CARL VON OSSIETZKY universität OLDENBURG

Sommersemester 2020

Objektorientierte Modellierung und Programmierung

Dr. Christian Schönberg

CARL VON OSSIETZKY universität OLDENBURG

Vererbung

CARL VON OSSIETZKY universität OLDENBURG

Lernziele

- Vererbung
  - in Java
  - in der UML
- `this()`, `super()`
- `Object`
- Sichtbarkeit
- Mehrfachvererbung

CARL VON OSSIETZKY universität OLDENBURG

Modellierung mit der UML

CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Listen

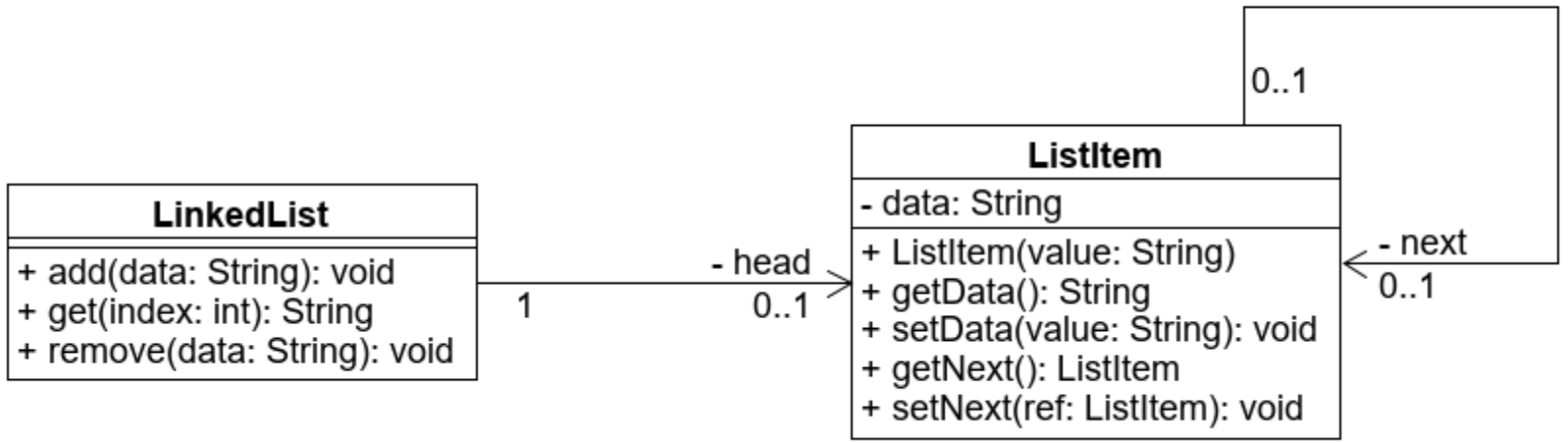
```
public class LinkedList {

    private ListItem head;

    public void add(String data) { … }

    public String get(int index) { … }

    public void remove(String data) { … }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Listen (2)

```
class ListItem {

    private String data;
    private ListItem next;

    public ListItem(String value) { … }
    public String getData() { … }
    public void setData(String value) { … }
    public ListItem getNext() { … }
    public void setNext(ListItem ref) { … }
}
```

Beispiel: Listen Klassendiagramm

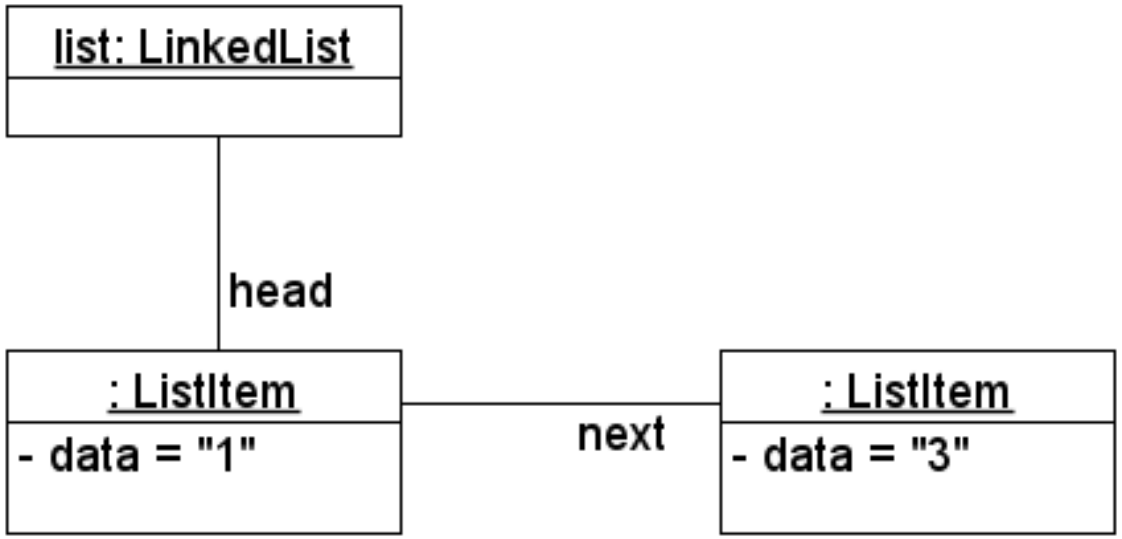


CARL VON OSSIETZKY universität OLDENBURG

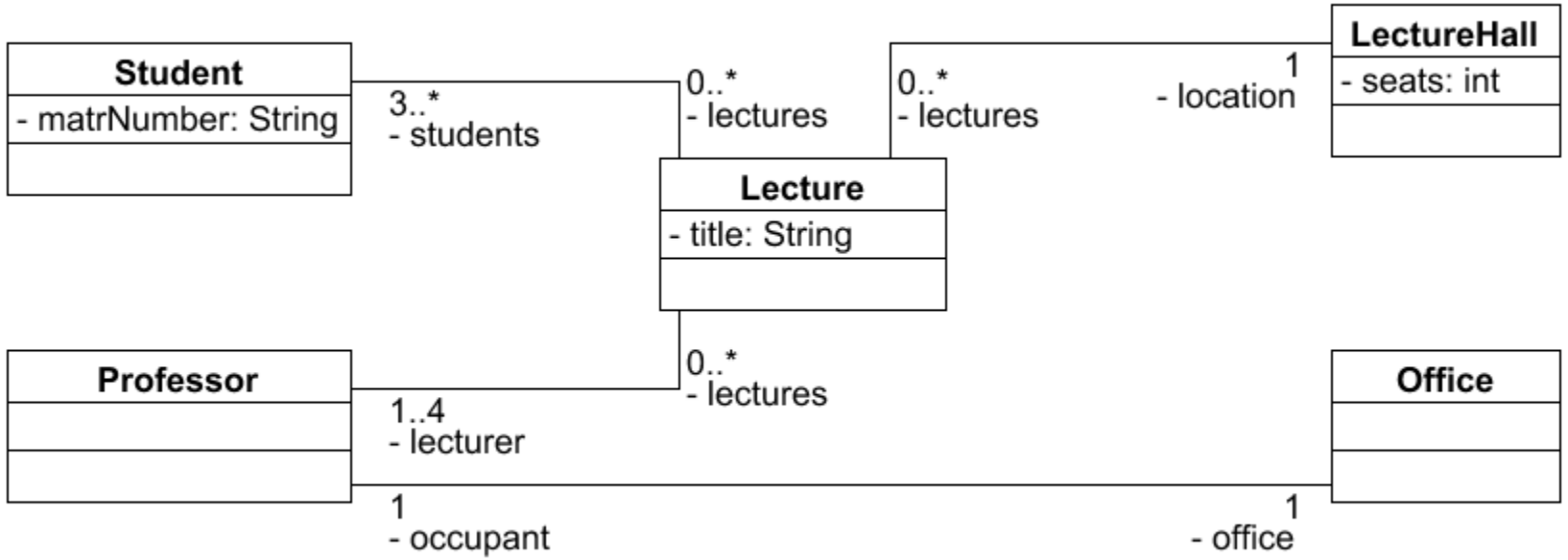
Beispiel: Listen (3)

