

## Übung 02: Vererbung und Exceptions

Abgabetermin: 03.04.2008, 8:15

**Name:** \_\_\_\_\_

**Matrikelnummer:** \_\_\_\_\_

**Informatik:**  G1 (Prähofer)     G2 (Prähofer)     G3 (Wimmer)     G4 (Wimmer)

**WIN:**         G1 (Ibrahim)     G2 (Ibrahim)     G3 (Schwinger)

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Punkte
Übung 2	24	Prosabeschreibung, Java-Programm, Ausgabe eines Testlaufs	Java-Programm	<input type="checkbox"/>	

### Übung 02: Transporter (24 Punkte)

#### a) Transporter

In dieser Übung sollen Sie ein Klassensystem für unterschiedliche Transportfahrzeuge (Transporter) realisieren. Transportfahrzeuge können zu unterschiedlichen Orten (Locations) fahren und Güter (Cargo) unterschiedlicher Art (fest und flüssig) und mit unterschiedlichem Gewicht transportieren. Für die Fahrten fallen Kosten an, die zu berechnen sind. Es gelten für den Transport unterschiedliche Einschränkungen, die geprüft werden und bei Verletzung Exceptions auslösen sollen. Es folgt eine genaue Beschreibung der zu realisierenden Klassen.

#### Klasse Location

Diese Klasse `Location` steht Ihnen als Download zur Verfügung. Locations stellen Orte im Transportsystem dar, die von den Transportern angefahren werden können. Eine `Location` ist definiert durch den Kontinent, auf dem sich der Ort befindet, den x/y-Koordinaten, sowie einen Namen. Die Klasse hat auch eine Methode zur Berechnung der Distanz zwischen Orten (wobei hier stark vereinfacht für die Luftliniendistanz der Abstand der x/y-Koordinaten verwendet wird).

#### Klasse Cargo

`Cargo` ist die Klasse für die Transportgüter. Transportgüter sind charakterisiert durch folgende Eigenschaften:

- Typ: flüssig oder fest
- Gewicht: in kg
- Bezeichnung: ein String zur Bezeichnung des Transportguts

Realisieren Sie für den Typ von Transportgütern einen Enumerations-Typ `CargoType` mit den beiden Werten `LIQUID` und `SOLID`.

#### Klasse Transporter:

`Transporter` ist die Basisklasse für Transportfahrzeug. Jeder `Transporter` hat

- einen Bezeichner
- ein maximales Transportgewicht
- eine Variable, die die Fahrkosten pro km angibt
- eine aktuelle Location
- ein aktuell geladenes Transportgut

Ein `Transporter` kann zu einer Location fahren

```
double goTo(Location destination)
```

wobei als Ergebnis der Methode die Fahrtkosten zurückgegeben werden. Die Kosten berechnen sich üblicherweise aus den Kosten pro km mal der Distanz von der aktuellen Location zum Ziel.

Ein Transporter kann weiters ein Transportgut laden

```
void load(Cargo cargo)
```

und sein Transportgut abladen

```
Cargo unload()
```

Klassen für spezielle Transporter:

Folgende Typen von Transportern sollen durch spezielle Klassen realisiert werden:

- Transportflugzeug (Klasse CargoPlane)
- Container-LKW (Klasse ContainerTruck)
- Tank-LKW (Klasse TankTruck)

Diese haben spezielles Verhalten wie folgt:

- Bei einem Transportflugzeug kommen bei den Kosten für die Fahrt konstante Kosten für den Start und die Landung dazu.
- Ein Tank-LKW kann kein festes Transportgut transportieren.
- Ein Flugzeug und ein Container-LKW können kein flüssiges Transportgut transportieren.
- Ein LKW kann nicht zu einer Location fahren, die sich auf einem anderen Kontinent befindet.

Dieses spezielle Verhalten sollen Sie durch Überschreiben der entsprechenden Methoden und durch Auslösen von spezifischen Exceptions realisieren. Implementieren Sie dazu entsprechende Exception-Klassen.

Anleitungen:

Folgendes ist bei der Implementierung zu beachten:

- Implementieren Sie für alle Klassen Konstruktoren, die es erlauben, die Objekte sinnvoll zu initialisieren
- Implementieren Sie für alle Klassen toString-Methoden
- Setzen Sie immer wenn möglich super-Aufrufe ein
- Führen Sie wenn sinnvoll weitere Zwischenklassen ein

b) *Test*

Schreiben Sie einen Test, mit dem für jede Operation und für jeden Transportertyp sowohl die korrekte Verwendung als auch die fehlerhafte Verwendung (die zu einer Exception führt) getestet wird.

Anmerkungen:

Bei der korrekten Verwendung einer Operation muss getestet werden, dass die Methode das richtige Ergebnis liefert und dass die Methode keine Exception wirft. Folgendes Programmfragment zeigt schematisch, wie man einen solchen Test implementiert.

```
try {
    call operation to test with valid parameter values
    test and print result of operation
} catch (Exception e) {
    Print out that unexpected exception has been thrown
}
```

Bei der inkorrekten Verwendung einer Operation muss getestet werden, dass die Methode tatsächlich eine Exception wirft. Wird diese nicht geworfen, stellt das einen Fehler dar.

```
try {
    Call operation to test with invalid parameter values
    Print out that expected exception has not been thrown
} catch (Exception e) {
    Print out that expected exception has been thrown
}
```