Übungsaufgabe #2: MWLibrary - Client/Server

27.10.2005

In dieser Aufgabe soll die Bibliothek aus der vorherigen Aufgabe so erweitert werden, dass Ausleihstationen auf mehreren Rechnern betrieben werden können. Kopieren Sie dazu alle Dateien aus Aufgabe 1 in das Verzeichnis `aufgabe2`. 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, während die Clients den Zugriff auf die Medien ermöglichen sollen.

Die Aufgabe ist zur einfacheren Bearbeitung in folgende Teilaufgaben untergliedert:

a) Design

Bevor mit der Implementierung begonnen wird, soll zunächst ein Design der Anwendung erstellt werden, um die anschließende Implementierung zu erleichtern. Es sollten die geplanten Klassen, Interfaces und Aufgaben festgelegt werden, benötigte Threads und die Synchronisation gemeinsamer Datenstrukturen.
Lesen Sie sich zuerst die folgenden Teilaufgaben durch und erstellen sie dann Ihre Lösung. Eine einfache ASCII-Textdatei reicht, bei umfangreicheren Grafiken kann auch ein PDF-Dokument erstellt werden.

b) 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 `<loginname>.mwlibrary` soll alle Klassen enthalten, die sowohl vom Server als auch von den Clients genutzt werden.
- Das Paket `<loginname>.mwlibrary.server` soll alle Klassen enthalten, welche gebraucht werden um den Server zu implementieren.
- Das Paket `<loginname>.mwlibrary.client` enthält die Klassen, die nur von den Clients genutzt werden.

c) Persistente Datenbank

Erweitern sie die Datenbank so das sie ihre Daten persistent in eine Datei speichern kann. Hierzu sind folgende Erweiterungen nötig:

Ergänzen sie den Konstruktor der `SimpleDB`-Klasse um einen Parameter `filename` der festlegt unter welchem Namen die Datei gespeichert wird. Die Implementierung muss testen ob schon eine gleichnamige Datei existiert und gegebenenfalls die enthaltenen Daten laden. Das Laden soll in einer Methode `load`, das Speichern in einer Methode `save` realisiert werden. Nach jeder modifizierenden Operation soll die Datenbank geschrieben werden.

d) 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 die Methode `Result perform(SimpleDB)` zur Verfügung, welche die entsprechende Aktion auf der Datenbank durchführen soll und das Ergebnis des Aufrufes als `Result`-Objekt zurückliefert. 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.

e) 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.

<u>Bearbeitung:</u> bis zum 11.11.2005/20:00 Uhr
Alle Dateien sollen, wie immer, mit dem `abgabe`-Programm abgegeben werden
Die Bearbeitung ist in 2er Gruppen möglich.