```
public class ListProgram {

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

}
```

Beispiel: Listen Objektmodell



Beispiel: Klassendiagramm



CARL VON OSSIETZKY universität OLDENBURG

Motivation

Motivation

- Farbe
- Gewicht
- Preis
- Herkunft
- Verfügbarkeit

Motivation

- Farbe
- Gewicht
- Preis
- Herkunft
- Verfügbarkeit

- Anzahl
- Gewicht
- Preis
- Herkunft
- Verfügbarkeit

Motivation

- Farbe
- Gewicht
- Preis
- Herkunft
- Verfügbarkeit

- Anzahl
- Gewicht
- Preis
- Herkunft
- Verfügbarkeit

- Blätter vorhanden
- Gewicht
- Preis
- Herkunft
- Verfügbarkeit

Motivation

Apple
- color: Color - weight: int - price: int - origin: String
+ isAvailable(): boolean

Banana
- count: int - weight: int - price: int - origin: String
+ isAvailable(): boolean

Pineapple
- hasLeaves: boolean - weight: int - price: int - origin: String
+ isAvailable(): boolean

CARL VON OSSIETZKY universität OLDENBURG

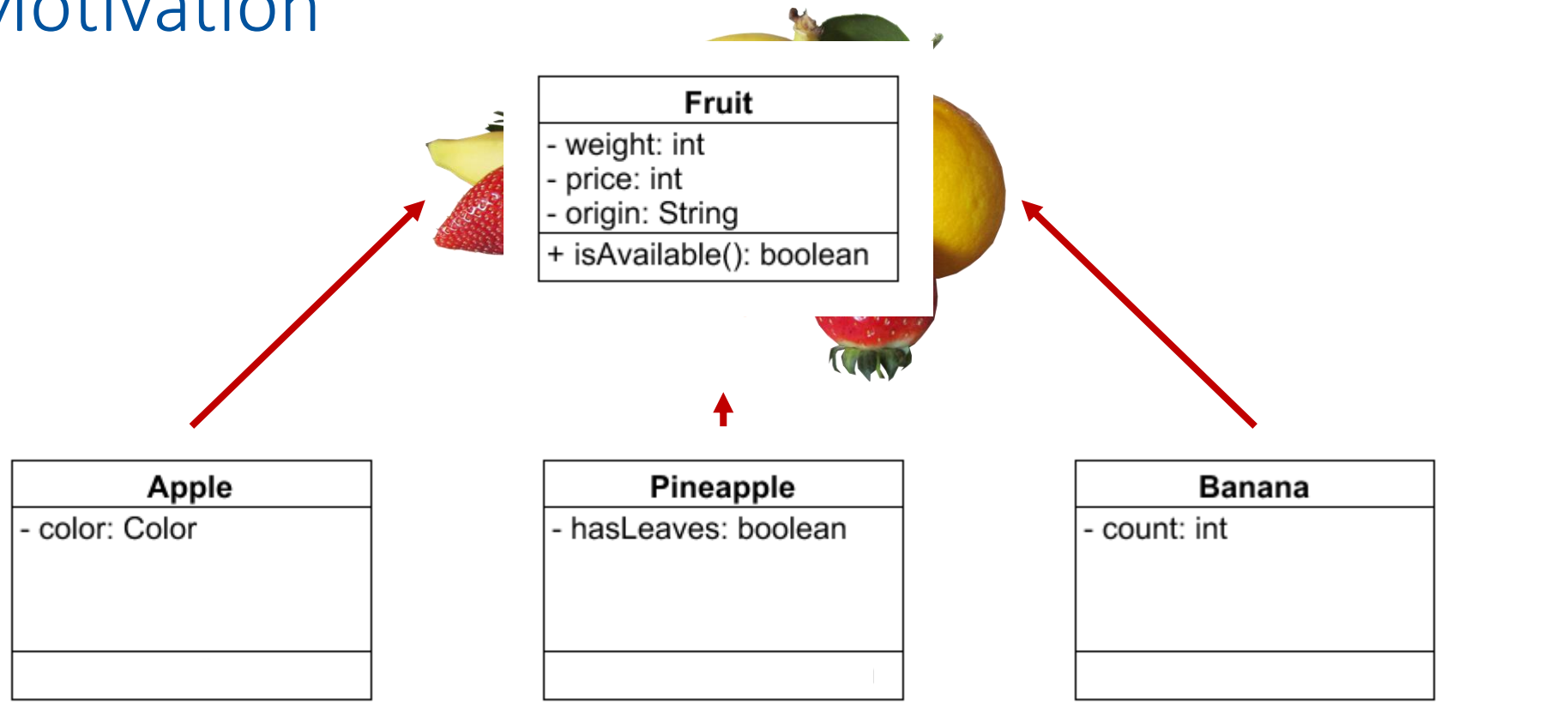
Motivation

Apple
- color: Color - weight: int - price: int - origin: String
+ isAvailable(): boolean

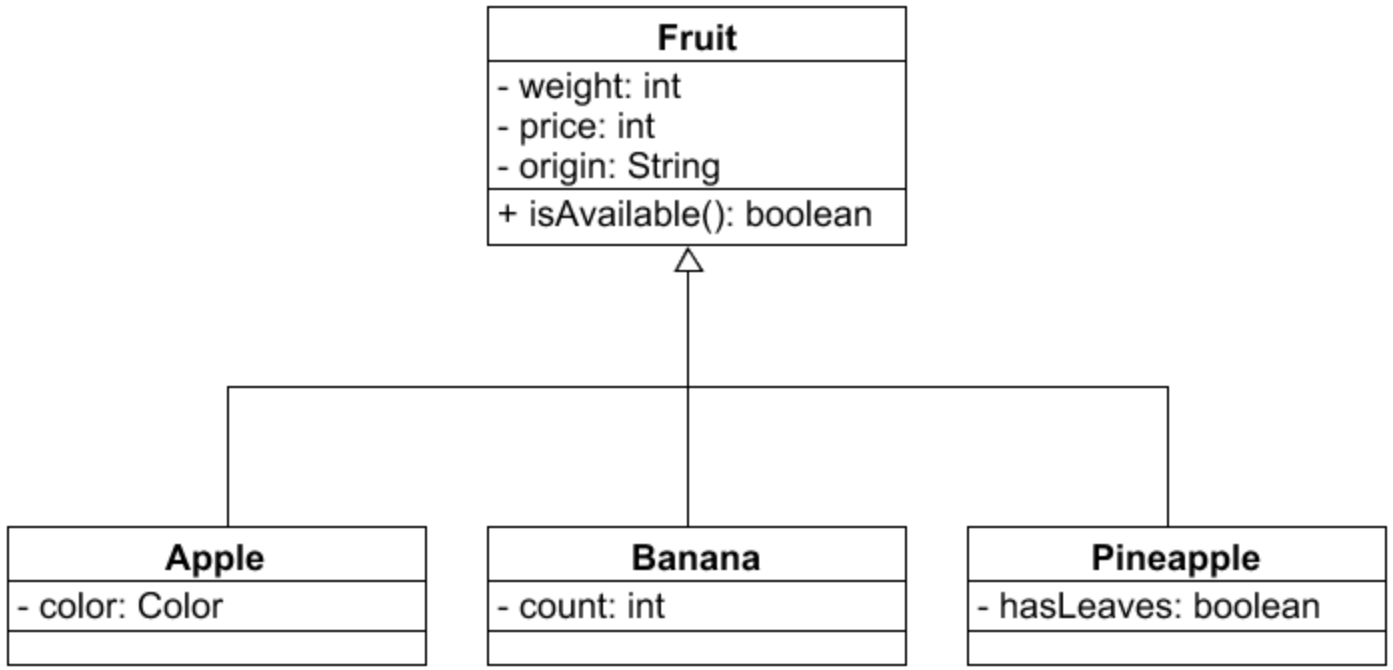
Pineapple
- hasLeaves: boolean - weight: int - price: int - origin: String
+ isAvailable(): boolean

Banana
- count: int - weight: int - price: int - origin: String
+ isAvailable(): boolean

Motivation


Fruit
- weight: int
- price: int
- origin: String
+ isAvailable(): boolean
Apple
- color: Color
Pineapple
- hasLeaves: boolean
Banana
- count: int

Motivation


Fruit
