Richtlinien und Hinweise für die MW-Übungsaufgaben

- Jeder Teilnehmer bekommt ein Projektverzeichnis unter `/proj/i4mw/<loginname>`. Dieses kann als Workspace für die Übungsaufgabe dienen.
- Die Lösungen sollen in 2-3er-Gruppen erstellt werden. Jede Gruppe bekommt ein Projektrepository unter `https://www4.informatik.uni-erlangen.de:8088/i4mw/<gruppe>`.
- Die Gliederung des Repositories sollte sich am Aufbau der Implementierung orientieren, also z.B. `MWLibrary`, ... (anstatt `aufgabe1`, ...)
- Zur Erstellung eines SVN-Repositories muss von einem Übungspartner das Programm `/proj/i4mw/bin/mwgroups` aufgerufen werden. Die Adresse des Repositories wird innerhalb von 48 Stunden per E-Mail mitgeteilt.
- Zum Abgabezeitpunkt muss die finale Version der Lösung in das Repository eingecheckt sein, die Abgabe erfolgt in der Rechnerübung.
- Die *Java Coding Guidelines* sollten befolgt werden:
    - Klassennamen beginnen mit einem *Großbuchstaben*.
    - Methodennamen beginnen mit einem *Kleinbuchstaben*.
    - Variablennamen beginnen mit einem *Kleinbuchstaben*.
    - Konstante (final static) Variablen bestehen nur aus *Großbuchstaben*.
    - Variablenamen sind ausdrucksfähig und beschreiben deren Zweck.
- Wenn eine bestimmte Programmstruktur in der Aufgabenstellung verlangt wird, muss die Lösung diese einhalten. Die Lösung soll
    - Namen von Klassen und Packages
    - Sichtbarkeit von Variablen und Methoden
    - Namen, Parametertypen und Rückgabewerte von Methoden

  wie in der Aufgabenstellung beschrieben verwenden.
  Abweichungen hiervon sind zu begründen.
- Es wird negativ gewertet, falls
    - verlangte Funktionalität nicht implementiert wird
    - nicht verlangte, unsinnige Funktionalität implementiert wird
    - das Programm nicht kompiliert.
- Der Code muss lesbar und nachvollziehbar sein. Falls komplizierte Teile vorkommen, sollten diese mit Kommentaren erläutert werden.
- Die Lösungen müssen eigenständig erstellt worden sein. Verstöße werden geahndet.

Bei Problemen mit der Aufgabenstellung könnt ihr euch jederzeit an uns wenden:
mw@i4.informatik.uni-erlangen.de

Übungsaufgabe #1: MWLibrary - Client/Server

29.10.2009

In dieser Aufgabe wird zunächst das Grundgerüst für eine Bibliotheksverwaltung erstellt. Es dient in den folgenden Aufgaben als Basis und sollte daher gut strukturiert werden. *Falls nötig, finden sich auf der Homepage zusätzliche Folien zur Objektorientierung, Vererbung und zum Exception-Handling.*

Teil 1:

Es sollen Klassen für Medienobjekte (Buch, CD, Zeitschrift) erstellt, sowie eine einfache Datenbank implementiert werden, welche diese Objekte verwaltet. Außerdem soll eine Benutzerschnittstelle erstellt werden um mit der Bibliotheksverwaltung zu interagieren.

a) Erstelle ein Interface `Item`, das die gemeinsamen Methoden aller Medienobjekte definiert. Im Hinblick auf spätere Aufgaben sind mindestens folgende Methoden vorzusehen:

   `String getTitle()` und `void setTitle(String title)`
   Zum Setzen und Abfragen des Titels.
   (Der Titel steht stellvertretend für die Informationen über das Medium)

   `void borrow()` und `int giveback()`
   Hiermit wird das Objekt als ausgeliehen bzw. zurückgeben markiert.
   Falls es bereits verliehen ist, soll eine `AlreadyBorrowedException` geworfen werden.
   Außerdem soll ein Ausleihzähler verwaltet werden, der bei jedem Ausleihen erhöht wird.
   `giveback` liefert den aktuellen Ausleihzählerwert zurück.

