Name:

Matrikelnummer:

Bearbeitungszeit: 120 min.

Aufgabe 1.

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

```
public Map<Integer,Auftrag> lasteEin(Auftrag[] auftraege),
```

die die gegebenen Aufträge an mehreren Tagen einlastet. Tage werden als ganzzahlige Werte durchnummeriert. Die Klasse `Auftrag` ist gegeben mit den Methoden

```
public int ersterTag(),
public int letzterTag(),
public int anzTage(),
```

die den ersten und den letzten Tag angeben, an denen der Auftrag bearbeitet werden kann, sowie die Anzahl von Tagen, die zur Bearbeitung des Auftrags notwendig sind.

An einem Tag kann nur ein Auftrag bearbeitet werden. Ein Auftrag kann an mehreren Tagen bearbeitet werden, die nicht direkt aufeinander folgen müssen. Zum Beispiel ist

Tag 1	Auftrag 1
Tag 2	Auftrag 2
Tag 5	Auftrag 3
Tag 7	Auftrag 2

eine mögliche Einlastung.

Die berechnete Einlastung `Map<Integer,Auftrag> einlastung` soll für die notwendige Anzahl von Tagen angeben, an welchem Tag welcher Auftrag bearbeitet werden soll. Dabei muss für jeden Auftrag `auftr` in `auftraege` die Anzahl der Tage mit

```
einlastung.get(tag) == auftr
```

genau `auftr.anzTage()` entsprechen. Weiters muss

```
auftr.ersterTag() <= tag <= auftr.letzterTag()
```

sein.

Wenn die Einlastung aller Aufträge nicht möglich ist, dann soll die Methode `lasteEin()` den Wert `null` zurückliefern.

Sie können annehmen, dass die Aufträge bereits aufsteigend nach ihrem letzten Tag sortiert sind.

Hinweis: Eine einfache Lösung erhält man, wenn die Aufträge in der Reihenfolge ihres letzten Tages eingelastet werden.

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

```
public Search(int x, int y),
public boolean isAtGoal(),
public void makeStep(int action).
```

Die Klasse `Search` stellt die Suche auf einem 2-dimensionalem Gitter dar. Der Konstruktor erstellt eine zufällige Zielposition und startet die Suche an der Position `(x,y)`. Die Methode `isAtGoal()` liefert `true` genau dann, wenn die Zielposition erreicht ist. Die Methode `makeStep(action)` bewirkt eine Bewegung nach links, rechts, oben oder unten, wobei `action` einer der Werte `1,2,3,4` ist.

Schreiben Sie eine Unterklasse `MovingSearch` von `Search`, die eine Suche darstellt, bei der sich die Zielposition verändert: Bei jedem Aufruf von `makeStep()` soll sich die Zielposition zufällig nach links, rechts, oben oder unten bewegen. Der Konstruktor

```
public MovingSearch(int x, int y),
```

startet wieder die Suche an der angegebenen Position und erstellt eine zufällige Startposition des Ziels. Überschreiben Sie eventuell auch die weiteren Methoden.

Aufgabe 3.
Mehrere Lieferungen mit unterschiedlichen Stückanzahlen sollen so auf zwei Standorte verteilt werden, dass jeder Standort genau die gleiche Anzahl von Stücken erhält.

Schreiben Sie ein Klasse `Aufteilen` mit einem parameterlosen Konstruktor und der Methode

```
public boolean istMoeglich(int[] anzahlen),
```

die überprüft, ob eine gleichmäßige Aufteilung möglich ist.

Z.B. `istMoeglich({3,5,6,7,11}) == true`, da $3 + 6 + 7 = 5 + 11$,
aber `istMoeglich({3,6,6,7,12}) == false`, da keine gleichmäßige Aufteilung möglich ist.