
Aufgabe 2.1

a) Implementieren Sie eine generische Klasse `Paar<E, Z>` mit

- zwei privaten Instanzattributen `erstes` vom Typ `E` und `zweites` vom Typ `Z`
- einem Konstruktor, dem je ein Parameter von Typ `E` bzw `Z` übergeben wird
- zwei get-Methode `E erstes()` und `Z zweites()`
- zwei set-Methoden `void setErstes(E e)` und `void setZweites(Z z)`

b) Überschreiben Sie die Methoden `toString()` und `equals()`.
Hinweis:
Die Methode `equals()` erwartet einen Parameter vom Typ `Object`.
Die Methode soll jedoch Paare (mit beliebigen Typen für E bzw Z) auf Gleichheit der Komponenten prüfen. Nur wenn der aktuelle Übergabewert vom gleichen Typ wie das `this`-Objekt ist, kann das Argument vom Typ `Object` auf den Paar-Typ gecastet werden und erst dann kann anschließend die Prüfung stattfinden, ob sowohl erste als auch zweite Komponente übereinstimmen.

Verwenden Sie wieder die `getClass()`-Methode, wie schon in der ertsen Übung.

Denken Sie auch an die notwendigen Annotationen!

c) Testen Sie die Klasse, indem Sie mehrere Paar-Objekte unterschiedlicher Typen erzeugen, bearbeiten und vergleichen. (vgl Folie 2-10)

(Zwei weitere Aufgaben auf den folgenden Seiten)

Aufgabe 2.2
Implementieren Sie eine Unterklasse Kind von Person. (Verwenden Sie die Klasse Person aus Übung1.)

a) Objekte der Klasse Kind haben ein zusätzliches (privates) Attribut eltern .
Die Eltern sind **ein Paar von Personen** (und nicht etwa zwei Einzelpersonen)!

b) Durch eine Methode setEltern (Person v, Person m) kann für ein Kind-Objekt das Elternpaar gesetzt werden.
Entsprechend soll es eine get-Methode geben, die das Elternpaar eines Kindes liefert.

c) Die Klasse Kind soll außerdem über eine statische boolesche Methode geschwister (Kind a, Kind b) verfügen, die prüft, ob die beiden übergebenen Kind-Objekte das gleiche Elternpaar haben.

d) Schreiben Sie eine Testklasse (mit main-Methode), in der Sie Objekte erzeugen, die den unten dargestellten Teil der Simpson-Familie repräsentieren. Lassen Sie prüfen, ob Bart und Lisa (ja) bzw Herb und Abe (nein) Geschwister sind.

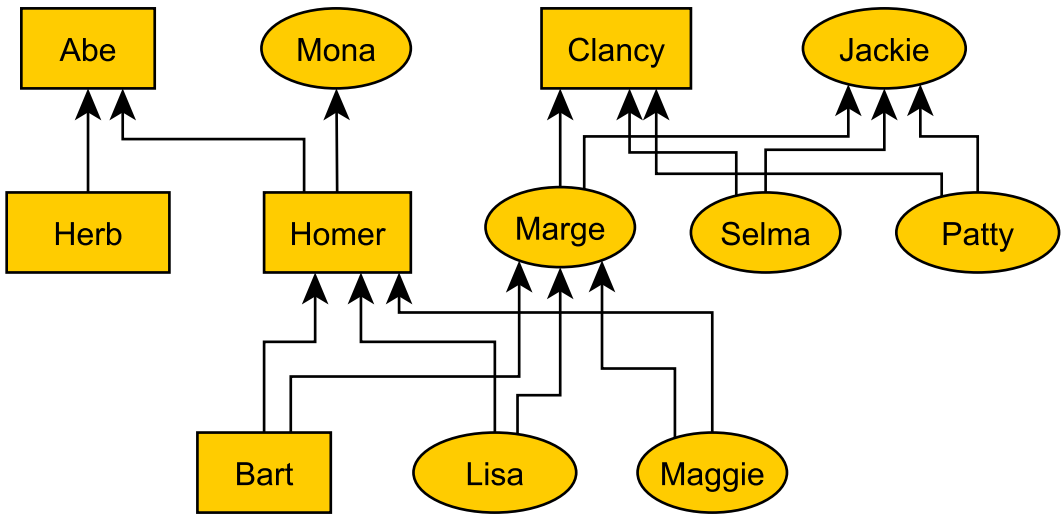
e) Implementieren Sie in der Testklasse eine statische Methode family (Kind k), die ein Paar bestehend aus dem Objekt k und einem Paar von Peronen (die Eltern von k) erzeugt und zurückliefert.