- weight: int
- price: int
- origin: String
+ isAvailable(): boolean
Apple
- color: Color
Banana
- count: int
Pineapple
- hasLeaves: boolean

CARL VON OSSIETZKY universität OLDENBURG

Motivation (2)

- Gleichartige Objekte → Erstellung einer Klasse
- Gleichartige Klassen → ?
- Beispiele
  - Abstraktion
    - z.B. verschiedene Sortierungsalgorithmen (BubbleSort, QuickSort, HeapSort, …)
  - Erweiterung
    - z.B. Heap zu einer Prioritäts-Warteschlange (Priority Queue)
  - Spezialisierung
    - z.B. Rechteck zum Quadrat

CARL VON OSSIETZKY universität OLDENBURG

Vererbung

Idee (Generalisierung)

- Nimm mehrere ähnliche Klassen
  - z.B. **`Apple`**, **`Banana`** und **`Pineapple`**
- Ziehe die gemeinsamen Attribute und Methoden heraus
- Erstelle eine neue allgemeinere Klasse und füge ihr diese gemeinsamen Attribute und Methoden hinzu
  - z.B. **`Fruit`**
- Die ähnlichen Klassen (Unterklassen) erben nun alle Attribute und Methoden der neuen Klasse (Oberklasse)

CARL VON OSSIETZKY universität OLDENBURG

Idee (Spezialisierung)

- Nimm eine allgemeine Klasse
  - z.B. **`Fruit`**
- Leite mehrere speziellere Klassen (Unterklassen) davon ab, die alle Attribute und Methoden (Eigenschaften und Verhalten) der allgemeinen Klasse (Oberklasse) erben, aber zusätzlich noch eigene spezifische Attribute und Methoden haben
  - z.B. **`Apple`**, **`Banana`**, **`Pineapple`**

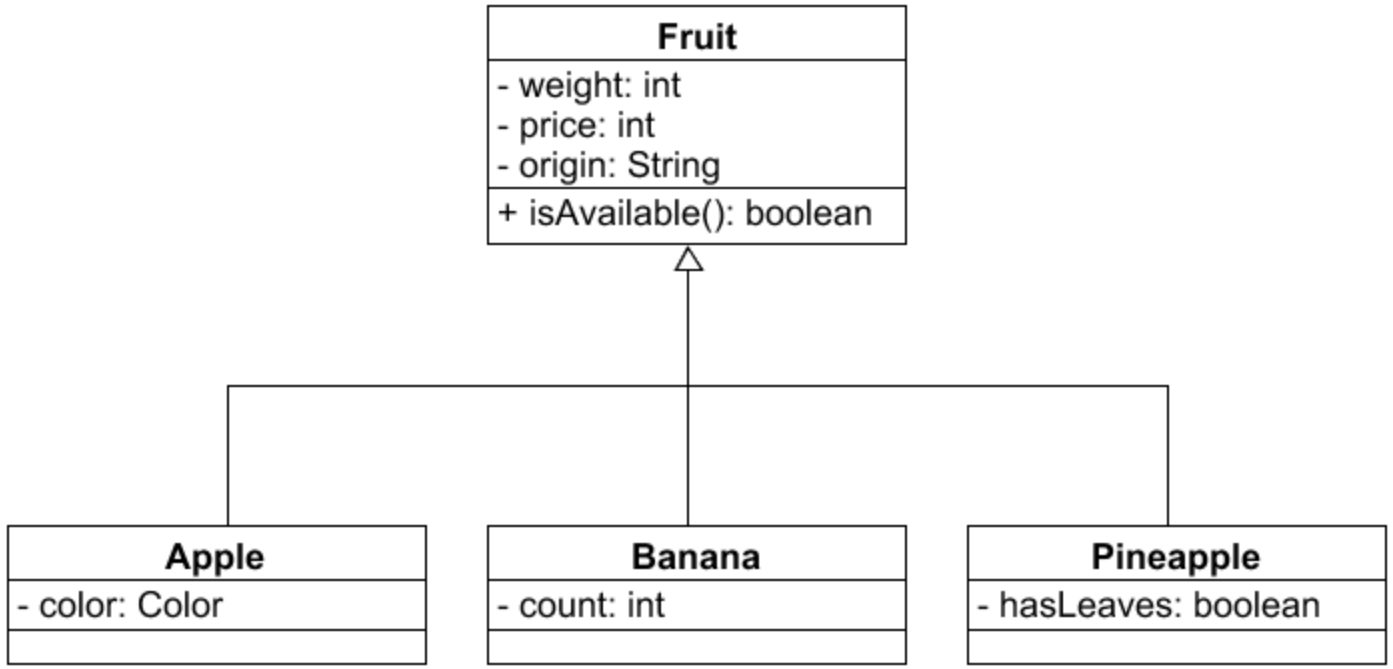
CARL VON OSSIETZKY universität OLDENBURG

Beispiel: Spezialisierung

