

Aufgabenblatt 5

(Zu bearbeiten bis 27.11.)

Vorlesung Verteilte Systeme Wintersemester 2025/26

Pflichtaufgabe 1: Programmierung: Uhrensynchronisation **Abgabe über moodle!**

In dieser Aufgabe sollen Sie mit Hilfe des Verteilte-Systeme-Simulators eine Uhrensynchronisation nach Cristian (siehe Abschnitt 6.1 der Vorlesungsfolien) programmieren.

Falls Sie es noch nicht getan haben, bearbeiten Sie bitte als Vorbereitung zunächst die Aufgabe 1 des Aufgabenblatts 4.

Bearbeiten Sie anschließend die Aufgabe „Clock synchronisation¹“ im Wiki (Zugriff nur aus dem Uni-VPN!).

Aufgabe 2: Programmierung: Factories

Das Entwurfsmuster „Factory“ sieht vor, das Erzeugen von Objekten an ein anderes Objekt zu delegieren, so daß dieses z.B. entscheiden kann, welche konkrete Implementierung einer abstrakten Klasse gewählt wird. Bei Java-RMI kann man das Entwurfsmuster nutzen, um eine entfernte Objekterzeugung zu realisieren, die mit `new` nicht möglich ist.

In der Vorlesung wurde bereits das Beispiel einer Bank genannt, die Konto-Objekte erzeugen kann. Dabei ist es weder notwendig noch sinnvoll, alle Konto-Objekte bei der RMI-Registry zu registrieren. Stattdessen wird nur das Bank-Objekt registriert, das die Konten verwaltet.

Im Archiv [u05Files.zip²](#) auf der Vorlesungswebseite finden Sie ein Code-Gerüst für dieses Szenario, das u.a. einen Client enthält, der über die folgenden Schnittstellen mit dem Bank-Server kommuniziert:

```
public interface Bank extends Remote
{
    Account newAccount(int accountNo) throws RemoteException; // neues Konto
    Account getAccount(int accountNo) throws RemoteException;
    List<Account> getAllAccounts() throws RemoteException;    // alle Konten
}
public interface Account extends Remote
{
    int getAccountNo() throws RemoteException;
    void deposit(double amount) throws RemoteException;
    void withdraw(double amount) throws RemoteException;
    String accountStatement() throws RemoteException;    // KontoNr und Stand
}
```

Ergänzen Sie die Implementierung der Klasse `BankImpl` und testen Sie Ihr Programm mit dem gegebenen Client. Die Ausgabe sollte sein:

```
Konto 123456789: 100000.0
Konto 234567890: 50000.0
Konto 345678901: 1020.0
```

¹<https://git.bs.informatik.uni-siegen.de/dsbox/exercises/wiki/2-clock-sync>

²<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u05Files.zip>

```
Konto 123456789: 95000.0
Konto 234567890: 65000.0
Konto 345678901: 1050.0
```

Was genau liefert die Methode `getAllAccounts()` an den Client zurück, d.h., was sendet hier der Server an den Client?

Pflichtaufgabe 3: Threadsicherheit

Abgabe über moodle!

Da Java RMI keine Angaben darüber macht, wie viele Threads serverseitig für Methodenaufrufe bereitgestellt werden, ist eine korrekte Synchronisation erforderlich. Wo und wie kann diese bei folgendem Beispiel umgesetzt werden?

Interface

```
public interface Number extends Remote {
    public void advanceBy(int add) throws RemoteException;
    public int getValue() throws RemoteException;
}
```

Client-Klasse

```
public class Client {
    public static void main(String[] args) {
        try {
            Number num = (Number)Naming.lookup(...);
            num.advanceBy(10);
            System.out.println(num.getValue());
            ...
        }
    }
}
```

Server-Klasse

```
public class Server extends UnicastRemoteObject implements Number {
    private int number;

    public Number() throws RemoteException {}

    public void advanceBy(int add) {
        number = number + add;
    }

    public int getValue() {
        return(number);
    }
    ...
}
```

Aufgabe 4: Programmierung: Client-Callback - Fortschrittsanzeige

Bei Client-Callbacks handelt es sich um Remote-Aufrufe, die der Server an den Client stellt. Dies wird häufig zur Benutzerinteraktion durchgeführt, z.B. um den Fortschritt eines Programms anzuzeigen. Erstellen Sie auf Basis des im Archiv [u05Files.zip](http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u05Files.zip)³ auf der Vorlesungswebseite vorgegebenen Codes ein Programm, das auf dem Server eine Berechnung auslöst und clientseitig eine Funktion `notify(int percent)` zur Ausgabe des Berechnungsfortschritts enthält. Simulieren Sie dazu auf dem Server eine Berechnung innerhalb der Methode `compute(ProgressNotifier pn)`, indem Sie innerhalb einer Schleife von 0 bis 100 den aktuellen Thread 500 Millisekunden warten lassen (Sie benötigen dazu die Klasse `java.lang.Thread`). Rufen Sie während der Berechnung regelmäßig die Methode `notify()` für die Ausgabe auf.

³<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u05Files.zip>

Bedenken Sie, daß für Client-Callbacks der Client selbst ein Remote-Objekt sein (oder zumindest beinhalten) muß. Wie verhält sich der Client am Ende, wenn Sie Ihre Klasse einfach von `UnicastRemoteObject` erben lassen? Warum?

Zur Lösung des Problems sollten Sie das Callback-Objekt mit Hilfe der Methode

```
UnicastRemoteObject.exportObject(Remote obj, int port)
```

exportieren (wobei `port = 0` gesetzt werden sollte) und am Ende die Methode

```
UnicastRemoteObject.unexportObject(Remote obj, boolean force)
```

aufrufen (siehe Java-Dokumentation).

Aufgabe 5: Namensdienste

Die wichtigste Operation, die ein Namensdienst unterstützt, ist die Auflösung von Namen, d.h. Einheiten nach einem vorgegebenen Namen zu suchen. Diskutieren Sie folgende Fragestellungen zu diesem Thema.

- a) Nennen Sie ein Beispiel, wo eine Adresse einer Einheit E weiter in eine andere Adresse aufgelöst werden muss, um wirklich auf E zugreifen zu können.
- b) Nennen Sie Beispiele für echte IDs (allgemein).
- c) Würden Sie eine URL wie z.B. `http://www.acme.org/index.html` als positionsunabhängig betrachten? Wie sieht es bei der URL `http://www.acme.nl/index.html` aus?
- d) Diskutieren Sie die Probleme, die durch die Verwendung von Aliasen in einem Namensdienst entstehen und zeigen Sie auf, ob und wie diese gelöst werden können.

Aufgabe 6: Namensdienste

Namensdienste sind aus Skalierbarkeitsgründen meist hierarchisch organisiert. Erklären Sie die prinzipielle Funktionsweise eines solchen hierarchischen Dienstes. Wie unterscheiden sich iterative und rekursive Namensauflösung?