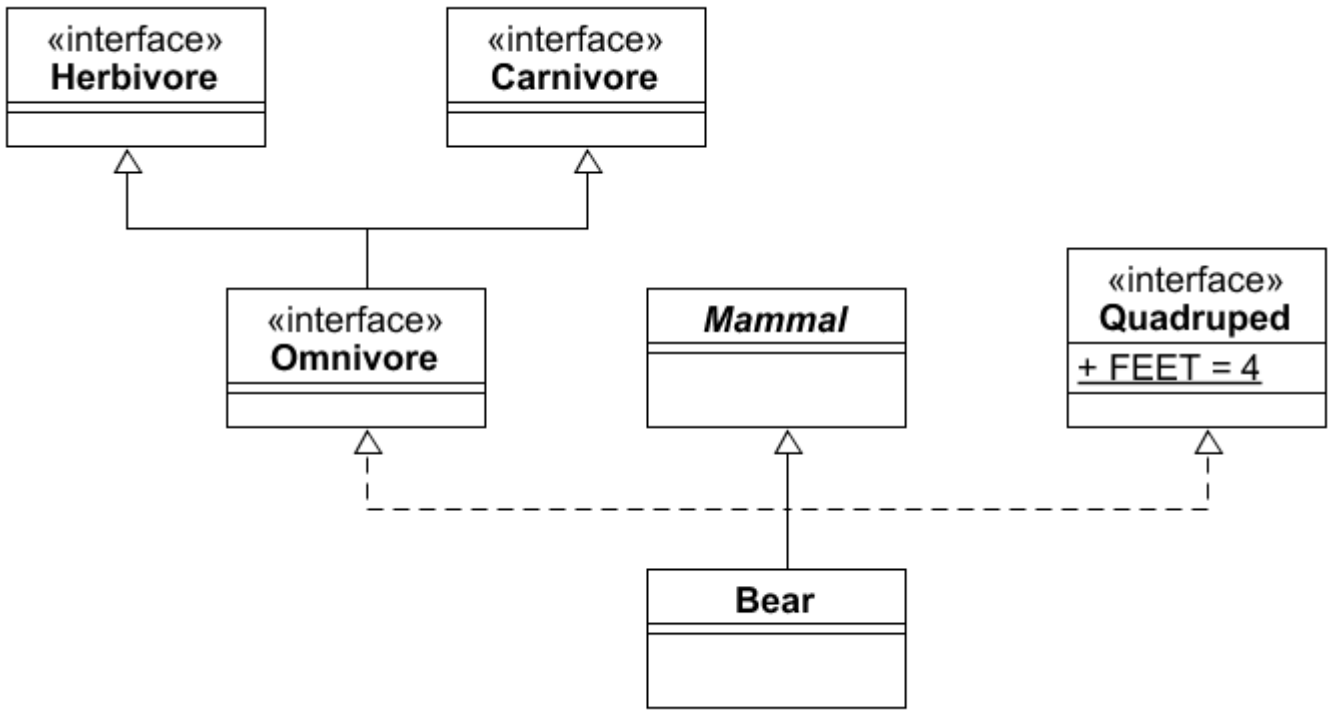
- In einem Interface können **keine** Attribute definiert werden
  - Interfaces definieren keinen internen Zustand
  - Alle angegebenen Attribute sind automatisch Konstanten (**`public static final`**)
- In einem Interface können nur Methodensignaturen angegeben werden, keine Implementierungen
  - alle Methoden sind daher automatisch **`abstract`**
  - alle Methodensignaturen sind automatisch **`public`**
  - seit Java 8 nicht mehr ganz korrekt → später
- Klassen können von genau einer Klasse erben (**`extends`**), aber (zusätzlich) mehrere Interfaces implementieren (**`implements`**)
- Interfaces können andere Interfaces erweitern (**`extends`**)
- Interfaces definieren einen Typ mit dem Namen des Interfaces
- Es lassen sich Objektvariablen vom Typ eines Interface definieren
- Objektvariablen vom Typ eines Interface sind polymorph zu Objekten von Klassen, die das Interface implementieren

Interfaces in der UML





Interfaces in der UML (2)



Beispiel: Interfaces

```
public interface Herbivore { }
```

```
public interface Carnivore { }
```

```
public interface Omnivore extends Carnivore, Herbivore { }
```

```
public abstract class Mammal { }
```

```
public interface Quadruped {
    int FEET = 4;
}
```