```
public class List {

    public void add(Object data) { … }

    public boolean contains(Object data) { … }

    public void remove(Object data) { … }

}
```

```
public class Set extends List {

    public void add(Object data) {
        if (!contains(data)) {
            super.add(data);
        }
    }

}
```

Vererbung: Modell



CARL VON OSSIETZKY universität OLDENBURG

Vererbung: Implementierung

```
public class Fruit {

    private int weight;
    private int price;
    private String origin;

    public boolean isAvailable() {
        return true;
    }

    public int getWeight() { return weight; }
    public void setWeight(int weight) { this.weight = weight; }
    public int getPrice() { return price; }
    public void setPrice(int price) { this.price = price; }
    public String getOrigin() { return origin; }
    public void setOrigin(String origin) { this.origin = origin; }
}
```

CARL VON OSSIETZKY universität OLDENBURG

Vererbung: Implementierung (2)

```
public class Apple extends Fruit {

    private Color color;

    public Color getColor() {
        return color;
    }

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

}
```

CARL VON OSSIETZKY universität OLDENBURG

Vererbung: Implementierung (3)

```
public class Banana extends Fruit {

    private int count;

    public int getCount() {
        return count;
    }

    public void setCount(int count) {
        this.count = count;
    }

}
```

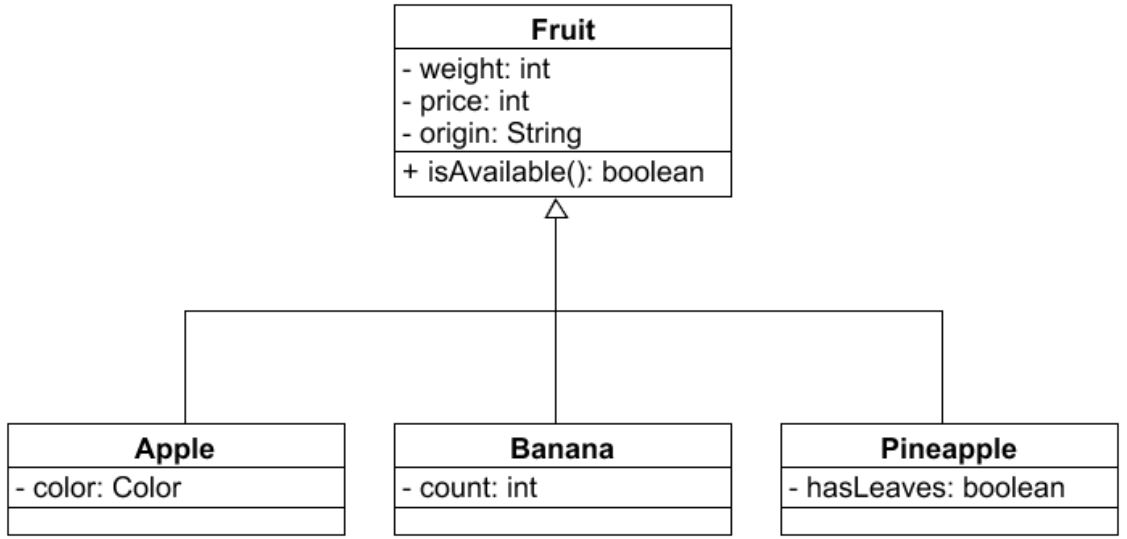
CARL VON OSSIETZKY universität OLDENBURG

Vererbung: Implementierung (4)

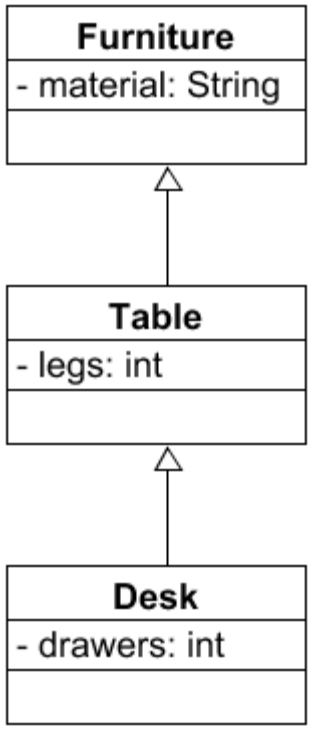
```
public class Pineapple extends Fruit {

