PR2 – Formular für Lesenotizen

SS2021

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	11.04.21

Vererbung – einmal schreiben und Wiederverwenden **nur bei „Ist-ein"-Beziehung!**
Superklasse (Oberklasse, Basisklasse): die Klasse, die erweitert wird.
Subklasse (Unterklasse, abgeleitete Klasse): die Klasse, die die Superklasse erweitert und deren Verhalten erbt.
Syntax der Vererbung

```
public class name extends superclass {
```

Bsp. `public class Secretary extends Employee {...}`

Methoden überschreiben
Überschreiben (override): neu implementieren!
Definition einer Methode gleicher Signatur in einer Subklasse, die die Implementierung der Superklasse ersetzt.

Überladen (overload):
Definition mehrerer Methoden gleichen Namens mit unterschiedlicher Signatur in derselben Klasse.

L.3.1.2 Die @Override-Annotation

```
@Override public String getVacationForm() { return "blue"; }
```

Wenn es keine Superklasse zu der gibt, entsteht ein Compilerfehler

L.3.2 Vererbungsebenen

Es ist möglich, von einer Klasse zu erben, die ihrerseits von einer weiteren Klasse erbt.

L.3.3 Vererbung und Konstruktoren

- **Konstruktoren werden nicht vererbt**
- Wenn eine Klasse **keinen Konstruktor** hat, stellt Java automatisch einen Standard- Konstruktor ohne Parameter bereit, der alle Attribute auf 0 initialisiert.
- Jeder Subklassen-Konstruktor muss einen Superklassen-Konstruktor aufrufen: muss das 1. Statement sein in der Method in der die Parameter eingegeben werden können

allgemeine Syntax:

```
super(<parameters>);
```

als erstes Statement im Subklassenkonstruktor

```
public class Lawyer extends Employee {
 public Lawyer() {
     super();  // calls Employee() constructor
} ... } // super(years);
```

L.3.3.3 Aufruf des Superklassen-Konstruktors

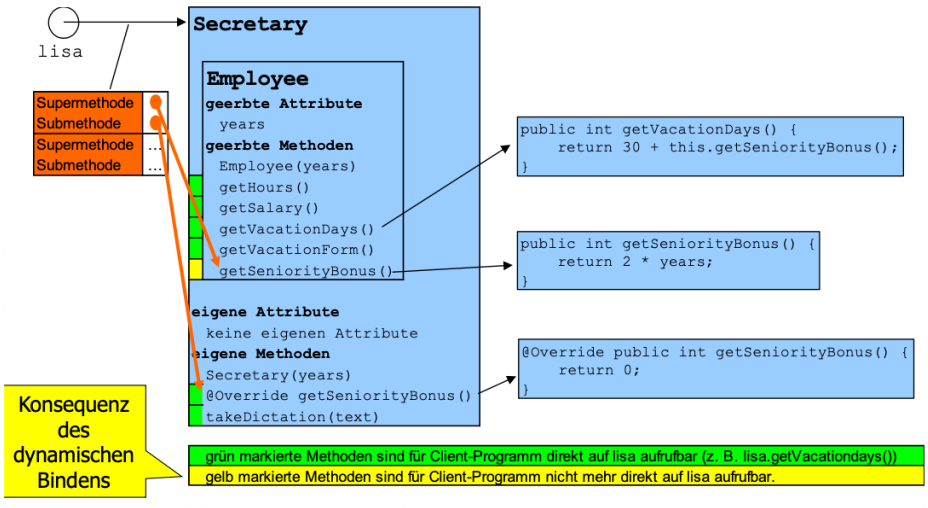
Jeder Subklassen-Konstruktor muss einen Superklassen-Konstruktor aufrufen:

- entweder (wie oben) implizit den Standardkonstruktor der Superklasse (Aufruf von `super()`-)
- oder durch einen expliziten `super(years)`-Aufruf.

L.3.4 Vererbung und Attribute

- Attribute sind immer private

L.3.5 Dynamisches Binden



• Überschreibende Subklassenmethode ist auch aus Superklasse aufrufbar:

– Ein Aufruf this.methode() wird zur Laufzeit dynamisch gebunden

– Dabei wird zur Laufzeit die orange dargestellte Tabelle ausgewertet. Eine solche Tabelle kann man sich an jeder Objektreferenz angehängt vorstellen.