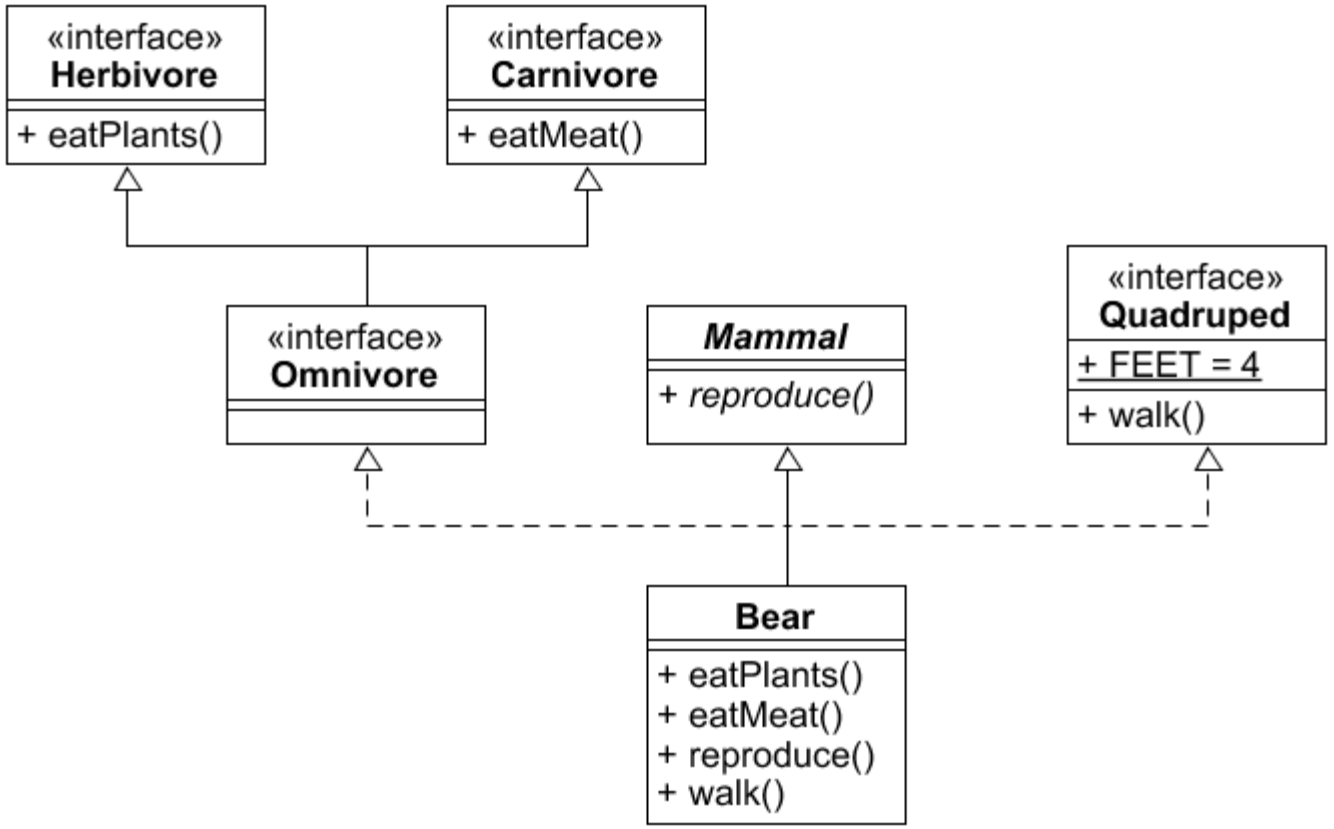
```
public class Bear extends Mammal implements Omnivore, Quadruped { }
```

CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Interfaces

```
Herbivore h = new Bear();
Carnivore c = new Bear();
Omnivore o = new Bear();
Mammal m = new Bear();
Quadruped q = new Bear();
Bear bear = new Bear();
```

Beispiel: Interfaces (2)



Beispiel: Interfaces (2)

```
public interface Herbivore {
    void eatPlants();
}
```

```
public interface Carnivore {
    void eatMeat();
}
```

```
public interface Omnivore extends Carnivore, Herbivore { }
```

```
public abstract class Mammal {
    public abstract void reproduce();
}
```

```
public interface Quadruped {
    int FEET = 4;
    void walk();
}
```

Beispiel: Interfaces (2)

```
public class Bear extends Mammal implements Omnivore, Quadruped {

    @Override
    public void eatMeat() { }

    @Override
    public void eatPlants() { }

    @Override
    public void walk() { }

    @Override
    public void reproduce() { }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Methoden in Interfaces

```
public interface List {

    public static void sort(List list) {
        // implement sorting alg.
    }

    void add(Object data);
    void insert(int index, Object data);
    Object get(int index);
    int find(Object data);
    Object remove(int index);

    default void remove(Object data) {
        int index = find(data);
        if (index > -1) {
            remove(index);
        }
    }