    private boolean hasLeaves;

    public boolean isHasLeaves() {
        return hasLeaves;
    }

    public void setHasLeaves(boolean hasLeaves) {
        this.hasLeaves = hasLeaves;
    }

}
```

Vererbung: Implementierung (5)

```
public class FruitBasket {

    public static void main(String[] args) {
        Apple apple = new Apple();
        Banana banana = new Banana();
        Pineapple pineapple = new Pineapple();
        int basketPrice =
                apple.getPrice() +
                banana.getPrice() +
                pineapple.getPrice();
        System.out.println(basketPrice);
    }

}
```



Beispiel: Vererbung

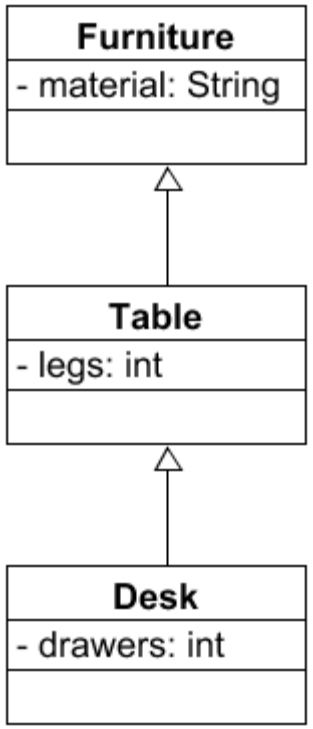



```
public class Furniture {

    private String material;

}
```

```
public class Table extends Furniture {

    private int legs;

}
```

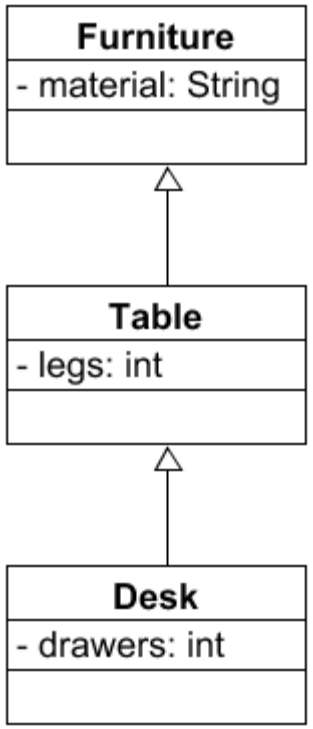
```
public class Desk extends Table {

    private int drawers;

}
```

Beispiel: Vererbung (2)




```
public class FurnitureTest {

    public static void main(String[] args) {
        Furniture furniture = new Furniture();
        furniture.setMaterial("Oak");
        furniture.setLegs(4);
        furniture.setDrawers(3);

        Table table = new Table();
        table.setMaterial("Oak");
        table.setLegs(4);
        table.setDrawers(3);

        Desk desk = new Desk();
        desk.setMaterial("Oak");
        desk.setLegs(4);
        desk.setDrawers(3);
    }

}
```

Beispiel: Vererbung (2)

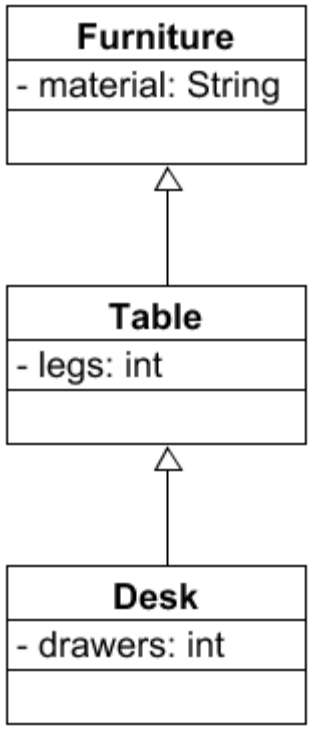



```
public class FurnitureTest {

    public static void main(String[] args) {
        Furniture furniture = new Furniture();
        furniture.setMaterial("Oak");
        furniture.setLegs(4);
```

The method **setLegs(int)** is undefined for the type **Furniture**

```
        Table table = new Table();
        table.setMaterial("Oak");
        table.setLegs(4);
        table.setDrawers(3);

        Desk desk = new Desk();
        desk.setMaterial("Oak");
        desk.setLegs(4);
        desk.setDrawers(3);
    }

}
```

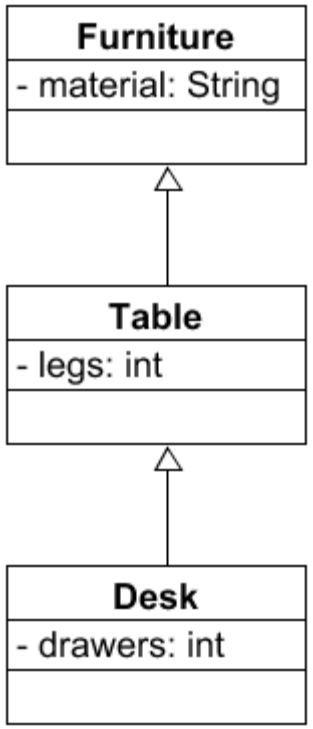
Beispiel: Vererbung (2)




