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 4.6.-8.6.2018 besprochen.

Übungsblatt 7

Aufgabe 1
Aktivitätsdiagramm

Geben Sie ein Aktivitätsdiagramm an, dass die Kontrollstrukturen des folgenden Java-Programms visualisiert.

```
int sum = 0, factor = 10;
for ( int i = 0; i<1000; i++ ) {
    sum = sum + apply( i, factor );
    while ( checkLimit( sum ) ) {
        sum = request( database, sum );
        if ( sum > i*factor ) {
           factor--;
        }
    }
}
```

Aufgabe 2
Anwendungsfalldiagramm

Zeichnen Sie ein Anwendungsfalldiagramm, das das unten beschriebene Abrechnungssystem (ARS) modelliert:

> Die Sachbearbeiterinnen können Rechnungen erstellen, ansehen oder als bezahlt markieren. Das Markieren beinhaltet immer ein vorheriges Ansehen der Rechnung. Abteilungsleiterinnen können zusätzlich zu den Tätigkeiten der Sachbearbeiterinnen auch noch Rechnungen stornieren.
>
> Die Prüferinnen der Revisionsabteilung können Rechnungen ansehen oder prüfen. Ein Prüfen beinhaltet immer ein vorheriges Ansehen der Rechnung. Bezahlte Rechnungen dürfen nicht geprüft werden.

Aufgabe 3
Iterator

Die Klasse `MysteryStructure` realisiert eine Datenstruktur, von der nur vier Methoden bekannt sind. Die Klasse `MysteryStructure` soll nun zusätzlich das Interface `Iterable` implementieren. Skizzieren Sie die Erweiterungen, die an der Klasse vorgenommen werden müssen. Skizzieren Sie auch den Aufbau einer Klasse `MysteryIterator`, die den benötigten Iterator auf der Basis der bekannten Methoden von `MysteryStructure` bereitstellt. Der bereitgestellte Iterator soll *fail-fast* implementiert werden: Sobald das `MysteryStructure`-Objekt geändert wurde, wirft ein bereits existierender Iterator bei seiner nächsten Nutzung immer eine ConcurrentModificationException.

```
public class MysteryStructure<E>
{
     public MysteryStructure() {...}
     public int size() {...}              // gibt die Anzahl der abgelegten E-Objekte zurück
     public E getAt( int i ) {...}        // gibt das E-Objekt an der Position i zurück,
                                          // Zugriff erfolgt in annähernd konstanter Zeit
     public void clear() {...}            // löscht alle abgelegten E-Objekte
     public void add( E content ) {...}   // nimmt das Argument in die Datenstruktur auf
     ...
}
```

```
public Interface Iterator<V>
{
     V next();
     boolean hasNext();
}
```

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