

---

## 4 Übungsaufgabe #4: Mobile Objekte

### 4.1 Allgemeines

In dieser Aufgabe wird das bestehende Bibliothekssystem so erweitert, dass sich Medien für Clients transparent zwischen Servern migrieren lassen. Um dies zu erreichen, sollen sie als *Life-Cycle*-Objekte implementiert werden. Darüber hinaus müssen auf Server-Seite zusätzliche Dienstleistungen und -anbieter hinzukommen. Ab sofort können Bibliotheks-Server folgende Dienste zur Verfügung stellen:

1. **Library**: Bereitstellung von Bibliotheksfunktionalität (wie bisher)
2. **Item**: Bereitstellung von Medienfunktionalität (wie bisher)
3. **ItemFactory**: Erzeugung von *Item-Life-Cycle*-Objekten
4. **FactoryFinder**: Zentraler Dienst zum Auffinden von *ItemFactory*-Objekten.

Pro Bibliothekssystem existiert ein Haupt-Server („Primary“), der alle vier Dienste bereitstellt. Daneben können eine beliebige Anzahl an Neben-Servern („Secondaries“) zum Einsatz kommen. Diese bieten ausschließlich die Dienste 2 und 3 an.

### 4.2 Vorbereitungen

Als Ausgangsbasis soll eine modifizierte Version des Systems aus Aufgabe 3 dienen. Folgende Änderungen sind dabei durchzuführen: *ItemServants* werden nicht mehr on-demand im *ItemServantLocator*, sondern bereits bei Erzeugung eines Mediums (*register()*) im *LibraryDBServant* erstellt. Da alle Servants zu jeder Zeit aktiv sind, entfällt die Verwendung der persistenten Datenbank (*ItemDatabase*). Stattdessen führt der *LibraryDBServant* Buch über alle vorhandenen Items (vgl. Vektor in Aufgabe 3, Teil 3.4). Der Cache des *ItemServantLocator* enthält weiterhin alle (= alle aktiven) *ItemServants* des lokalen *ItemPOA*. Da die Aktivierung von Servants entfällt, muss in *preinvoke()* zunächst nur eine Referenz auf den richtigen *ItemServant* zurückgegeben werden.

Unter `/proj/i4mw/pub/aufgabe4` steht eine erweiterte IDL-Beschreibung zur Verfügung. Diese ist im Laufe dieser Übungsaufgabe mindestens an den markierten Stellen anzupassen. Um auch die in 4.3 benötigte *Tie*-Klasse zu erzeugen, muss der *idlj* folgendermaßen aufgerufen werden:

```
idlj -fallTie Library.idl
```

Aufgaben:

→ Anpassung der Klassen *LibraryDBServant* und *ItemServantLocator*

Hinweise:

- Statt den bestehenden Code zu ändern, können im Rahmen dieser Übungsaufgabe auch Unter- bzw. Ersatzklassen für die anzupassenden Klassen implementiert werden.
- Die vom IDL-Compiler erzeugte Klasse *mw.library.ItemContainerPOATie* wird nicht benötigt und kann gelöscht werden.

### 4.3 Objekterzeugung

Bisher wurden Medienobjekte vom *LibraryDBServant* nur lokal erzeugt. Ab sofort soll dies auch entfernt auf anderen Servern möglich sein. Dies erfolgt mittels einer dort bereitgestellten *ItemFactory*, die sich zuvor beim *FactoryFinder* angemeldet (*register()*) hat. Zur Erzeugung eines Objekts sind folgende Schritte notwendig:

1. Beschaffung einer Referenz auf eine passende *ItemFactory* per *find\_factory()* am *FactoryFinder*
2. Aufruf von *create\_object()* an der *ItemFactory*

Der *FactoryFinder* ist ein im Primary aktives *CORBA*-Objekt, dessen Referenz global bekannt ist (z.B. durch den Namensdienst). Die Semantik der Schlüssel mit denen sich die einzelnen *Factory*-Objekte bei ihm anmelden ist frei wählbar (z.B. Host-Namen).