```
public class FurnitureTest {

    public static void main(String[] args) {
        Furniture furniture = new Furniture();
        furniture.setMaterial("Oak");
        furniture.setLegs(4);
        furniture.setDrawers(3);
        Table table = new Table();
        table.setMaterial("Oak");
        table.setLegs(4);
        table.setDrawers(3);

        Desk desk = new Desk();
        desk.setMaterial("Oak");
        desk.setLegs(4);
        desk.setDrawers(3);
    }

}
```

The method **setDrawers(int)** is undefined for the type **Furniture**

Beispiel: Vererbung (2)

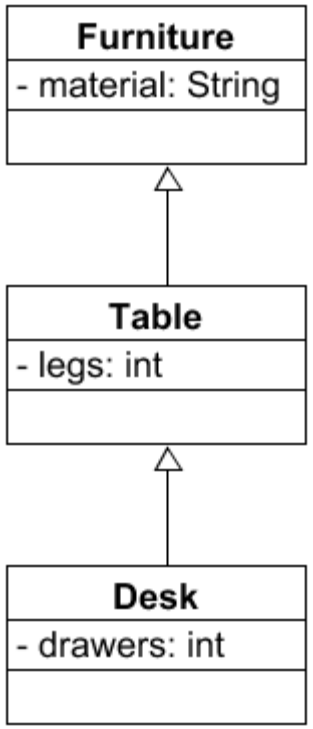



```
public class FurnitureTest {

    public static void main(String[] args) {
        Furniture furniture = new Furniture();
        furniture.setMaterial("Oak");
        furniture.setLegs(4);
        furniture.setDrawers(3);

        Table table = new Table();
        table.setMaterial("Oak");
        table.setLegs(4);
        table.setDrawers(3);
```

The method **setDrawers(int)** is undefined for the type **Table**

```
        Desk desk = new Desk();
        desk.setMaterial("Oak");
        desk.setLegs(4);
        desk.setDrawers(3);
    }

}
```

Beispiel: Vererbung (2)




```
public class FurnitureTest {

    public static void main(String[] args) {
        Furniture furniture = new Furniture();
        furniture.setMaterial("Oak");
        furniture.setLegs(4);
        furniture.setDrawers(3);

        Table table = new Table();
        table.setMaterial("Oak");
        table.setLegs(4);
        table.setDrawers(3);

        Desk desk = new Desk();
        desk.setMaterial("Oak");
        desk.setLegs(4);
        desk.setDrawers(3);
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

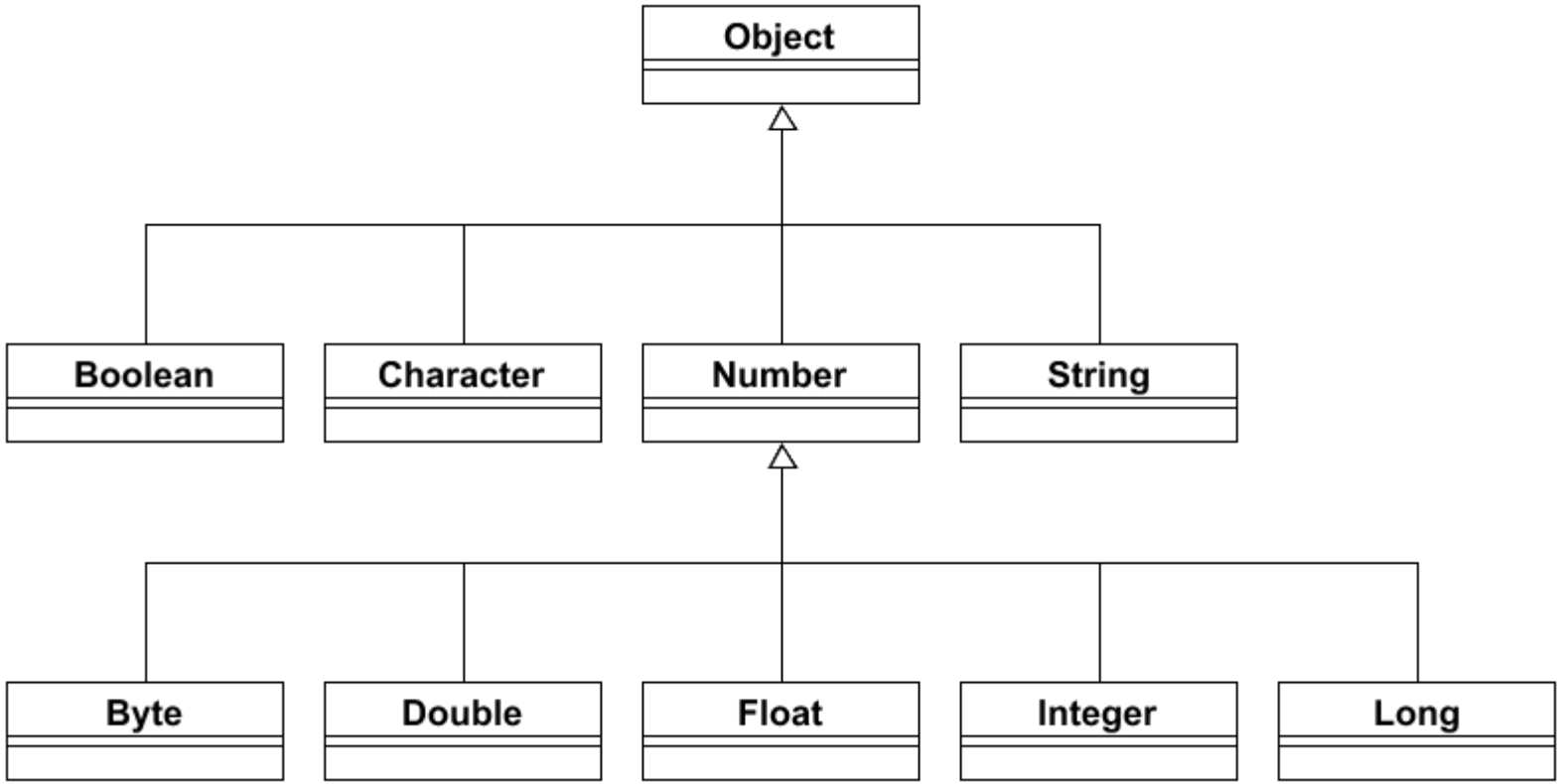
Vererbung

- Erben
  - die Erben (Unterklassen) einer Klasse (Oberklasse) sind Klassen, welche die Spezifikation und Implementierung der Oberklasse übernehmen und sie erweitern oder modifizieren (ohne Code-Duplizierung)
- Vererbung
  - neue Klassen (Unterklassen) können durch Erweiterung bzw. Modifikation bereits existierender Klassen (Oberklassen) definiert werden

