Name:

Matrikelnummer:

Bearbeitungszeit: 120 min.

Aufgabe 1. Orte in einem Straßennetz sind durch ihre Nummern `0,...,anzOrte-1` gekennzeichnet. Alle Straßen des Straßennetzes sind Einbahnen, und das Straßennetz ist durch die gegebene Klasse `Strassennetz` mit den Methoden

```
public int getAnzOrte(),
public int[] getVerbundene(int einOrt),
```

gegeben. Die Methode `getAnzOrte()` liefert die Anzahl der Orte im Straßennetz zurück, und die Methode `getVerbundene(einOrt)` liefert eine Liste jener Ort, die vom Ort `einOrt` aus direkt mit einer (Einbahn-)Straße verbunden sind.

Schreiben Sie eine Klasse `Wege` mit einem parameterlosen Konstruktor und der Methode

```
public List<Integer> getErreichbare(Strassennetz netz, int einOrt),
```

die eine Liste aller Orte zurückgeben, die vom Ort `einOrt` aus erreichbar sind, evt. über mehrere andere Orte und mehrere Straßen.

Hinweis: Die Aufgabe kann sowohl rekursiv wie auch iterativ gut gelöst werden.

Aufgabe 2. Gegeben ist die Klasse `Pruefungsfrage` mit dem Konstruktor und den Methoden

```
public Pruefungsfrage(String text, String[] antworten),
public String getText(),
public String[] getAntworten().
```

Die get-Methoden liefern den im Konstruktor definierten String bzw. das String-Array zurück.

Schreiben Sie die Unterklasse `FrageMitVarianten` von `Pruefungsfrage` mit den Methoden

```
public static FrageMitVarianten create(Pruefungsfrage[] varianten),
public Pruefungsfrage getRandomVariante().
```

Die Methode `create(varianten)` soll eine neue `FrageMitVarianten` erzeugen und zurückgeben, wobei die Varianten durch die übergebenen Prüfungsfragen gegeben sind. Die Methoden `getText()` und `getAntworten()` sollen in `FrageMitVarianten` **nicht überschrieben werden!** Diese Methoden sollen den Text und die Antworten der Standardvariante (das ist die erste Frage in `varianten`) zurückgeben.

Die Methode `getRandomVariante()` soll bei jedem Aufruf eine zufällig gewählte Variante (inklusive der Standardvariante) zurückgeben.

Aufgabe 3. Die zeitliche Verzahnung von parallel laufenden Prozessen kann als Liste von Prozessschritten dargestellt werden, wobei in jedem Prozessschritt angegeben ist, welcher Prozess aktiv ist und welche Aktion dieser Prozess ausführt. Dabei werden hier nur Aktionen zur Synchronisation der Prozesse berücksichtigt. Die Synchronisation erfolgt durch das Sperren (acquire) bzw. Freigeben (release) von Objekten der gegebenen Klasse `SyncLock`. Ein Aufruf `einSyncLock.acquire()` wird erst ausgeführt, wenn `einSyncLock` frei ist; dann wird `einSyncLock` vom aufrufenden Prozess gesperrt. Beim Aufruf von `einSyncLock.release()` wird das zuvor gesperrte `einSyncLock` wieder freigegeben.

Die Prozessschritte werden von der gegebenen Klasse `Prozessschritt` mit den Methoden

```
public int getProzess(),
public String getAktion(),
public SyncLock getLock(),
```

dargestellt, wobei `getProcess()` die Prozessnummer das aktiven Prozesses liefert, `getAktion()` entweder `"ACQ"` oder `"REL"`, und `getLock()` das `SyncLock`, auf das sich die Aktion bezieht. (In der Klasse `SyncLock` sind die Methoden `equals()` und `hashCode()` geeignet überschrieben.)

Schreiben Sie eine Klasse `Concurrency` mit einem parameterlosen Konstruktor und der Methode

```
public boolean isPossible(Prozessschritt[] schritte),
```

die zurückgibt, ob die Prozessschritte in der angegebenen Reihenfolge ausgeführt werden können (`true`) oder nicht (`false`). Eine Reihenfolge ist nicht möglich, wenn

- ein Prozess ein `SyncLock` sperrt, das nicht frei ist,
- ein Prozess ein `SyncLock` frei gibt, das er nicht zuvor selbst gesperrt hat.

Beispiele: Eine mögliche Reihenfolge:

Prozess	1	2	1	3	2
Aktion	ACQ	ACQ	REL	ACQ	REL
SyncLock	A	B	A	A	B

Eine nicht mögliche Reihenfolge (Prozess 3 gibt ein nicht von ihm gesperrtes `SyncLock` frei):

Prozess	1	2	3	3	2
Aktion	ACQ	ACQ	REL	ACQ	REL
SyncLock	A	B	A	A	B

Eine andere nicht mögliche Reihenfolge (Prozess 3 sperrt ein bereits gesperrtes `SyncLock`):

Prozess	1	2	3	2
Aktion	ACQ	ACQ	ACQ	REL
SyncLock	A	B	A	B