Für den Aufruf family (bart) sollte die Methode also etwas von der Struktur
(bart , (homer, marge)) liefern.

Schreiben Sie eine weitere statische Methode dAlter (...) , die eine „Familie“, also ein Paar bestehend aus einem Kind und seinen Eltern übergeben bekommt und das Durchschnittsalter der Eltern bei der Geburt des Kindes berechnet.

Beispiel: Wenn Bart's Eltern in Jahren den 1970 bzw 1972 geboren sind und Bart selber im Jahr 2000, dann sollte der Aufruf dAlter (family (bart)) den Wert 29 liefern.

Und hier ein Ausschnitt aus dem weit verzweigten Stammbaum der Simpsons (die Geburtsjahre dürfen Sie sich ausdenken):



Aufgabe 2.3
Betrachten Sie das generische Interface Menge:

```
public interface Menge<T> {

        int size();
        boolean isEmpty();
        T get();
        void insert(T e);
        void delete(T e);
        boolean contains(T e);
}
```

Die Semantik der Methoden sei wie folgt beschrieben:

- size () liefert die Anzahl der Elemente.
- isEmpty() prüft, ob die Menge leer ist.
- get () liefert (irgend)ein Element der Menge (falls diese nicht leer ist), *ohne* es zu entfernen!
- insert (T e) fügt das Element e der Menge hinzu; falls e bereits (vorher) Element der Menge war, wird die Menge dadurch nicht verändert. (Mengen enthalten keine Duplikate.)
- delete (T e) entfernt das Element e aus der Menge. Falls e nicht enthalten ist, hat die Methode keine Wirkung.
- contains (T e) prüft, ob e in der Menge enthalten ist.

a) (**Theorie**)
Schreiben Sie (zunächst auf Papier) eine Klasse MengeUtil mit einer (einzigen) statischen generischen Methode merge. Diese Methode soll zwei Objekte (a und b) vom Typ `Menge` als Eingabe erhalten und diese beiden Mengen verschmelzen: anschließend ist die Menge a die Vereinigung der beiden Mengen und b ist leer.

Implementieren Sie „gegen das Interface“, dh nutzen Sie nur Methoden, deren Existenz durch das Interface garantiert ist.

(Sie brauchen für diesen Teil der Aufgabe keine Implementierung der Schnittstelle zu kennen!)

b) Um Ihre merge-Methode auch in der Praxis zu testen, brauchen wir eine konkrete Klasse, die das Interface implementiert. Sie können dafür die Klasse MengeLimited nutzen (ohne sie zu verändern - eigentlich auch, ohne sie anzuschauen).
Sie können sich natürlich auch an einer eigenen Implementierung der Schnittstelle versuchen.

c) Schreiben Sie JUnit-Tests, um insbesondere zu prüfen ob in MengeLimited (bzw in Ihrer eigenen Implementierung von Menge) das Verhalten von insert und delete korrekt implementiert wurde: Das Hinzufügen von bereits vorhandenen Elementen bzw das Entfernen von nicht-vorhandenen Elementen darf keine Wirkung haben!

Testen Sie auch das Verschmelzen zweier Mengen. (Wenn Sie Ihre Mengen mal *sehen* wollen: auch in JUnit-Tests können Sie mit print-Anweisungen Bildschirmausgaben erzeugen.)

Bemerkung:
Die Klasse MengeLimited ist (noch) nicht ausreichend gegen Ausnahmen gesichert. Sie kann zum Beispiel nur Mengen mit maximal 100 Elementen modellieren. Beim Einfügen von Elementen bzw. beim Verschmelzen zweier Mengen kann es daher zu Laufzeitfehlern kommen.
Das soll im Moment aber nicht unser Problem sein. Wählen Sie Testbeispiele mit deutlich weniger Elementen.

Sie finden den Code des Interface Menge und der Klasse MengeLimited im LEA-Kurs.