   `String toString()`
   Die über ein Medium gespeicherten Informationen (Titel, Ausleihstatus, Ausleihzähler, ...) sollen ausgegeben werden.

b) Erstelle drei Klassen `Book`, `Journal` und `CD`, die alle das Interface `Item` implementieren.

c) Schreibe die Datenbank-Klasse `SimpleDBImpl` zur Verwaltung der `Item`-Objekte. Dazu ist ein Interface `SimpleDB` mit mindestens folgenden Methoden zu implementieren:

   `void register(Object key, DBObject data)`
   Fügt ein neues Objekt `data` unter dem eindeutigen Schlüssel `key` der Datenbank hinzu.
   Als Schlüssel kann ein beliebiges Objekt dienen, Daten sollen das Interface `DBObject` implementieren. `DBObject` soll dabei ein leeres "Marker-Interface" sein.

   `Object[] getAllKeys()`
   Gibt die Schlüssel aller gespeicherten Objekte als Array zurück

   `DBObject get(Object key)`
   Liefert das Objekts mit dem entsprechenden Schlüssel zum lesenden Zugriff zurück.

   `DBObject lockAndGet(Object key)`
   Wie `get()`, jedoch für schreibenden Zugriff. Vor einem erneuten Aufruf von `lockAndGet()` muss das Objekt mit `unlockAndPut()` wieder freigegeben werden,

   `void unlockAndPut(Object key, DBObject data)`
   Gibt ein mit `lockAndGet`() gesperrtes Objekt wieder frei.

   Eine mögliche Implementierung könnte darin bestehen die Objekte in zwei verschiedenen Hashtabellen zu speichern, eine für gesperrte und eine für zugreifbare Objekte.

   Im Fehlerfall sollen geeignete Exceptions generiert werden (z.B. `AlreadyExistsException`, `NotFoundException` und `AlreadyLockedException, NotLockedException`).

d) Schreibe eine Klasse `LibraryFrontend`, die als Schnittstelle zwischen dem Benutzer und der Datenbank dient und mindestens folgende Methoden aufweist:

   `void registerItem(String classname, String title)`
   Fügt ein neues Objekt der Datenbank hinzu. Durch `classname` wird der Typ des Objekts angegeben (`Book`, `CD` oder `Journal`). (Klasse dynamisch laden!)

   `void list()`
   Zeigt eine Liste aller Medien an.

`void borrowItem(String title)`
: Leiht das Objekt aus und erhöht den Ausleihzähler erhöhen.

`int returnItem(String title)`
: Das Objekt wird wieder zurückgegeben. Der Rückgabewert entspricht dem Ausleihzähler.

Fehler sind durch geeignete Exceptions auszudrücken (z.B. `AlreadyExistsException`, `NotFoundException`, `AlreadyBorrowedException` und `NotBorrowedException`).

Die Kommunikation mit dem Benutzer soll über eine Eingabemaske ähnlich einer Shell realisiert werden. Zum Registrieren eines neuen Buches könnte z.B. folgender Aufruf dienen:

```
> register Book "Operating Systems"
```

Teil 2:

Als nächstes soll die Bibliothek so erweitert werden, dass Ausleihstationen auf mehreren Rechnern betrieben werden können. Zur Lösung der Aufgabe werden die aus der Tafelübung bekannten Datenströme (Streams), die Netzwerkkommunikation über Sockets und Threads benötigt.

Um auf die Bücher von verschiedenen Rechnern aus zugreifen zu können, soll die Bibliothek in einen Server-Teil und einen Client-Teil aufgeteilt werden. Der Server verwaltet die Datenbank (`SimpleDBImpl`), während die Clients den Zugriff auf die Medien ermöglichen sollen (`LibraryFrontend`).

