

## Modellierung und Programmierung 1 Übungsserie 4

Abgabetermin: 22.12.2013, 23:55 Uhr

Grundsätzlich sind Nebenrechnungen anzugeben und Antworten zu begründen.  
Einzureichen sind, bei mehreren Dateien als .zip-Archiv:  
Lösungen und UML-Klassendiagramme als .pdf-Datei, Programme als Quellcode,  
Ergebnisdateien, Online-Dokumentationen.

### 1. Rekursion

Der Binomialkoeffizient  $\binom{n}{k}$  ist eine Funktion, welche zum Beispiel in Aufgaben der Kombinatorik Anwendung findet: Er gibt an, auf wie viele verschiedene Arten man  $k$  Elemente aus einer Menge von  $n$  Elementen auswählen kann.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

mit  $n, k \in \mathbb{N}$  und  $k \leq n$ ,  $\binom{n}{k} = \binom{n}{n-k}$  und  $\binom{n}{n} = \binom{n}{0} = 1$ .

- a) Begründen Sie, warum die oben angegebene Formel für die praktische Berechnung des Binomialkoeffizienten in Java nicht geeignet ist.  
Entwerfen Sie in einer Klasse **Binom.java** zunächst eine iterative Methode **binom1** zur Berechnung von Binomialkoeffizienten ohne Einsatz der Fakultät. Multiplizieren und dividieren Sie stets im Wechsel. Achten Sie auf die Effizienz Ihrer Implementierung.
- b) Implementieren Sie in der Klasse **Binom.java** eine rekursive Methode **binom2**, welche zur Berechnung von Binomialkoeffizienten das Pascalsche Dreieck nutzt:

$$\binom{n+1}{k+1} = \binom{n}{k+1} + \binom{n}{k}$$

- c) Implementieren Sie eine weitere rekursive Methode **binom3**, welche Binomialkoeffizienten mit Hilfe der folgenden Formel berechnet:

$$\binom{n+1}{k+1} = \binom{n}{k} \cdot \frac{n+1}{k+1}$$

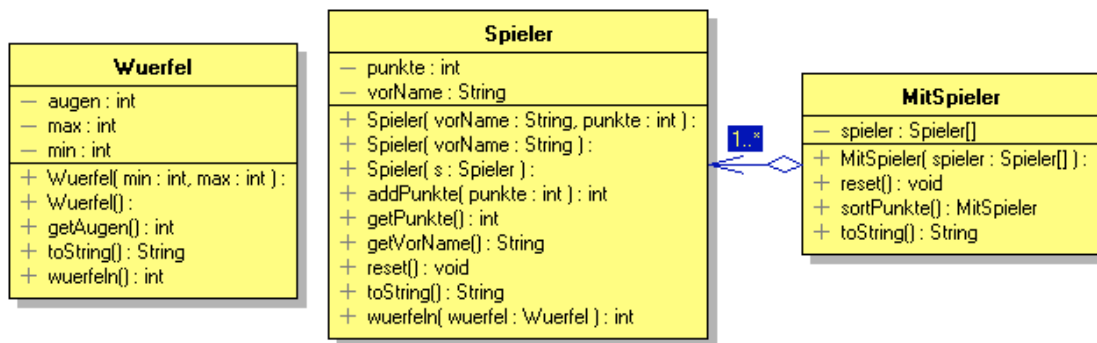
- d) Bestimmen Sie das Ergebnis der drei Methoden und die Anzahl der notwendigen Funktionsaufrufe der rekursiven Methoden für  $\binom{19}{17}$  und für  $\binom{19}{13}$ . Diskutieren Sie die Resultate, auch in Hinblick auf die Effizienz der Methoden.

Hinweis: Verwenden Sie bei allen Berechnungen den Datentyp **long**.

### 2. Klassen

Würfelspiele werden oft mit zwei oder mehr Mitspielern gespielt. Schaffen Sie entsprechend dem gegebenen Klassendiagramm eine neue Klasse **MitSpieler**, in der alle Mitspieler zusammengefasst und verwaltet werden. Verwenden Sie dabei die in den Klassen **Würfel** und **Spieler** zur Verfügung stehenden Methoden aus **MitSpieler.zip**.

Hinweis: Achten Sie auf die Einhaltung der Signaturen in den Methoden.



- Ein Konstruktor erzeugt aus einem Spielerfeld ein Objekt der Klasse **MitSpieler**.
- Eine Methode **toString()** stellt die aktuellen Spielerdaten aller Spieler als String zusammen. Für ein neues Spiel setzt eine Methode **reset()** die Punkte jedes Spielers zurück.
- Eine weitere Methode **sortPunkte()** soll das Spielerfeld nach Punkten sortieren: Um die Reihenfolge der Spieler nicht zu verändern, wird das Spielerfeld zunächst kopiert und die Spieler in der Kopie sortiert. Wählen Sie dabei eines der Ihnen bekannten Sortierverfahren.
- Testen Sie Ihre Klasse mit dem mitgelieferten Testprogramm **MitSpielerTest**. Erstellen Sie eine Online-Dokumentation für alle verwendeten Klassen mit

```
javadoc -private -d MitSpielerDoc *.java.
```

Hinweis:

Beachten Sie, dass auch alle *privaten* Attribute und Methoden zu dokumentieren sind.

### 3. Modellierung

Der verrückte Wissenschaftler **Dr. E. Ville** ist versucht, die Weltherrschaft an sich zu reißen. Sein geheimes unterirdisches **Versteck** umfasst die aus mehreren Räumen großzügig angelegte **Wohnung** des Wissenschaftlers und die auf mehrere **Etagen** verteilten **Labore**, **Lagerräume**, **Quartiere** und **Zellen**.

Jeder dieser **Räume** hat eine eindeutig zugeordnete *Raumnummer*, bestehend aus der *Etagennummer* und einer *Zimmernummer*. Aus Sicherheitsgründen ist der Zugang zu den Räumen eingeschränkt. Jede **Person** besitzt eine Karte mit einem **Zugangscode**. Jeder Raum registriert das Zutrittsrecht von Personen über deren *Zugangscode*.

**Lagerräume** haben *ein momentan* und *ein maximal eingelagertes Volumen an Material*. Desweiteren wird vermerkt, ob *gefährliches Material* aufbewahrt wird. **Labore** haben eine *festgelegte Anzahl von Arbeitsplätzen*. Manchen Laboren ist *ein spezieller Lagerraum* zugeordnet. In den einfach ausgestatteten **Quartieren** schlafen die **Wissenschaftler**. In den **Zellen** schlafen **Gefangene**. Diese **Schlafräume** sind nur für eine *begrenzte Anzahl von Personen* ausgelegt.

Um seine bösartigen Eroberungspläne effizient vorantreiben zu können und sich vor eventuellen Angriffen von **007** zu schützen, benötigt Dr. E. Ville ein Raumverwaltungsprogramm für sein zwielichtiges Institut.

Modellieren Sie hierzu eine entsprechende Klassenhierarchie:

- Stellen Sie für alle Klassen ein entsprechendes UML-Klassendiagramm mit den nötigen Attributen, ohne Funktionalitäten, zusammen. Berücksichtigen Sie in Ihrer Klassenhierarchie Vererbungsbeziehungen der Klassen untereinander. Achten Sie auf kleine wiederverwendbare Klassen.
- Ergänzen Sie Ihre Klassenhierarchie durch die Darstellung von bestehenden Assoziationen, Aggregationen und Kompositionen, einschließlich der Multiplizitäten. Begründen Sie Ihre Wahl.