SoSe 2014
Prof. Dr. Margarita Esponda
ALP2
9. Übungsblatt (Abgabe am 23. Juni)

Ziel: Weitere Auseinandersetzungen mit OO-Konzepten, insbesondere Schnittstellen, Polymorphie, Ausnahmefehler (Exceptions) und generische Klassen in Java.

1. Aufgabe (10 Punkte)

a) Wann ist es sinnvoll, Methoden als Klassenmethoden zu definieren? Welche Einschränkungen entstehen dabei?

b) Warum macht es keinen Sinn, lokale Variablen als **private** zu deklarieren? Können Klassenmethoden als **private** deklariert werden? Wenn ja: wann wäre das sinnvoll?

c) Was passiert, wenn in einer Klassendefinition kein Konstruktor definiert wird? Können Konstruktoren vererbt werden? Können Konstruktoren als **private** deklariert werden?

f) Warum muss die **main**-Methode immer als **static** definiert werden und was passiert, wenn das nicht so ist?

g) Wo sind Instanzvariablen sichtbar, wenn diese als **protected** deklariert werden? Können Konstruktoren als **protected** deklariert werden?

h) Was sind abstrakte Klassen? Wozu sind sie gut? Können Variablen einer abstrakten Klasse deklariert werden?

i) Können Objekte einer abstrakten Klasse erzeugt werden? Kann man Konstruktoren in abstrakten Klassen definieren?

b) Was ist ein Interface (Schnittstelle)? Wozu sind Interfaces gut?

c) Kann eine Variablen vom Typ A einem Objekt der Klasse B zugewiesen werden, wenn A eine Unterklasse von B ist? Begründen Sie Ihre Antwort.

d) Was ist der Unterschied zwischen Methodenüberladung und Methodenüberschreibung?

e) Können Variablen einer Klassen in Unterklassen überschrieben werden?

f) Was ist automatisches boxing/unboxing? Warum wurde das in Java eingeführt? Welche Probleme gibt es in Java damit?

g) Was ist eine generische Klasse? Wozu sind diese gut?

h) Was verstehen Sie unter Vererbungspolymorphie?

i) Was sind Ausnahmefehler? Was ist der Unterschied zwischen Runtime-Exceptions und allgemeinen Exceptions in Java?

2. Aufgabe (24 Punkte)

a) Programmieren Sie eine generische **Driver**-Klasse, die einen Autofahrer modelliert, der verschiedene Fahrzeuge fahren kann bzw. mit verschiedenen konkreten Autofahrer-Klassen parametrisiert werden kann.

b) Definieren Sie für die Fahrzeuge zuerst eine abstrakte **Vehicle**-Klasse, die die allgemeinen Eigenschaften und Methoden eines Fahrzeugs modelliert. Hier sollen Eigenschaften wie

Führerscheinklasse, Marke, Tankgröße in Litern, Kilometer pro Liter (Durchschnitt) usw. definiert werden und eine Reihe abstrakter Methoden, die alle konkreten Unterklassen implementieren müssen.

c) Programmieren Sie mindestens zwei Unterklassen der **Vehicle** Klasse, die Sie in einer **TestVehicle**-Klasse benutzen sollen.

d) Die **Driver**-Objekte sollen mindestens folgende Methoden ausführen können.

```
public Time drive(int km, int speed)
                              throws NotEnoughFuelException;
/* Der Driver fährt sein Auto eine bestimmte Anzahl von Kilometern mit der
angegebenen Geschwindigkeit in Kilometer pro Stunde, wenn er genug Benzin/
Diesel dafür hat.
Die verbrauchte Zeit soll als Rückgabewert berechnet werden */

public int tankUp(); /* der Tank wird gefüllt */

public void tank(int liter)
                          throws NotEnoughCapacityException;
/* Der Driver versucht eine bestimmte Anzahl von Litern zu tanken. Eine
Exception wird geworfen, wenn dem Auto weniger Liter fehlen */
```

e) Mindestens folgende Hilfsmethode sollen dazu noch programmiert werden:

```
public boolean isEmpty(); /* Frage, ob Benzin/Diesel vorhanden ist */

public boolean isFull(); /* Frage, ob der Tank voll ist */
```

f) Programmieren Sie in der Exception-Klasse entsprechende Konstruktoren, in denen aussagekräftige Information für die Ausnahmebehandlung gespeichert wird.

g) Schreiben Sie eine **TestVehicle**-Klasse, um alle Methoden der **Driver**-Klasse ausführlich zu testen.

Wichtige Hinweise:

1) Verwenden Sie selbsterklärende Namen von Variablen und Methoden.
2) Für die Namen aller Bezeichner müssen Sie die Java-Konventionen verwenden.
3) Verwenden Sie vorgegebene Klassen und Methodennamen.
4) Methoden sollten klein gehalten werden, sodass auf den ersten Blick ersichtlich ist, was diese Methode leistet.
5) Methoden sollten möglichst wenige Argumente haben.
6) Methoden sollten entweder den Zustand der Eingabeargumente ändern oder einen Rückgabewert liefern.
7) Verwenden Sie geeignete Hilfsvariablen und definieren Sie sinnvolle Hilfsmethoden in Ihren Klassendefinitionen.
8) Zahlen sollten durch Konstanten ersetzt werden.
9) Löschen Sie alle Programmzeilen und Variablen, die nicht verwendet werden.