– Wenn methode überschrieben ist, wird die Subklassen-Methode ausgeführt: this.getSeniorityBonus() führt die Methode der Subklasse Secretary aus.

– ... und zwar auch dann, wenn this vom Typ Employee ist

Zugriff auf private Attribute der Superklasse

`protected int getGesamtAnzahl() {` – Protected sind in der Subklasse aufrufbar

protected ist schlechter, da eine direkte Verwendung des Attributs zu einer engen Kopplung zwischen Super- und Subklasse führt. getter ist aber besser

Zugriffsmodifizierer

zugreifbar für Code ...	Zugriffsmodifizierer vor Attribut/Methode in der Klasse C			
	public	**protected**	**(Keiner)**	**private**
... in der Klasse C	Ja	Ja	Ja	Ja
... in einer anderen Klasse desselben Package	Ja	Ja	Ja	Nein
... einer Subklasse von C im selben Package wie C	Ja	Ja	Ja	Nein
... einer Subklasse von C, die in einem anderen Package steht	Ja	Ja	Nein	Nein
... einer beliebigen Klasse eines anderen Package	Ja	Nein	Nein	Nein

Unabhängig vom Zugriffsrecht müssen geeignete import-Statements angegeben werden, wenn aus anderen Packages zugegriffen werden soll.

Zugriffskontrolle für ganze Klassen

Nur zwei Möglichkeiten: public oder package – (als innere Klasse auch protected/private möglich!)

`public class Person{...}` – Die Klasse ist außerhalb des Pakets bekannt

– Es darf nur eine public-Klasse innerhalb einer Datei geben

`class Person{...}`

– Die Klasse ist nur den Klassen innerhalb des Pakets bekannt, in dem auch Person steht.

Aufruf von Methoden aus dem Konstruktor

- Regel: Rufe aus dem Konstruktor keine **zustandsabhängigen** Methoden auf, da es zu diesem Zeitpunkt noch keinen vollständigen Zustand des Objekts gibt.
- Wähle entweder **statische*** Methoden oder zustandsunabhängige **private** oder **finale*** Instanzmethoden.

Das äußere Objekt Rechnung erschafft das innere Objekt Rechnungsposition und liefert keine Referenz des inneren Objekts nach außen

```
public class Rechnung {
    private int nummer;
    private ArrayList<Rechnungsposition> positionen;

    public Rechnung (int nummer) {
        this.nummer = nummer;
        positionen = new ArrayList<Rechnungsposition>();
    }
    public void addPos(int artikelnummer, double preis) {
        positionen.add(new Rechnungsposition(artikelnummer, preis));
    }
    public int getArtikelnummer(int pos) {
        return positionen.get(pos).getArtikelnummer();
    }
    public double getPreis(int pos) {
        return positionen.get(pos).getPreis();
    }
}
```

```
public class Rechnungsposition {
    private int artikelnummer;
    private double preis;

    Rechnungsposition(int artikelnummer,
double preis) {
        this.artikelnummer = artikelnummer;
        this.preis = preis;
    }
    int getArtikelnummer() {
        return artikelnummer;
    }
    double getPreis() {
        return preis;
    }
}

Komposition !!!!!!!!!!!!!!!!!!!
```

```
public class Employee {
    private int years;
    public Employee(int initialYears) {
        years= initialYears;
    }
    public int getHours() {
        return 40;
    }
    public double getSalary() {
        return 40000.0;
    }
    public int getSeniorityBonus() {
        return 2*years;
    }
    public int getVacationDays() {
        return 30 + getSeniorityBonus();
    }
    public String getVacationForm() {
        return "yellow";
    }
}
```

```
public class Lawyer extends Employee {
    public Lawyer(int years) {
        super(years);
    }
    @Override public String getVacationForm() {
        return "blue";
    }
    @Override public int getVacationDays() {
        return 25 + getSeniorityBonus();
    }
    @Override public double getSalary() {
        return super.getSalary()+ 5000.0;
    }
    @Override public int getVacationDays() {
        return super.getVacationDays()-5;
    }

}
```