University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatik I – Eprog HS12

Übung 11

1 Aufgabe: Interfaces & Visitor-Pattern

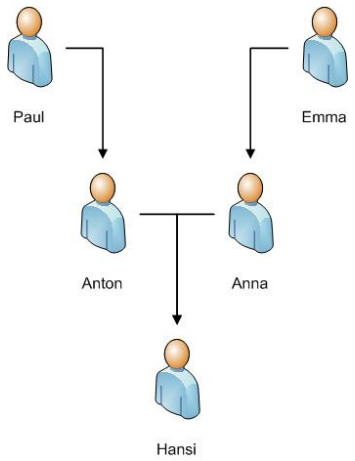
1.1 Lernziele

1. Die Verwendung von Interfaces trainieren.
2. Das Visitor Design-Pattern kennenlernen.

1.2 Aufgabenstellung

Die Instanzen der Klasse `Person` (siehe nächste Seite) können dazu verwendet werden, um Familien-Stammbäume in Java abzubilden. Zudem hat der Programmierer der Klasse mit der `accept`-Methode eine elegante Möglichkeit geschaffen, um nachträglich neue Funktionalität hinzuzufügen, ohne bestehenden Code anpassen zu müssen.

Die nachfolgende Illustration zeigt exemplarisch einen Stammbaum einer Familie:



Neue Funktionalität kann hinzugefügt werden, indem man Klassen schreibt, die das Interface `IVisitor` implementieren. Instanzen dieser Klassen können dann der `accept`-Methode von `Person`-Objekten übergeben werden.

```java
public class Person {
   private int age;
   private String name;

   private Person mom;
   private Person dad;

   public Person(String name, Person mom, Person dad) {
      this.name = name;
      this.mom = mom;
      this.dad = dad;
   }

   public int getAge() { return age; }

   public void setAge(int age) { this.age = age; }

   public String getName() { return name; }

   public void accept(IVisitor visitor) {
      visitor.visit(this);
      if(mom != null) {
         mom.accept(visitor);
      }
      if(dad != null) {
         dad.accept(visitor);
      }
   }
}
```

```java
public interface IVisitor {
   public void visit(Person p);
}
```

Wichtig: Für die nachfolgenden beiden Aufgaben dürfen Sie weder die Klasse `Person`, noch das Interface `IVisitor` verändern!

1. Schreiben Sie eine Klasse `NameCollectingVisitor`, welche das `IVisitor`-Interface implementiert und es erlaubt, sämtliche Namen aller Personen in einem Stammbaum zurückzuliefern. Im Falle des obenstehenden Stammbaumes soll es also möglich sein, eine Instanz von `NameCollectingVisitor` der `accept`-Methode der `Person`-Instanz mit dem Namen *Hansi* zu übergeben und anschliessend über die Methode `getCollectedNames` einen `String` zu erhalten, der sämtliche Namen aller fünf Personen enthält, also z.B. `"Hansi Anna Emma Anton Paul"`.
2. Schreiben Sie nun eine Klasse `AverageAgeVisitor`, analog zu jener aus der vorangehenden Aufgabe, die das Durchschnittsalter aller Personen im Stammbaum beim Aufruf der `getAverageAge`-Methode zurückliefert.

2 Aufgabe: Statischer und dynamischer Typ

2.1 Lernziele

1. Repetition statischer und dynamischer Typ

2.2 Aufgabenstellung

Gegeben seien folgende Klassen:

```java
public class Super {
   public void printHello() {
      System.out.println("Hallo Welt!");
   }

   public void printSomething() {
      System.out.println("Etwas.");
   }
}
```

```java
public class SubEnglish extends Super {

   public void printHello() {
      System.out.println("Hello World!");
   }
}
```

Kreuzen Sie jeweils an, welches der folgenden Statements in einem TestDriver in der main-Methode zulässig ist und schreiben Sie allfälligen Konsolen-Output des Statements daneben.

```java
Super x = new Super();
x.printHello();
```

Zulässig □ Unzulässig Output:

```java
Super y = new SubEnglish();
y.printSomething();
```

□ Zulässig □ Unzulässig Output:

```java
SubEnglish z = new Super();
z.printHello();
```

□ Zulässig □ Unzulässig Output:

3 Aufgabe: Rekursion

3.1 Lernziele

1. Formale Notation verstehen und erklären können.
2. Rekursive Methode in Java implementieren können.

3.2 Aufgabenstellung

1. Gegeben seien folgende Rekursionsformeln:

$$fak_x \begin{cases} x \times fak_{x-1} & , \text{ falls } x > 1 \\ 1 & , \text{ falls } x = 1 \text{ oder } x = 0 \\ \text{nicht definiert} & , \text{ sonst} \end{cases} \tag{1}$$

$$fib_i \begin{cases} fib_{i-1} + fib_{i-2} & , \text{ falls } i > 1 \\ 1 & , \text{ falls } i = 1 \\ 0 & , \text{ falls } i = 0 \\ \text{nicht definiert} & , \text{ sonst} \end{cases} \tag{2}$$

2. Was berechnen die obenstehenden Formeln?
3. Implementieren Sie die obenstehenden Formeln mittels einer rekursiven Methode. Für undefinierte Werte soll auf der Konsole eine Fehlermeldung ausgegeben werden.

