University of Zurich[UZH]

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

Informatik I – Eprog HS13

Übung 7

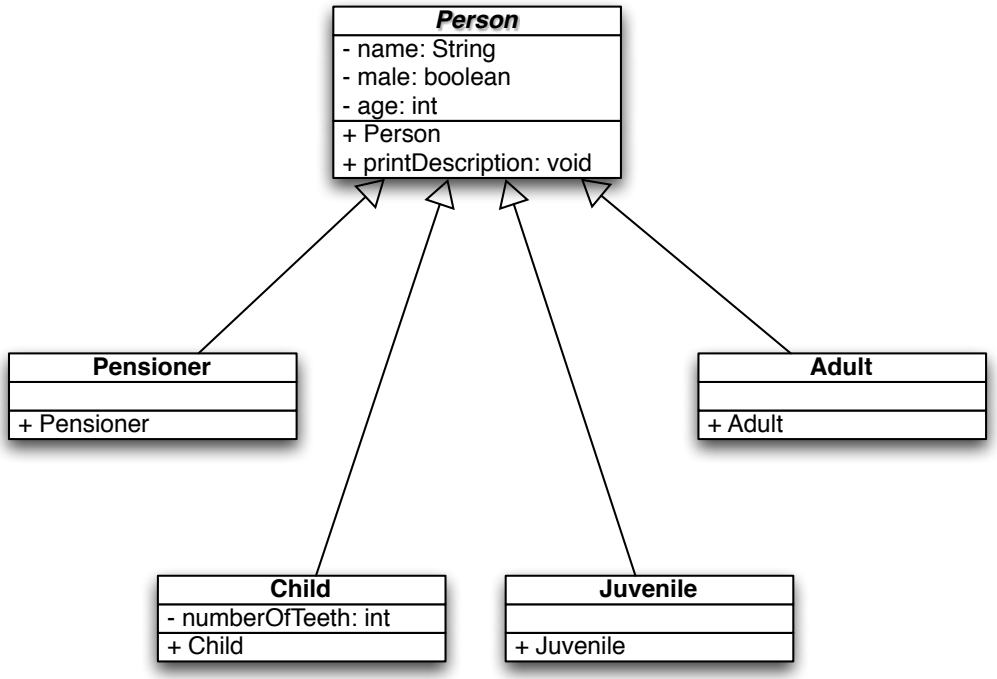
1 Aufgabe: Vererbung

1.1 Lernziele

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

1.2 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.

2 Aufgabe: OOP

2.1 Lernziele

1. Vertiefung der programmatischen Abbildung einer Problemstellung.
2. Vererbungsstrukturen erkennen und implementieren.

2.2 Ausgangslage

- Paul ist ein 13-jähriger Junge. Natürlich kann er Lesen und Schreiben. Am liebsten isst er Pizza.
- Miezi ist ein junges Kätzchen. Sie gehört Oma Hugentobler und frisst am liebsten Katzi-Katzenfutter. Ihr Fell ist getigert. Wenn sie einen Laut von sich gibt, dann miaut sie. Gerne spielt sie auch mit Oma Hugentoblers Wollknäuel.
- Lupo ist bereits 4 Jahre alt. Lupo, der Hund, gehört zu Paul. Am liebsten gräbt er im Garten nach Knochen. Wurst frisst er für sein Leben gern. Gibt er einen Laut von sich, dann bellt er. Sein Fell ist braun.
- Oma Hugentobler ist 81 Jahre alt. Paul ist ihr Neffe. Lesen und Schreiben kann sie schon seit langem! Ein gutes Stück Apfelkuchen kann sie nie ablehnen.

2.3 Aufgabenstellung

Gegeben sind vier Objekte. Identifizieren Sie daraus Klassen, Attribute und Methoden. Bilden Sie aus den zu den Objekten gehörenden Klassen eine Vererbungshierarchie, indem Sie Gemeinsames und Verwandtes in eine gemeinsame «Elternklasse» verschieben. Machen Sie sich dazu zunächst einige Notizen oder zeichnen Sie ein UML-Diagramm. Implementieren Sie dann diese Klassen in Java.

3 Aufgabe: Eine Adressbuch-Applikation - Teil 1

3.1 Aufgabenstellung

Im Rahmen dieser Aufgabe sollen Sie ein klassisches Adressbuch programmieren, welches die üblichen Funktionen, wie Hinzufügen von neuen Kontakten und das Löschen, ebenso wie das Suchen nach existierenden Kontakten ermöglicht. In den verbleibenden Wochen werden wir die Applikation sukzessive erweitern, sodass Ihre Applikation schliesslich eine graphische Benutzeroberfläche besitzen wird und sogar in der Lage ist, hinzugefügte Kontakte dauerhaft zu speichern.