Wie in der Tafelübung erläutert, ist es unter Zuhilfenahme von *ValueTypes* möglich, den *Life-Cycle-Service* plattformunabhängig zu implementieren. Für das Bibliothekssystem bedeutet dies, dass statt den bisher verwendeten *ItemServants* nun Stellvertreter (*ItemPOATies*) als Servants zum Einsatz kommen. Diese verfügen intern über eine Referenz auf eine Implementierung des *Item-ValueType* (→ *ItemContainerImpl*), an die sie alle Aufrufe delegieren.

---

Aufgaben:

- Implementierung des *FactoryFinder*- und des *ItemFactory*-Servant
- Implementierung der Klasse *ItemContainerImpl*
- Implementierung einer *Main*-Methode für den Primary
- Anpassung des *LibraryDBServant*

Hinweise:

- Zu diesem Zeitpunkt sollte mindestens Folgendes funktionieren: Ein Client registriert (mittels *LibraryFrontend*) bei dem auf dem Primary laufenden *LibraryDBServant* ein neues Item. Dieses wird per *ItemFactory* als Life-Cycle-Objekt auf dem Primary erstellt und am *ItemPOA* registriert. Im Anschluss daran ist es über den angepassten *ItemServantLocator* erreichbar.

## 4.4 Objektduplizierung

Als nächstes sollen zusätzlich Secondary-Server eingesetzt werden. Dabei handelt es sich um abgespeckte Varianten des Primary (siehe 4.1), die nur über eine *ItemFactory* und einen *ItemPOA* mit *ItemServantLocator* verfügen. Sie besitzen weder einen *LibraryDBServant* noch einen eigenen *FactoryFinder*.

Um weitere Server einzubeziehen, werden die *LibraryDB*-Schnittstelle und das *LibraryFrontend* um die Methode *void copyItem(String title)* erweitert. Mit ihrer Hilfe ist es möglich, die Kopie eines *Item* auf einem der vorhandenen Secondary-Server zu erstellen. Auf welchem Secondary genau dies erfolgt liegt im Ermessen des *LibraryDBServant* (z.B. Zufallsentscheidung).

Die Erzeugung einer *Item*-Kopie erfolgt (direkt am Objekt selbst) durch den Aufruf der von *LifeCycleObject* geerbten *copy()*-Methode. Diese sorgt in der Folge mittels *createCopyFromValueType()* an einer passenden *ItemFactory* für die Duplizierung. Die *Item*-Kopie wird im Anschluss vom *LibraryDBServant* in die Liste der existierenden Medien aufgenommen und steht somit der gesamten Bibliothek zur Verfügung.

Aufgaben:

- Implementierung einer *Main*-Methode für Secondary-Server
- Implementierung Objektduplizierungsfunktionalität

Hinweise:

- Bevor ein *Item* kopiert werden kann, ist sicherzustellen, dass es gegenwärtig keine Requests bearbeitet. Nur so lässt sich garantieren, dass der Zustand des Objekts korrekt transferiert wird.

## 4.5 Objektmigration

Im letzten Schritt soll es nun außerdem möglich gemacht werden, *Items* zwischen Servern zu verschieben. Hierzu werden die *LibraryDB*-Schnittstelle und das *LibraryFrontend* erneut erweitert. Hinzu kommt die Methode *void clearPrimary()*, die beim Aufruf dem Primary sämtliche auf ihm befindlichen *Items* entzieht und sie auf alle zur Verfügung stehenden Secondary-Server (gleichmäßig) verteilt. Auf diese Weise wird der Primary entlastet während die *Item*-Objekte weiterhin für Clients erreichbar bleiben.

Die Migration von *Items* wird mittels der von *LifeCycleObject* geerbten *move()*-Methode vollzogen. Dabei muss zunächst eine entfernte *Item*-Kopie erzeugt werden. Anschließend ist dafür zu sorgen, dass bei Objektaufrufen am bisherigen Standort eine *ForwardRequest* geworfen wird, die auf den neuen hinweist.

Aufgaben:

- Implementierung der Objektmigrationsfunktionalität

Hinweise:

- Analog zum Kopieren ist auch beim Migrieren von Objekten darauf zu achten, dass ein konsistenter Zustand übertragen wird. Zusätzlich müssen alle Requests, die während des *Item*-Transfers eintreffen, auf das neue Objekt umgeleitet werden.

## 4.6 Abgabe: am 17.12.2009 in der Rechnerübung