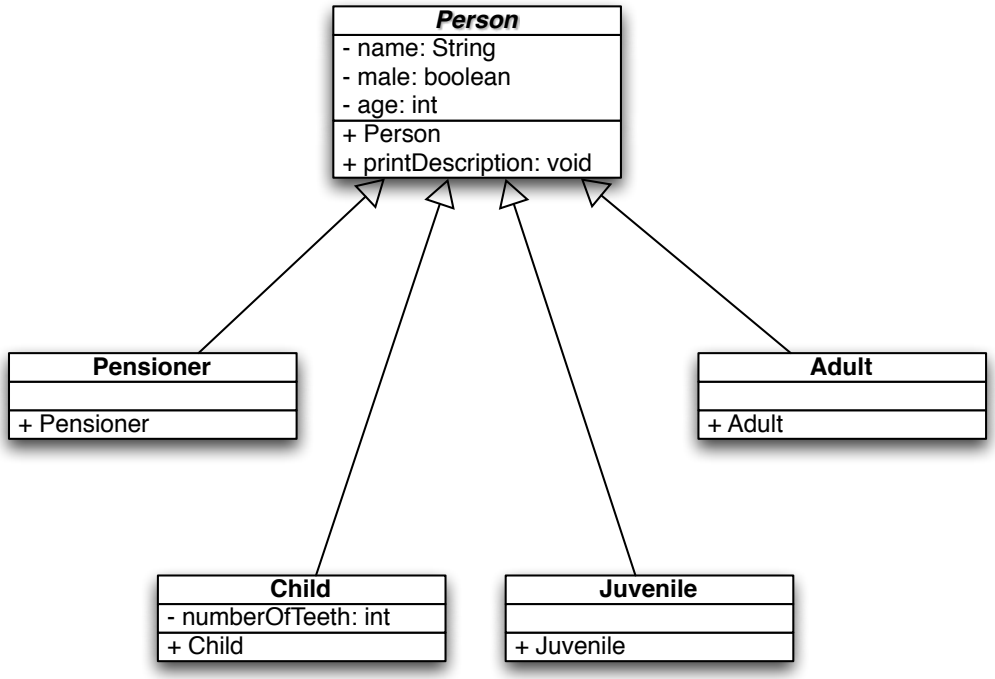
4 Aufgabe: Vererbung

4.1 Lernziele

1. Sie können ein UML-Diagramm in Code umsetzen.
2. Sie verstehen das Prinzip von Vererbung und Polymorphismus.

4.2 Aufgabenstellung

Gegeben sei folgendes UML-Klassendiagramm:



a) UML Notation

Hinweise zur Notation im Klassendiagramm:

- Kursive Notation kennzeichnet abstrakte Klassen/Methoden (siehe `Person`).
- Pfeile mit weisser Spitze zeigen die Vererbungshierarchie auf. `Child`, `Juvenile`, `Adult` und `Pensioner` sind also Subklassen von `Person`.
- Im obersten Teilkästchen wird jeweils der Klassenname notiert. Anschliessend folgen die Attribute und zum Schluss die Methoden.
- Attribute werden gemäss der folgenden Notation illustriert:

  ```
  <+/-> <name> :  <type>
  ```

  wobei ein + für den Visibilitymodifier `public` und ein – für `private` steht.
- Methoden werden wie folgt notiert:

  ```
  <+/-> <name> :  <return-type>
  ```

  wobei + und – wieder für `public`, bzw. `private` stehen.

b) Implementierung

1. Implementieren Sie die Klassen `Person`, `Child`, `Juvenile`, `Adult`, und `Pensioner` gemäss den Vorgaben im Klassendiagramm. Fügen Sie wo nötig nicht-abstrakte accessor-Methoden in der Klasse `Person` hinzu. Die Methode `printDescription()` soll eine kurze Beschreibung ausgeben:

   Ich heisse Hans Muster, bin männlich, 70 Jahre alt und Pensionär(in).

2. Überschreiben Sie in der Klasse `Pensioner` die `setAge(int age)`-Methode so, dass beim Versuch ein Alter `<61` zu setzen eine Meldung auf die Konsole geschrieben wird.
3. Erstellen Sie in der `main()`-Methode eines TestDrivers verschiedene Objekte der abgeleiteten Klassen und legen Sie diese in einem Array ab. Führen Sie anschliessend für alle darin gespeicherten Objekte in einer geeigneten Schleife die Methode `printDescription()` aus.
4. Fügen Sie der Klasse `Child` zusätzlich eine Instanzmethode hinzu, deren Signatur in keiner der anderen Klassen vorkommen soll (beispielsweise eine Methode `play()`). Benutzen Sie erneut die Objekte welche in der Testklasse aus 3 erzeugt wurden, um die neu hinzugefügten Methoden aufzurufen.
   - Notieren Sie die entstehende(n) Fehlermeldunge(n)
   - Wie erklären Sie sich diese? - Schreiben Sie Ihre Überlegungen nieder.