Dieses Übungsblatt dient der Vorbereitung auf die Klausur. Die Lösungen der Aufgaben sollen nicht abgegeben werden. Die Aufgaben werden in den Übungen der Woche vom 15.5.-19.5.2017 besprochen.

Übungsblatt 3

Aufgabe 1

Java-Datenstrukturen mit *fail-fast*-Iteratoren

Die Iteratoren von Java-Datenstrukturen wie beispielsweise `java.util.ArrayList<T>` setzen das *fail-fast*-Konzept um: Sobald das Listenobjekt geändert wird, also ein Element eingefügt oder gelöscht wird, verweigert jeder bereits für diese Liste erzeugte Iterator die Weiterarbeit und wirft eine `ConcurrentModificationException` bei seiner nächsten Nutzung. Recherchieren Sie, auf welche Weise der Iterator von der Änderung an der Liste erfährt.

Aufgabe 2

Entwurfsmuster – Iterator

Die Klasse `Graph` realisiert eine Datenstruktur, um gerichtete Graphen zu verwalten, dessen Knoten als natürliche Zahlen fortlaufend nummeriert sind. Die Informationen über einen Graph werden in einem zweidimensionalen Feld `matrix` abgelegt. Hier nicht bekannte Methoden sorgen für den korrekten Aufbau dieses Feldes.

Datenhaltung:
`matrix[i][j]==false` bedeutet, dass die beiden Knoten `i` und `j` nicht verbunden sind.
`matrix[i][j]==true` bedeutet, dass eine Kante vom Knoten `i` zum Knoten `j` führt.

Die Klasse `Graph` soll nun zusätzlich das Interface `Iterable<Edge>` implementieren. Skizzieren Sie die Erweiterungen, die an der Klasse `Graph` vorgenommen werden müssen.

Skizzieren Sie auch den Aufbau einer Klasse `GraphIterator`, die den benötigten Iterator bereitstellt. Der Iterator soll nacheinander alle Kanten des Graphen genau einmal liefern.

```
public class Graph {
     private boolean[][] matrix;
     ...
}
```

```
public class Edge {
     public Edge( int s, int e ){...}
     ...
}
```

```
public interface Iterator<T> {
     T next();
     boolean hasNext();
}
```

```
public interface Iterable<T> {
     Iterator<T> iterator();
}
```

Aufgabe 3

Entwurfsmuster Adapter

Implementieren Sie einen Klassenadapter, der auf der Basis der Klasse `Combination` das Interface `Stack` realisiert.

```
public interface Stack<E> {
     void push( E obj );        // adds an object obj to the stack
     E peek();                  // returns the object that has been added at last; returns
                                // null, if the stack is empty
     E pop();                   // returns and removes the object that has been added at last;
                                // returns null, if the stack is empty
     boolean isEmpty();         // return true, if the stack is empty
}
```

Die Klasse `Combination<E>` besitzt die folgenden öffentliche Konstruktoren und Methoden:

```
–  Combination()          // constructs an empty Combination-Object
–  int elements()         // returns the number of elements in this combination
–  E extract( int i )     // returns the element at the specified position in this
                          // combination; does nothing and returns null, if i is undefined
–  void delete( int i )   // removes the element at the specified position in this
                          // combination; does nothing if i is undefined
–  void extend( E o )     // appends the specified element to the end of this combination
```