Übungen zu Objektorientierte Programmierung 2

frieder.loch@ost.ch

Woche 13: Design Patterns

Aufgabe 1: Iterator

Aufgabe 1.1

In der Vorlesung haben Sie Arten von Iteratoren kennengelernt. In einem ersten Schritt sollen Sie drei Iteratoren implementieren. 1. Ein normalen Iterator, der einfach durch eine List durch iteriert. 2. Ein reverse Iterator, der rückwärts durch die Liste iteriert. 3. Ein Snapshot Sortier-Iterator, welcher über die Elemente einer Liste sortiert iteriert, ohne das die Liste geändert wird.

Aufgabe 1.2

Ihre Aufgabe ist es nun einen `ParallelIterator` zu implementieren, der gleichzeitig durch eine oder zwei Listen iteriert. Wenn nur durch eine Liste iteriert wird, muss es möglich sein, dass die eine Liste auf zwei verschiedene Arten (z.B. sortiert, rückwärts) durchiteriert wird.

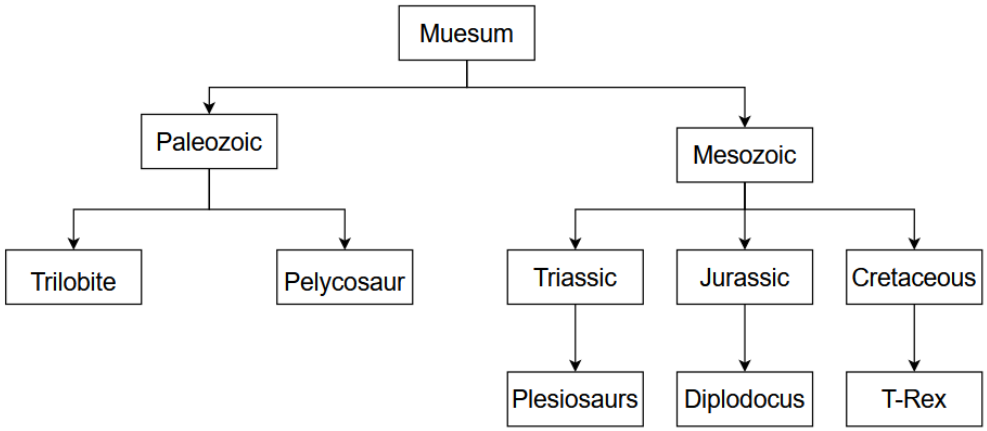
Wenn durch zwei Listen iteriert wird, dann soll nur solange durch die Listen iteriert werden, wie in beiden Listen Elemente vorhanden sind.

Es sollen immer die Elemente beider Iterationen zurückgegeben werden (zwei Iterationen auf einer Liste, eine Iteration auf zwei Listen). Dafür kann die Klasse `Pair` genutzt werden.

Aufgabe 2: Visitor Pattern

Ihre Aufgabe ist es, einen Museumsrundgang mit dem Visitor Pattern darzustellen. Ein Museum besteht aus verschiedenen Gebieten `Areas`, welche aus kleineren Gebieten oder Ausstellungsstücken `Exhibit` besteht. Führen Sie mit dem Visitor Pattern eine Museumstour durch.

Ein Dinosaurier-Museum kann zum Beispiel so aussehen:



Für dieses Museum soll die Tour die folgende Ausgabe haben:

```
Entering Dinosaur Museum
Visiting Paleozoic Area
Looking at Trilobite
Leaving Trilobite
Looking at Pelycosaur
Leaving Pelycosaur
Leaving Paleozoic Area
Visiting Mesozoic Area
Visiting Triassic Area
Looking at Plesiosaurs
Leaving Plesiosaurs
Leaving Triassic Area
Visiting Jurassic Area
Looking at Diplodocus
Leaving Diplodocus
Leaving Jurassic Area
Visiting Cretaceous Area
Looking at T-Rex
Leaving T-Rex
Leaving Cretaceous Area
Leaving Mesozoic Area
Leaving Dinosaur Museum
```

Erklärung Visitor Pattern

Reales Beispiel eines Museumbesuchs Stellen Sie sich vor, dass Sie ein Museum besuchen. Wie beschrieben besteht das Museum aus verschiedenen Gebieten, welche aus Untergebieten oder Ausstellungsstücken besteht. Ihre Tour wird zum Beispiel so ablaufen: 1. Sie betreten das Museum 2. Sie betreten einen Bereich 3. Sie schauen sich ein Ausstellungsstück an 4. Sie gehen von dem Ausstellungsstück weiter in einen Unterbereich oder zum nächsten Ausstellungsstück 5. Wenn Sie einen Bereich komplett durchgeschaut haben, dann verlassen Sie ihn wieder 6. Haben Sie alle Bereiche durchgeschaut, dann verlassen Sie das Museum

Visitor Pattern erstellen anhand des realen Beispiels Nun bilden wir die Tour in Klassen ab. Die Visitor Klasse, welche den Besucher abbildet, hat für jeden Teil des Museums `MuseumPart` eine `visit` und eine `leave`

Methode

```
public interface MuseumVisitor {
    void visit(Museum museum);
    void visit(Area area);
    void visit(Exhibit exhibit);
    void leave(Museum museum);
    void leave(Area area);
    void leave(Exhibit exhibit);
}
```

In der Klasse, welche dieses Interface implementiert beschreiben Sie zum Beispiel, was passiert, wenn man das Museum besucht und auch wieder verlässt.

Wenn wir den Ablauf nun zum oben beschriebenen Ablauf gleichstellen, dann kann die `visit`-Methode für ein Museum zum Beispiel so aussehen:

```
    @Override
    public void visit(Museum museum) {
        System.out.println("Entering " + museum.getName());
    }
```

Wie beschrieben, besteht ein Museum aus verschiedenen Teilen. Um diese Teile abzubilden, wird ein Interface erstellt:

```
public interface MuseumPart {
    void accept(MuseumVisitor visitor);
}
```

Das Interface `MuseumPart` schreibt nun eine Methode `accept` vor, welche einen `MuseumsVisitor` als Parameter entgegennimmt.

Das Museum, sowie alle Teile des Museums, implementiert dieses Interface.

```
public class Museum implements MuseumPart {

}
```

Das bedeutet natürlich auch, dass diese Methode auch in der Museumsklasse implementiert werden muss.