Vererbung (2)

- Generalisierung
  - die Oberklasse ist eine Generalisierung der Unterklassen
- Spezialisierung
  - die Unterklassen sind Spezialisierungen der Oberklasse
- Vererbungsbeziehung
  - die Vererbung führt zur hierarchischen Anordnung von Klassen und kann sich über mehrere Stufen erstrecken
  - die Beziehung ist eine **`is-a`** Beziehung, d.h. **`<Unterklasse> is-a <Oberklasse>`**, z.B. **`Apple is-a Fruit`** (jeder Apfel ist eine Frucht)
  - die Umkehrung gilt in der Regel **nicht**, z.B. nicht jede Frucht ist ein Apfel

Java-Typhierarchie (Ausschnitt)



CARL VON OSSIETZKY universität OLDENBURG

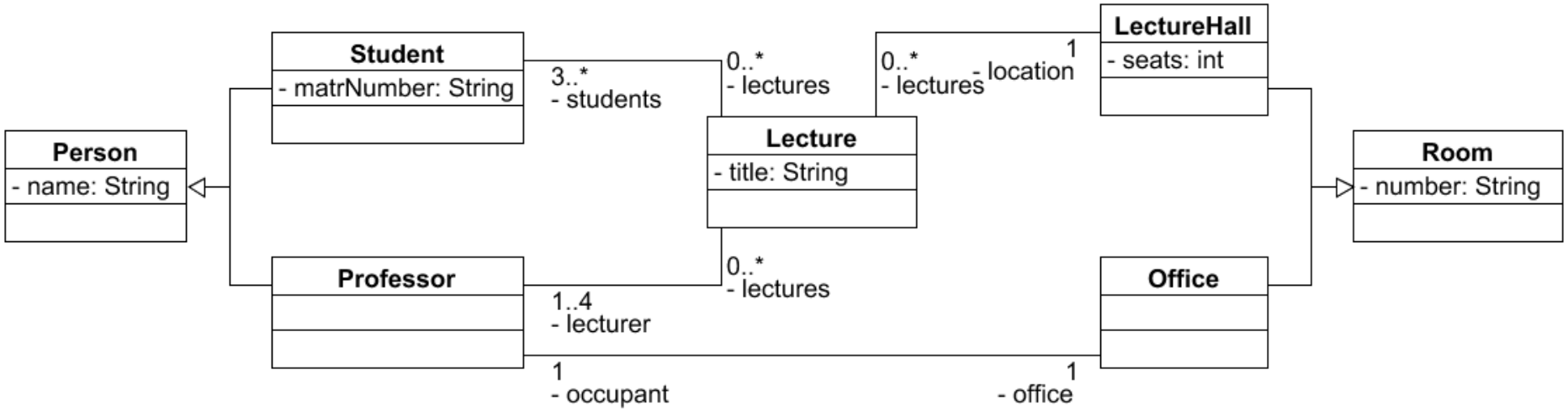
Object

- In Java sind alle Klassen direkte oder indirekte Unterklassen der Klasse **`Object`**
- Jede Klasse hat entweder eine explizite Oberklasse (**`extends Superclass`**) oder hat implizit **`Object`** als Oberklasse (**`extends Object`**)
- Beispiel

```
public class Apple extends Fruit
```

```
public class Fruit
```
==
```
public class Fruit extends Object
```

Beispiel: Vererbung (3)



CARL VON OSSIETZKY universität OLDENBURG

Konstruktoren

```
public class Fruit {

    private int weight;
    private int price;
    private String origin;

    public Fruit() {
        price = 99;
    }

}
```

```
public class Apple extends Fruit {

    private Color color;

    public Apple() {
        super();
        color = Color.RED;
    }

}
```

CARL VON OSSIETZKY universität OLDENBURG

Konstruktoren (2)

```
public class Fruit {

    private int weight;
    private int price;
    private String origin = "France";

    public Fruit(int price) {
        this.price = price;
    }

}
```

```
public class Apple extends Fruit {

    private Color color;

    public Apple() {
        super(99);
        color = Color.RED;
    }

}
```

Konstruktoren (3)

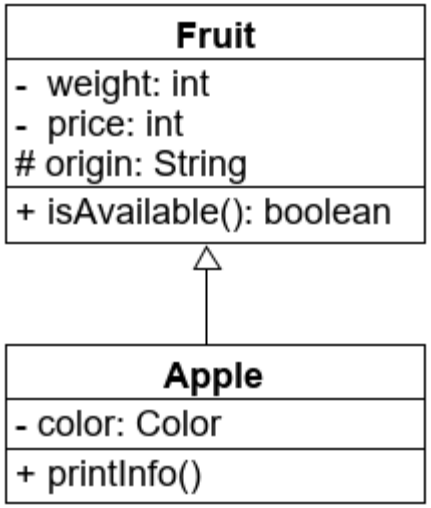
```
public class Fruit {

    // ...

    public Fruit() {
        this(99);
    }

    public Fruit(int price) {
        this.price = price;
    }

}
```

```
public class Apple extends Fruit {

    private Color color;

    public Apple() {
        super();
        color = Color.RED;
    }

}
```

