Lehrstuhl Stuckenschmidt
Javakurs FSS 2012
Tag 4 - Ausgewählte Datenstrukturen

Tim Endlich, Thorsten Knöller, Niklas Dürr, Christian Meilicke

27.04.2012

Aufgabe 1: Prozess-Prioritäten

`prozess`

Es sollen im folgenden Prozesse geschedult werden. Da das System nur über begrenzte Resourcen verfügt, müssen die Prozesse in einer Warteschlange gespeichert werden und dann der Reihe nach abgearbeitet werden. Dabei wird die Auwahl des als nächstes abzuarbeitenden Prozesses bestimmt durch

- die Priorität des Prozesses (int Wert zwischen 1 und 10, 1 = niedrigste Priorität, 10 = höchste Priorität),
- den Ankunftszeitpunkt.

Dabei soll stets ein Prozess mit höherer Priorität bevorzugt werden. Wenn zwei Prozesse die gleiche Priorität haben, dann wird der bevorzugt, der zuerst angekommen ist.

Schreiben Sie ein Programm, dass die Prozesse korrekt Scheduled. Mit der Klasse `Prozess` können Sie Prozesse erstellen, welche Sie dann in Ihre Wartechlange eingefügen können.
Verwenden Sie zum Sortieren der Warteschlange einmal das Interface `Comparable` und einmal das Interface `Comparator`. Legen Sie für den `Comparator` eine neue Klasse mit dem Name `ProcessComparator` an.

Aufgabe 2: Poker *

poker

Schreiben Sie ein Programm, welches berechnet wie wahrscheinlich es ist, ein Full-House beim Pokern zu ziehen. Ein Full-House sind beispielweise zwei 7er und drei Damen. Wir betrachten eine Variante, bei der man 5 Karten aus einem 32 Karten Skatblatt (7, 8, 9, 10, Bube, Dame, König, Ass jeweils in den Farben Karo, Herz, Pik, Kreuz) zieht. Es dürfen keine Karten getauscht bzw. nachgezogen werden. *Achtung: Die Aufgabe kann natürlich (relativ einfach) analytisch gelöst werden. Aber zu Übungszwecken wollen wir die Aufgabe mittels einer Monte-Carlo Simulation lösen.*

Einige Klassen mit Methodenrümpfen sind bereits vorgegeben. Vervollständigen Sie diese und beachten Sie auch jeweils die Dokumentation der Methoden. Gehen Sie dabei folgendermaßen vor:

a) Beginnen Sie mit der Klasse `Card`! Erzeugen Sie anschließend in einer `main` Methode einige Objekte diesen Typs und lassen Sie sich diese ausgeben!

b) Die Klasse `Stack` repräsentiert einen vollständigen Kartenstapel. Dieser kann mittels eines Konstruktors erzeugt werden und man kann die erste Karte vom Stapel abnehmen. Die Karten eines Stapels sollen in einer `ArrayList` verwaltet werden. Testen Sie diese Klasse, indem Sie in einer `main` Methode einen Stapel erzeugen, 5 Karten von diesem ziehen und die gezogenen Karten in einer `ArrayList` abspeichern! Sortieren Sie diese 5 Karten und lassen Sie sich die sortierten Karten anzeigen!

c) Implementieren Sie die statische Methode `isFullHouse` in der Klasse `PokerRules`! Hinweis: Wenn Sie die Eingabe `hand` in der Methode zunächst sortieren, läßt sich die Funktionalität recht einfach implementieren. Nebenfrage: Wieso ist diese Methode statisch?

d) Schreiben Sie nun eine `main` Methode, in der Sie die eigentliche Aufgabe lösen (Monte-Carlo Simulation)! Die gesuchte Wahrscheinlichkeit liegt zwischen 0.5% und 1%.

Aufgabe 3: Wörterbuch *

`dict`

Bei dieser Aufgabe sind keine Klassen und Methoden vorgegeben. Sie müssen sich selbst überlegen, welche Aufteilung in Klassen und Methoden am meisten Sinn macht! Ebenso müssen Sie sich einige Beispiele überlegen, anhand derer Sie die Korrektheit ihrer Implementierung überprüfen!

Schreiben Sie ein Programm, mit dem Sie Wörterbücher (deutsch-englisch) verwalten können! Zunächst einmal sollte es möglich sein, ein Wörterbuch zu erzeugen. Einem Wörterbuch sollte man Einträge hinzufügen können und es sollte möglich sein ein gesamtes Wörterbuch darzustellen. Sucht man nach einer Übersetzung zu einem Wort, so sollte es hierfür eine Methode geben. Weiterhin sollte die Funktionalität unterstützt werden, zwei Wörterbücher zu einem neuen Wörterbuch zu vereinigen. Dabei sind verschiedene Fälle zu unterscheiden (Einträge zu verschiedenen Wörtern, zwei gleiche Einträge zum selben Wort, zwei verschiedene Einträge zum selben Wort). Überlegen Sie sich, was in diesen Fällen optimalerweise passieren sollte und implementieren Sie dies!

Beispiel A für ein Wörterbuch:

```
Agent  agent
Berg   mountain
Feuer  fire
Haus   house
```

Beispiel B für ein Wörterbuch:

```
Berg   hill
Feuer  fire
Krieg  war
```

Vereinigung von A und B:

```
Agent  agent
Berg   hill, mountain
Feuer  fire
Haus   house
Krieg  war
```

Noch ein Tip: Verwendet man die Datenstruktur `HashMap<String, HashSet<String>>`, dann läßt sich die Funktionalität recht leicht implementieren.