In diesem ersten Schritt geht es vorerst einmal um die Implementation der Grundbausteine des Addressbuchs.

a) Kontakte

1. Bilden Sie den nachfolgenden Sachverhalt auf Klassen, Attribute und Methoden ab: Ein Kontakt hat einen Vor- und einen Nachnamen, sowie ein Geburtsdatum. Zudem eine Telefonnummer, eine Email- und eine Wohnadresse. Die Wohnadresse besteht aus Strasse/Hausnummer, Postleitzahl und aus einem Wohnort. Vor- und Nachnamen müssen bereits bekannt sein, wenn ein neuer Kontakt angelegt wird. Alle anderen Informationen (Telefon, Email, etc.) sind fakultativ und können auch nachträglich zum Kontakt hinzugefügt werden. Wenn eine Adresse hinzugefügt oder verändert wird, dann geschieht dies stets indem sowohl Strasse, Postleitzahl, wie auch Wohnort angegeben werden. Es ist nicht möglich, z.B. nur die Postleitzahl hinzuzufügen, bzw. zu verändern – alle Adresskomponenten müssen bekannt sein.
2. Das Geburtsdatum soll intern in einem `java.util.Date` Objekt verwaltet werden. Dieses soll aber in der öffentlichen Schnittstelle eines Kontakts nicht auftauchen. Stattdessen sollen Sie eine Methode `void setBirthday(int day, int month, int year)` bereitstellen, die das Geburtsdatum festlegt.
3. Zur internen Verwaltung soll jeder Kontakt eine eigene, eindeutige Nummer besitzen. Diese Nummer ändert sich – einmal zugewiesen – nicht mehr[1].
4. Stellen Sie für den oben beschriebenen Kontakt eine `toString()` Methode bereit. Die Methode soll insofern flexibel sein, als dass stets eine sinnvolle Beschreibung des Kontaktes zurückgeliefert werden soll – auch wenn nicht alle Felder gesetzt sind (z.B. kein Geburtsdatum bekannt ist). Verwenden Sie die Klasse `java.lang.StringBuilder` um den Rückgabewert zusammenzusetzen. `StringBuilder` erlaubt es Ihnen, im Gegensatz zur Klasse `String`, Zeichenketten nachträglich zu verändern. Dies ist zwar für diese Aufgabe nicht weiter relevant, es geht jedoch darum, die Verwendung der Java API zu trainieren. Formatieren Sie ausserdem das Geburtsdatum vor der Ausgabe mit Hilfe der Klasse `java.text.SimpleDateFormat`.
5. Testen Sie die oben stehende Funktionalität rudimentär in einem Test Driver.

b) Adressbuch

1. Für das Adressbuch kann beim Start ein Name festgelegt werden (zum Beispiel "Michael's Adressbuch"), sowie die verfügbare Kapazität, also jene Anzahl Kontakte, die im Addressbuch gespeichert werden können soll. Zum Speichern von Kontakten sollen Sie einen `Array` verwenden.

[1] Tip: Die Methode System.nanoTime() gibt typischerweise für jeden Aufruf eine andere Zahl zurück. Sie können in der Java API nachlesen, weshalb dem so ist.

2. Dem Adressbuch sollen natürlich neue Kontakte hinzugefügt werden können. Dies soll einerseits geschehen können, indem ein bereits erzeugter Kontakt übergeben wird (also mittels einer Methode `void addContact(Contact contact)`). Ausserdem sollen Sie eine Methode `Contact addContact(String firstName, String lastName)` bereitstellen, welche selber einen neuen Kontakt anlegt, diesen speichert und dann eine Referenz auf selbigen zurückliefert. Auf diese Weise kann man bequem Methodenaufrufe verketten, z.B.:

```
1  // ohne Verkettung:
2  Contact contact = addressBook.addContact("Anton", "Muster");
3  contact.setBirthday(3, 10, 1980);
4
5  // mit Verkettung:
6  addressBook.addContact("Hans", "Muster").setBirthday(3, 10, 1980);
```

3. Kontakte sollen auch gelöscht werden können. Implementieren Sie zu diesem Zweck eine Methode `void deleteContact(Contact contact)`. Beim Löschen sollen Lücken im Kontakte-`Array` automatisch bereinigt werden. `java.util.Arrays` wird Ihnen auch hier nützliche Funktionalität anbieten.
4. Schreiben Sie einige `Contact findBy*(String value)` Methoden, mit deren Hilfe Sie nach Kontakten suchen können. Die Methode `Contact findByLastName(String lastName)` soll beispielsweise den erstbesten Kontakt zurückliefern, der den übergebenen Nachnamen besitzt.
5. Schreiben Sie einige `void sortBy*()` Methoden, mit deren Hilfe Sie das Addressbuch neu sortieren können. Die Methode `void sortByLastName()` soll beispielsweise alle Einträge des Addressbuchs nach Nachnamen sortieren.

Schreiben Sie mehrere TestDriver, um die implementierte Funktionalität zu testen!