

Übung 04: Interfaces, Generics, Collections

Abgabetermin: 2. 4. 2009, 8:15

Name: _____

Matrikelnummer: _____

Informatik: G1 (Prähofer) G2 (Prähofer) G3 (Würthinger) G4 (Prähofer)

WIN: G1 (Khalil) G2 (Khalil) G3 (Schwinger)

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Punkte
Übung 4	24	Prosabeschreibung, Java-Programm, Ausgabe eines Testlaufs	Java-Programm	<input type="checkbox"/>	

Übung 04: Zugfahrplan

In dieser Übung ist unter Verwendung der generischen Collection-Klassen ein Klassensystem für den Aufbau und die Abfrage eines Zugfahrplans zu implementieren. Ein Zugfahrplan (Klasse `Timetable`) besteht aus Objekten Bahnhof (`Station`), Zug (`Train`) und eine Reihe von Stopps¹ (Klasse `Stop`), die zu einem Zug gehören und zu einer bestimmten Zeit, in einem bestimmten Bahnhof und auf einem bestimmten Bahnsteig stattfinden.

Züge (Klasse `Train`)

Züge haben einen eindeutigen Namen und speichern eine Liste von Stopps aufsteigend nach der Zeit sortiert. Die Klasse `Train` soll folgende Operationen zur Verfügung stellen:

- Das Anfügen der Stopps an einem bestimmten Bahnhof, auf einem bestimmten Bahnsteig und zu einer gegebenen Zeit (gleichzeitig soll der Stopp auch beim Bahnhof angefügt werden)
- Zugriff auf den ersten und letzten Stopp, bzw. auf den Quell- und Zielbahnhof
- Zugriff auf alle Stopps des Zugs in aufsteigender zeitlicher Reihenfolge
- Zugriff auf die Menge der Stopps ab einem gegebenen Bahnhof (der Name des Bahnhofs wird als Parameter übergeben)

Bahnhöfe (Klasse `Station`)

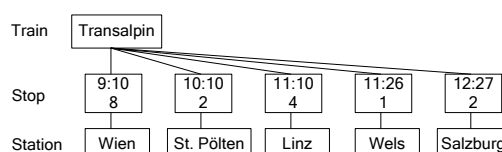
Bahnhöfe haben einen eindeutigen Namen und speichern die Liste der Stopps aufsteigend nach Zeit und bei Zeitgleichheit aufsteigend nach Bahnsteig sortiert. Die Klasse `Station` soll folgende Operationen zur Verfügung stellen:

- Zugriff auf alle Stopps an diesem Bahnhof
- Zugriff auf alle Züge, die an diesem Bahnhof halten

Stopps (Klasse `Stop`)

Stopps stellen die Verbindung zwischen Bahnhöfen und Zügen her. Sie speichern den Zug, zu dem sie gehören, den Bahnhof, den Bahnsteig und die Uhrzeit (von einem `Stop`-Objekt kann man daher auf das `Train`- und `Station`-Objekt zugreifen).

Folgende Abbildung zeigt anhand eines Beispiels die Zusammenhänge zwischen den Objekttypen `Train`, `Station`, und `Stop`.



¹ Um das System einfach zu halten, wollen wir nicht Ankunft und Abfahrt unterscheiden, sondern nur mit Stopps zu einer bestimmten Zeit arbeiten.

Zugfahrplan (Klasse Timetable)

Der Zugfahrplan hat eine Menge von Bahnhöfen. Die Menge der Bahnhöfe ist nach dem Namen aufsteigend sortiert.

Der Zugfahrplan speichert des Weiteren die Menge der Züge. Die Menge der Züge sei aufsteigend nach der Abfahrtszeit vom Quellbahnhof und bei Zeitgleichheit nach dem Namen des Quellbahnhofs sortiert. Stellen Sie entsprechende Zugriffsmethoden zur Verfügung.

Testen Sie das System, indem Sie einen einfachen Zugfahrplan mit mehreren Zügen und Bahnhöfen aufbauen und dann entsprechende Abfragen durchführen und diese in formatierter Weise ausgeben:

- Ausgeben aller Bahnhöfe in alphabetisch sortierter Reihenfolge, zusammen mit den Zeiten der Zugstopps und dem Zielbahnhof des stoppenden Zuges

```
Bahnhof: Linz
  10:12 Graz
  10:46 Salzburg
  ...
Bahnhof: Wels
  11:03 Salzburg
  ...
```

- Ausgeben aller Züge, die an einem bestimmten Bahnhof halten, mit allen dem Bahnhof folgenden Stopps des Zuges

```
Zugverbindungen von Linz
Steirerland:
  10:12 Linz (2)
  10:45 Kirchdorf (2)
  11:18 Rottenmann (2)
  12:09 Graz (1)
Donauwalzer:
  10:46 Linz (3)
  11:03 Wels (1)
  12:33 Salzburg (3)
```

- Ausgeben aller Züge zusammen mit allen Stopps

```
Donauwalzer
  09:10 Wien (1)
  10:46 Linz (3)
  11:03 Wels (1)
  12:33 Salzburg (3)

Steirerland
  10:12 Linz (2)
  10:45 Kirchdorf (2)
  11:18 Rottenmann (2)
  12:09 Graz (1)
```

Hinweise:

Verwenden Sie die Interfaces `List`, `Set` und `SortedSet` und Klassen `ArrayList`, `HashSet` und `TreeSet` vom `Collection`-Package (`java.util`).

Überlegen Sie genau den Einsatz der unterschiedlichen Collections `List`, `Set` und `SortedSet`.

Überlegen Sie genau die Realisierung der Sortierreihenfolgen für die unterschiedlichen Objektsammlungen. Dies kann entweder durch die Implementierung des `Comparable`-Interfaces durch die Klassen oder durch die Definition eigener `Comparator`-Klassen erfolgen.

Für die Zeiten können Sie wieder die Klasse `datetime.Time` verwenden. Diese implementiert bereits ein `compareTo`, was in der Aufgabe sinnvoll eingesetzt werden kann.