Sichtbarkeit

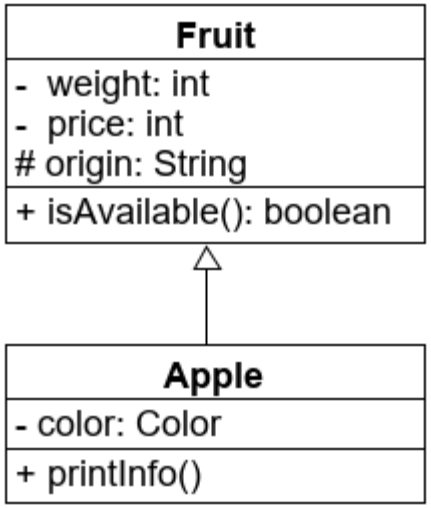
Beschreibung	Java	UML
für alle sichtbar:	`public`	`+`
nur für die aktuelle Klasse sichtbar:	`private`	`-`
für die aktuelle Klasse und alle Unterklassen sichtbar:	`protected`	`#`

Sichtbarkeit: Beispiel




```
public void printInfo() {
    if (color.equals(Color.RED)) {
        System.out.print("Red ");
    } else if (color.equals(Color.GREEN)) {
        System.out.print("Green ");
    } else {
        System.out.print("Misc. ");
    }
    System.out.print("Apple from ");
    System.out.print(origin);
    System.out.print(" that weighs ");
    System.out.print(weight);
    System.out.println(" grammes.");
}
```

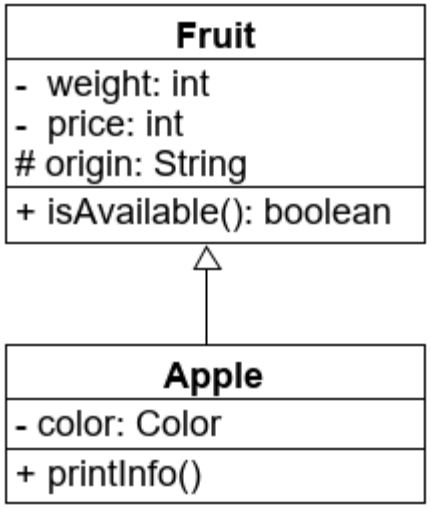
Sichtbarkeit: Beispiel




```
public void printInfo() {
    if (color.equals(Color.RED)) {
        System.out.print("Red ");
    } else if (color.equals(Color.GREEN)) {
        System.out.print("Green ");
    } else {
        System.out.print("Misc. ");
    }
    System.out.print("Apple from ");
    System.out.print(origin);
    System.out.print(" that weighs ");
    System.out.print(weight);
}
```

The field **Fruit.weight** is not visible

Sichtbarkeit: Beispiel



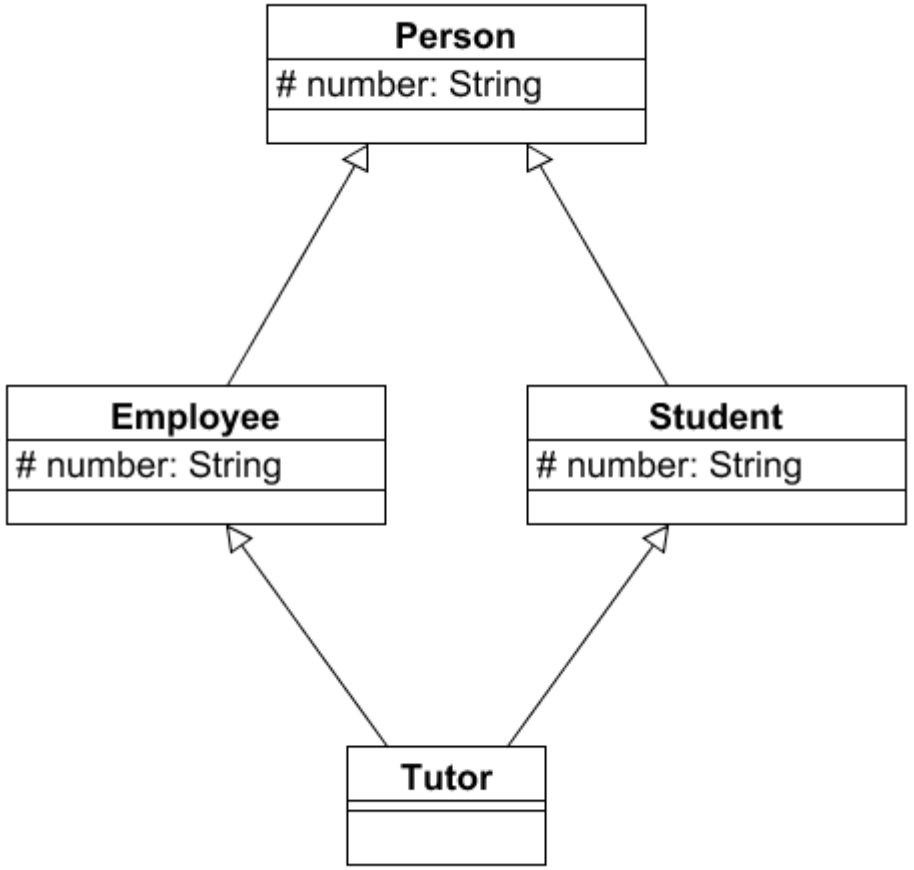

```
public void printInfo() {
    if (color.equals(Color.RED)) {
        System.out.print("Red ");
    } else if (color.equals(Color.GREEN)) {
        System.out.print("Green ");
    } else {
        System.out.print("Misc. ");
    }
    System.out.print("Apple from ");
    System.out.print(origin);✅
    System.out.print(" that weighs ");
    System.out.print(weight);
    System.out.print(getWeight());
    System.out.println(" grammes.");
}
```

Mehrfachvererbung

- Prinzipiell könnte eine Unterklasse mehrere Oberklassen haben
  → Mehrfachvererbung
- Die UML unterstützt Mehrfachvererbung
- Java unterstützt Mehrfachvererbung **nicht**

Mehrfachvererbung: Diamant-Problem

- **`Person`** definiert eine Nummer (z.B. Ausweis-Nummer)
- **`Employee`** und **`Student`** definieren ihre eigenen Nummern (z.B. Personalnummer und Matrikelnummer)
- Was erbt **`Tutor`**?



CARL VON OSSIETZKY universität OLDENBURG

Lernziele

- Vererbung
  - in Java
  - in der UML
- `this()`, `super()`
- `Object`
- Sichtbarkeit
- Mehrfachvererbung