}
```

Ab Java 8: Statische und Default-Methoden

- Statische Methoden in Interfaces
  - Klassenmethoden können sowieso nicht auf den internen Objektzustand zugreifen
    - statische Methoden sind objektunabhängig
    - statische Methoden können problemlos in Interfaces implementiert werden
    - statische Methoden sind nicht automatisch **`public`**

- Default-Methoden in Interfaces
  - in einigen Fällen ist es sinnvoll, eine Standard-Implementierung für Methoden auf Schnittstellen-Ebene anzubieten
    - kein Zugriff auf internen Objektzustand über Attribute
    - aber Zugriff auf Methoden, inkl. getter-/setter-Methoden
    - dadurch indirekt Zugriff auf Objektzustand zur Laufzeit nach Instanziierung
    - Default-Methoden sind automatisch **`public`**

CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Default-Methoden in Interfaces

```
public interface List {

    void add(Object data);
    void insert(int index, Object data);
    Object get(int index);
    int find(Object data);
    Object remove(int index);

    default void remove(Object data) {
        int index = find(data);
        if (index > -1) {
            remove(index);
        }
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Default-Methoden in Interfaces

```
public class LinkedList implements List {

    @Override
    public void add(Object data) { … }

    @Override
    public void insert(int index, Object data) { … }

    @Override
    public Object get(int index) { … }

    @Override
    public int find(Object data) { … }

    @Override
    public Object remove(int index) { … }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Default-Methoden in Interfaces

```
public class ListUser {

    public static void main(String[] args) {
        List list = new LinkedList();
        list.add("1");
        list.add("2");
        list.add("3");
        list.remove("2");
    }

}
```

Default-Methoden und Vererbung

```java
interface Interface {
    default void doSomething() {
        System.out.println("Interface");
    }
}
```

```java
class SuperClass {
    public void doSomething() {
        System.out.println("SuperClass");
    }
}
```

```java
class SubClass extends SuperClass implements Interface { }
```

```java
SubClass instance = new SubClass();
instance.doSomething();
```

SuperClass

Default-Methoden und Vererbung

```java
interface SuperInterface {
    default void doSomething() {
        System.out.println("SuperInterface");
    }
}
```

```java
interface Interface extends SuperInterface {
    default void doSomething() {
        System.out.println("Interface");
    }
}
```

```java
class SubClass implements Interface { }
```

```java
SubClass instance = new SubClass();
instance.doSomething();
```

```
Interface
```

Default-Methoden und Vererbung

```
interface Interface1 {
    default void doSomething() {
        System.out.println("Interface 1");
    }
}
```

```
interface Interface2 {
    default void doSomething() {
        System.out.println("Interface 2");
    }
}
```

```
class SubClass implements Interface1, Interface2 { }
```

Duplicate default methods named **doSomething** with the parameters **()** and **()** are inherited from the types **Interface2** and **Interface1**

Default-Methoden und Vererbung

```java
interface Interface1 {
    default void doSomething() {
        System.out.println("Interface 1");
    }
}
```

```java
interface Interface2 {
    default void doSomething() {
        System.out.println("Interface 2");
    }
}
```

