

## Excercise Sheet 5

(To be processed until 27.11.)

## Lecture Distributed Systems Winter Term 2025/26

### Compulsory Exercise 1: Programming: Clock synchronization Submit via moodle!

In this exercise, you will use the distributed systems simulator to program a clock synchronization according to Cristian (see Section 6.1 of the lecture slides).

If you have not already done so, please complete Exercise 1 on Exercise Sheet 4 as preparation.

Then complete the task „Clock synchronization<sup>1</sup>“ in the wiki (accessible only from within the Uni-VPN!).

### Exercise 2: Programming: Factories

The design pattern “Factory” provides for delegating the creation of objects to another object, so that the latter can decide, for example, which concrete implementation of an abstract class is chosen. With Java-RMI one can use the design pattern to realize a remote object creation, which is not possible with `new`.

In the lecture already the example of a bank was mentioned, which can create account objects. It is neither necessary nor useful to register all account objects with the RMI registry. Instead, only the bank object that manages the accounts is registered.

In the archive [u05eFiles.zip](http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u05eFiles.zip)<sup>2</sup> on the lecture’s web page you will find a code framework for this scenario, which contains, among other things, a client that communicates with the bank server via the following interfaces:

```
public interface Bank extends Remote
{
    Account newAccount(int accountNo) throws RemoteException; // new account
    Account getAccount(int accountNo) throws RemoteException;
    List<Account> getAllAccounts() throws RemoteException; // all accounts
}
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; // account number and balance
}
```

Supplement the implementation of the class `BankImpl` and test your program with the given client. The output should be:

```
Konto 123456789: 100000.0
Konto 234567890: 50000.0
Konto 345678901: 1020.0
Konto 123456789: 95000.0
```

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

<sup>2</sup><http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u05eFiles.zip>

```
Konto 234567890: 65000.0
Konto 345678901: 1050.0
```

What exactly does the `getAllAccounts()` method return to the client, i.e. what does the server send to the client?

### Compulsory Exercise 3: Thread safety

Submit via moodle!

Since Java RMI does not specify how many threads are provided on the server side for method calls, correct synchronization is required. Where and how can this be implemented in the following example?

Interface

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

Client class

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

Server Class

```
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);
    }

    ...
}
```

### Exercise 4: Programming: Client callback - progress bar

Client callbacks are remote calls that the server makes to the client. This is often done for user interaction, for example, to indicate the progress of a program. Based on the given code in the archive [u05eFiles.zip](http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u05eFiles.zip)<sup>3</sup> on the lecture's web page, create a program that triggers a calculation on the server and contains a function `notify(int percent)` on the client side to output the calculation progress. To do this, simulate a calculation on the server within the method `compute(ProgressNotifier pn)` by making the current thread wait 500 milliseconds within a loop from 0 to 100 (you need the class `java.lang.Thread` for this). During the calculation, regularly call the `notify()` method for the output.

Note that for client callbacks, the client itself must be (or at least contain) a remote object. How does the client behave at its end if you simply inherit your class from `UnicastRemoteObject`? Why?

---

<sup>3</sup><http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u05eFiles.zip>

To solve the problem, you should export the callback object using the method

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

(where you should set `port = 0`) and at the end call the method

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

(see Java documentation).

### Exercise 5: Name services

The most important operation that a naming service supports is the resolution of names, that is, searching for units by a given name. Discuss the following questions on this topic.

- a) Name an example where an address of a unit E must be further resolved to another address to actually be able to access E.
- b) Give examples of real IDs (general).
- c) Would you regard a URL such as `http://www.acme.org/index.html` as position independent? What about the URL `http://www.acme.nl/index.html`?
- d) Discuss the problems that arise from the use of aliases in a naming service and show whether and how these can be solved.

### Exercise 6: Name services

For reasons of scalability, naming services are usually organized hierarchically. Explain the basic functionality of such a hierarchical service. How do iterative and recursive name resolution differ?