Die Aufgabe ist zur einfacheren Bearbeitung in folgende Teilaufgaben untergliedert:

a) Packages

Die Klassen und Interfaces sollen für die Aufgabe auf Pakete aufgeteilt werden. Die Aufteilung soll wie folgt geschehen:

- Das Paket `common.mwlibrary` soll alle Interfaces enthalten, die sowohl vom Client als auch vom Server verwendet werden.
- Das Paket <*gruppe*>`.mwlibrary` soll alle Klassen enthalten, die sowohl vom Server als auch von den Clients genutzt werden.
- Das Paket <*gruppe*>`.mwlibrary.server` soll alle Klassen enthalten, welche gebraucht werden um den Server zu implementieren.
- Das Paket <*gruppe*>`.mwlibrary.client` enthält die Klassen, die nur von den Clients genutzt werden.

b) Client/Server

Die Kommunikation zwischen den Clients (LibraryFrontend) und dem Server (SimpleDB) soll über Socket-Verbindungen geschehen. Implementieren Sie hierzu eine Klasse `LibraryServer`, die an einem bestimmten Port Verbindungen entgegen nimmt. Als Portnummer soll Ihre Benutzerkennung aus dem CIP-Pool dienen (diese können Sie mit dem Programm `id` ermitteln).

Über die Verbindung werden Anfragen in der Form von `Command`-Objekten ausgetauscht. Ein `Command`-Objekt stellt folgende Methode zur Verfügung:

`Result perform(SimpleDB)`
: Führt die entsprechende Aktion auf der Datenbank durch und liefert das Ergebnis des Aufrufes als `Result`-Objekt zurück.

Der Server liest vom Socket über einen Objektstream die `Command`-Objekte ein und ruft an diesem die `perform`-Methode auf. Anschließend muss das Ergebnis zum Client zurück geschickt werden. Hinweis: auch eine Exception ist ein Ergebnis.

Verändern Sie die Klasse `LibraryFrontend` nun so, dass die Klasse anstelle von `SimpleDBImpl` ein Objekt der Klasse `SimpleDBProxy` benutzt. Diese Klasse implementiert ebenfalls das Interface `SimpleDB`. Beim Erzeugen einer Instanz soll eine Socket-Verbindung zum Server aufgebaut werden. Über diese Verbindung soll für jeden Methodenaufruf an `SimpleDB` ein entsprechendes `Command`-Objekte zum Server weitergeleitet werden. Wird ein Rückgabewert erwartet, soll der Client blockieren bis die Antwort zur Verfügung steht.

Anmerkung: `Command`-Objekte können in verschiedenen Ausführungen existieren, welche die entsprechenden Aufrufparameter enthalten und die entsprechende Methode an einem `SimpleDB`-Objekt aufrufen. Sie sollen daher je ein `Command`-Objekt für jede Methode der Schnittstelle `SimpleDB` erstellen.

c) Multithreaded Server

Für jede geöffnete Verbindung soll der Server nun jeweils einen neuen Thread erzeugen, welcher Anfragen vom Client entgegen nimmt, die Änderungen an der interne Datenbank vornimmt und das Ergebnis zurücksendet. Achtung Koordinierung notwendig!!

Noch ein Tipp zum Schreiben der Client/Server Anwendung:

- Ein Objektstream überträgt den Zustand eines Objektes nur einmal. Anschließend wird nur noch eine symbolische Referenz übertragen. Um die Zustandsänderung eines Objektes zu übertragen kann entweder jeweils eine Objektkopie (`clone()`) übertragen, oder mit der Methode `reset()` der Objektstrom zurückgesetzt werden.

Bearbeitung: bis zum 12.11.2009/13:45 Uhr
Alle notwendigen Quelldateien und ein Makefile müssen im SVN-Repository eingecheckt sein.
Die Bearbeitung ist in 3er Gruppen möglich.