```java
class SubClass implements Interface1, Interface2 {
    @Override
    public void doSomething() {
        Interface1.super.doSomething();
    }
}
```

Interfaces und Abstrakte Klassen

- Gemeinsamkeiten
  - beide können nicht instanziiert werden
  - beide können abstrakte Methoden enthalten
    - ➢ diese Methoden müssen dann von abgeleiteten Klassen (implementierenden Klassen bzw. Unterklassen) implementiert werden
- Unterschiede
  - Interfaces können keine Attribute enthalten
  - Interfaces können nur **`public`**-Methoden(-signaturen) definieren
  - Default-Implementierungen haben keinen (direkten) Zugriff auf den Objektzustand
  - Klassen können nur von einer (abstrakten) Klasse erben, aber beliebig viele Interfaces implementieren

Interfaces und Abstrakte Klassen

	Klasse	Abstrakte Klasse	Interface
instanziierbar	ja	nein	nein
Attribute	ja	ja	nein
Konstanten	ja	ja	ja
abstrakte Methoden	nein	ja	ja
statische Methoden	ja	ja	ja
default Methoden	nein	nein	ja
implementierte Methoden	ja	ja	nur default-Methoden
public Methoden	ja	ja	ja
private/protected Methoden	ja	ja	nein
andere Klasse kann von mehreren … erben	nein	nein	ja

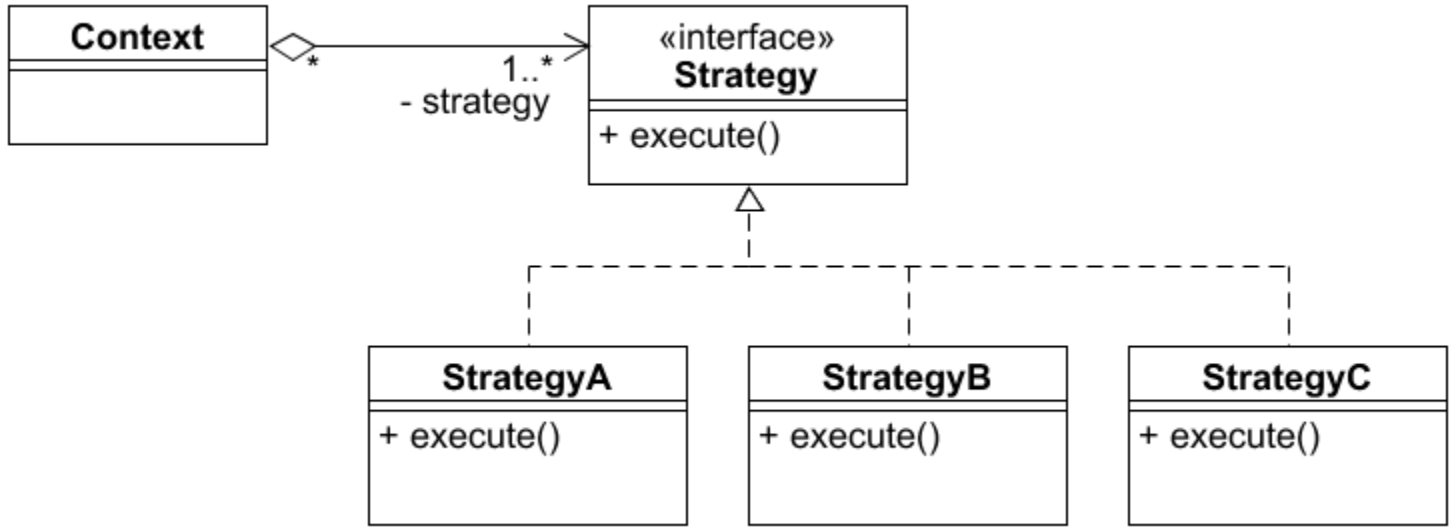
CARL VON OSSIETZKY universität OLDENBURG

Strategy Pattern

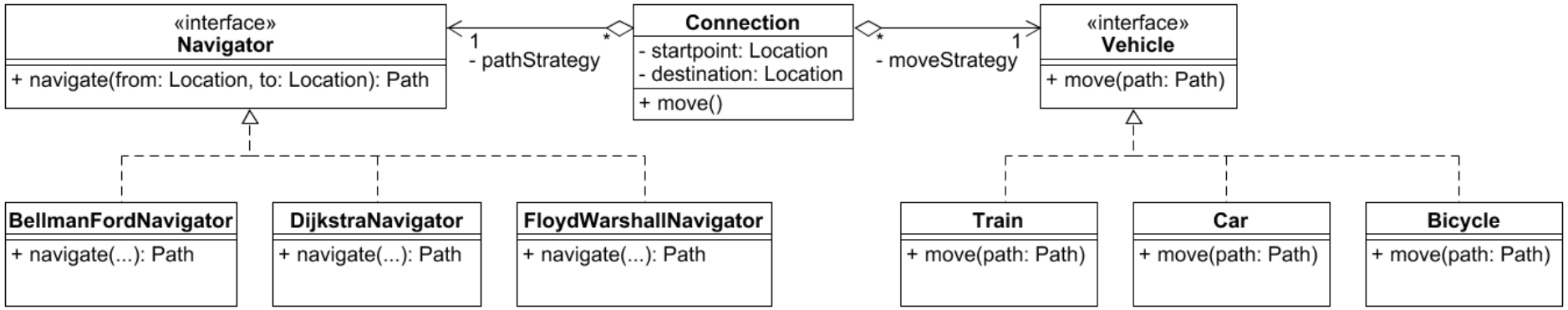
Motivation

- Lösung für spezielle Fragestellung gesucht
  - Algorithmus zum Sortieren
  - Algorithmus für kürzeste Wege in Graphen
  - Transport zu einem bestimmten Ort
- Konkretes Lösungsmittel ist austauschbar
  - Quicksort, Heapsort, Mergesort
  - Bellman-Ford, Dijkstra, Floyd-Warshall
  - Auto, Zug, Fahrrad
- Lösung
  - Problemstellung über Interface beschreiben
  - verschiedene Implementierungen des Interfaces anbieten, die im Kontext verwendet werden

Strategy Pattern



Beispiel: Strategy Pattern



CARL VON OSSIETZKY universität OLDENBURG

Lernziele

- Abstrakte Klassen
- Interfaces
- Lösungsalternativen
- Strategy-Pattern