Prof. Dr. A. Weber, Dr. G. Kniesel
J. Tautges, P. Bihler
oose.cs.uni-bonn.de

Universität Bonn
Institut für Informatik II
Institut für Informatik III
27.05.2008

Sommersemester 2008

# Übungen zur Vorlesung
# Objektorientierte Softwareentwicklung (BA-INF-024)
# Aufgabenblatt 7

Zu bearbeiten bis: 06.06.2008

**Aufgabe 1** *(Vererbung, Ausnahmen - 2 Punkte)*

Überlegen Sie sich für jede Aussage ein Argument, mit dem Sie eindeutig bestimmen können, ob die Aussage wahr oder falsch ist. Für jede richtig angekreuzte Antwort werden 0,5 Punkte angerechnet, für jede falsch angekreuzte Antwort 0,5 Punkte abgezogen. Nicht angekreuzte Antworten geben keinen Punktabzug.

| wahr | falsch | |
|---|---|---|
| | | Unterklassen dürfen öffentliche Methoden einer Überklasse verdecken, indem in der Unterklasse entsprechende Methoden mit der gleichen Signatur und leerem Methodenrumpf definiert werden. |
| | | Konstruktoren werden automatisch von der Oberklasse übernommen. |
| | | Exceptions können nur dann geworfen werden, wenn die umgebende Methode ein entsprechendes `throws`-Statement besitzt. |
| | | Alle Exceptions, die für eine Methode mit einem `throws`-Statement definiert sind, müssen beim Aufruf der Methode behandelt werden. |

**Aufgabe 2** *(Begriffsdschungel - 6 Punkte)*

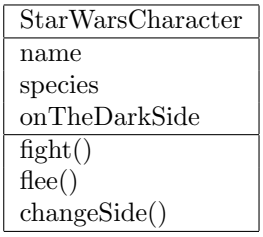
Ordnen Sie synonyme Begriffe einander zu. Greifen Sie dazu bei Bedarf auf Sekundärliteratur zurück:

| | | | |
|---|---|---|---|
| statische Variable | Aktion | Objektabstraktion | Klasse |
| Beziehung | Objekt | Klassenfunktion | Subklasse |
| Feld | Superklasse | Instanzvariable | Metafunktion |
| Parameter | Memberfunktion | Prozedur | Nachrichten |
| Instanz | Routine | Member | Klassenvariable |
| Namensraum | Argument | Oberklasse | Ereignisse |
| Relation | Paket | Eigenschaft | Datenelement |
| Objekttyp | Unterklasse | Operation | Attribut |
| statische Methode | Basisklasse | Exemplar | abgeleitete Klasse |
| Ableitung | Elementfunktion | Vererbung | Methode |

**Aufgabe 3** *(Wiederholung - 10 Punkte)*

In dieser Aufgabe geht es darum, verschiedene Konzepte der Objektorientierung, die sowohl in der Vorlesung als auch in der Übung bereits behandelt wurden, zu wiederholen.

**a**) Auf der Folie *2-13* des Foliensatzes *oose-02-oop-basics.pdf* wird die UML-Notation einer einfachen Java-Klasse dargestellt. Implementieren Sie eine Klasse `StarWarsCharacter` gemäß dem folgenden UML-Diagramm:




Die Variable `onTheDarkSide` soll dabei vom Typ `boolean` sein, `name` und `species` sollen durch `Strings` repräsentiert werden. Der Aufruf der Funktion `changeSide()` soll den internen Zustand `onTheDarkSide` ändern. Beim Aufruf von `fight()` bzw. `flee()` soll eine Ausgabe gemacht werden, die neben der Information *fleeing* oder *fighting* auch Name und Rasse der Instanz enthält.

**b**) Leiten Sie von der Klasse `StarWarsCharacter` die Klasse `Jedi` ab. Diese soll als zusätzliche `String`-Variable das Feld `rank`[1] besitzen, als zusätzliche Operationen die Methoden `increaseRank()` und `decreaseRank()`, die den internen Zustand `rank` entsprechend modifizieren. Erweitern Sie die Funktionen `fight()` und `flee()` dahingehend, dass auch der Rang eines Jedis ausgegeben wird (Die Implementierung der Oberklasse soll weiterhin aufgerufen werden!).

**c**) Da Flucht für einen Jedi, der mindestens den Rang *Knight* trägt, ausgeschlossen ist, muss die Methode `flee()` in der Klasse `Jedi` überschrieben werden: Während Jedis der beiden niedrigsten Ränge beim Aufruf von `flee()` wie gehabt die Flucht ergreifen sollen (greifen Sie hierzu auf die Implementierung der Oberklasse zurück), soll fortgeschritteneren Jedis die Fluchtmöglichkeit verwehrt bleiben: Hier soll stattdessen die Funktion `fight()` aufgerufen werden.

**d**) Erstellen Sie in den beiden Klassen `StarWarsCharacter` und `Jedi` Standardkonstruktoren, die alle Variablen nach Ihren Wünschen vorinitialisieren[2]. Erstellen Sie in der Klasse `Jedi` zusätzlich einen Konstruktor, der es erlaubt, den Rang zu setzen. Testen Sie Ihre Implementierung, indem Sie eine neue Klasse `Demo` erstellen, in deren `main`-Funktion Sie zunächst ein Jedi-Objekt mit dem Rang *Younglin* erschaffen, einmal fliehen, den Rang zweimal erhöhen und anschließend einen weiteren Fluchtversuch unternehmen!

[1] Wie Sie sicherlich wissen, existieren die Jedi-Ränge *Younglin*, *Padawan*, *Knight*, *Master*, *Council Member* und *Grand Master*.

[2] Hier finden Sie Anregungen für die Wahl der `species`: `http://en.wikipedia.org/wiki/List_of_Star_Wars_races`

**Aufgabe 4** *(Threads - 6 Punkte)*

**a**) Nennen Sie die beiden Möglichkeiten, die Java anbietet um Threads zu erzeugen, und geben Sie für jede Methode einen Vorteil an.

**b**) Die Klasse `ThreadTest` sei vorgegeben:

```
public class ThreadTest {
  public static void main(String args[]){
    Thread t1 = new Thread(new DateCommand());
    t1.start();
    Thread t2 = new Thread(new CounterCommand());
    t2.start();
  }
}
```

Erstellen Sie die Klassen `DateCommand` und `CounterCommand` durch Implementieren des Interfaces `Runnable`. Der parallel auszuführende Programmcode soll in beiden Klassen lediglich aus einer `for`-Schleife bestehen, die zehnmal durchlaufen wird. In der Klasse `DateCommand` soll in jedem Schleifendurchlauf das aktuelle Datum ausgeben werden[3], in der Klasse `CounterCommand` lediglich der aktuelle Wert der Schleifenvariablen.

**c**) Benutzen Sie die Methode `sleep()` der Klasse `Thread`, um die Schleifendurchläufe zu verzögern. Nutzen Sie die Funktion `new java.util.Random().nextInt(1000)` um die Länge der Verzögerung zufällig zu wählen. Hinweis: Da eine `InterruptedEception` ausgelöst wird, wenn ein Thread unterbrochen wird, muss die Verzögerung in einem `try-catch`-Block untergebracht werden.[4] Führen Sie das Programm `ThreadTest` aus und erläutern Sie das Ergebnis!

[3] Auf das aktuelle Datum können Sie einfach über die Methode `Date()` aus dem Paket `java.util` zugreifen.
[4] Ausnahmebehandlungen können vernachlässigt werden.