

## Excercise Sheet 4

(To be processed until 20.11.)

## Lecture Distributed Systems Winter Term 2025/26

### Exercise 1: Simulator for Distributed Systems

In the course of the exercise, you will solve several tasks with the help of a simple simulator for distributed systems, **including some mandatory tasks.**

First, download the simulator from [the research group's Git server](#)<sup>1</sup> (accessible only from within the Uni-VPN!) and install it on your PC or laptop<sup>2</sup>. Installation instructions can also be found on the Git server.

Then take a look at the [brief description of the simulator](#)<sup>3</sup> in the wiki.

For practical training, complete the [Hello World](#)<sup>4</sup> task in the wiki.

### Exercise 2: Programming: Java-RMI - Follow-up example **Mandatory exercise, submit via moodle!**

Create a distributed application that includes a remote object with a method that adds two integers. Proceed as follows: Create

- first the interface `Numbers`, which defines the method `add()`,
- then a class `NumbersImpl` implementing `Numbers`,
- a server application in the class `Server`,
- a client application that calls the remote method in the server and returns the result of the addition.

Try to start the RMI registry directly from your program code, so that no invocation of `rmiregistry` is necessary on the command line. For that purpose, use the two classes `java.rmi.registry.LocateRegistry` and `java.rmi.registry.Registry` (see [Java API documentation](#)).

### Exercise 3: Programming: Generic proxy objects

Since JDK 1.5 the use of the RMI compiler `rmic` to create the client stub classes is no longer necessary when using Java RMI. Instead, the necessary stub classes are created at runtime using the class `java.lang.reflect.Proxy` from the Java Reflection framework. To learn more about this mechanism, you should create a generic proxy for the Java class `CalculatorImpl`, which you can find in the archive [u04eFiles.zip](#)<sup>5</sup> on the lecture's web page, which delegates all method calls to the actual object and logs them additionally.

First inform yourself about the following methods:

<sup>1</sup><https://git.bs.informatik.uni-siegen.de/dsbox/simulator/releases>

<sup>2</sup>Unfortunately, only Linux and Windows are currently supported. If you do not have a suitable computer, you can also use the computers in lab H-A 4111. In this case, please submit a [chip key application form](#) so that you have access to the room. Please submit the application to the secretary's office H-B 8403 or simply hand it in after the lecture or exercise.

<sup>3</sup><https://git.bs.informatik.uni-siegen.de/dsbox/exercises/wiki/0-introduction>

<sup>4</sup><https://git.bs.informatik.uni-siegen.de/dsbox/exercises/wiki/1-hello-world>

<sup>5</sup><http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u04eFiles.zip>

- `java.lang.reflect.Proxy.newProxyInstance()` – Dynamically creates a proxy object
- `java.lang.reflect.InvocationHandler.invoke()` – All method calls on the proxy object are delegated to this method
- `java.lang.Object.getClass()` – Returns the class of an object
- `java.lang.reflect.Class.getClassLoader()` – Returns the class loader for a class
- `java.lang.reflect.Class.getInterface()` – Returns all interfaces the class implements

Then add the `CalculatorProxy` class that implements the `InvocationHandler` interface. The method `invoke()` should output the called method and its arguments, delegate the call to the “real” object and then output the return value. In the `CalculatorExample` class, replace the “real” `Calculator` object (reference in the `calc` variable) with the proxy object and test your program.

#### Exercise 4: Parameter passing

Consider the procedure `incr` that uses two integer parameters. This procedure adds a 1 to each parameter.

Suppose it is called twice with the same variable, for example `incr(i, i)`. If `i` is initially 0, what is its final value, if *call-by-reference* is used? What value would `i` have if *call-by-value* is used? What about *call-by-copy/result*?

#### Exercise 5: Transparency of Java RMI

**Mandatory exercise, submit via moodle!**

An inexperienced programmer has been given the task of offloading the sorting of a list that was previously carried out locally in a client to a server. Since the interface had to remain absolutely unchanged (parts of the client are not available in source code), the programmer implemented a wrapper class that searches for the server object using the name service and handles the exceptions that occur. A (simplified) version of its code can be found in the archive [u04eFiles.zip](#)<sup>6</sup> on the lecture’s web page. `LocalSorter` (in `Client.java`) is the original class, `RemoteSorter` is the new wrapper class. The server code consists of the remote interface `SortServer` and the implementation class `Server`. To test the compatibility between the classes `RemoteSorter` and `LocalSorter`, the method `testSorter()` in `Client.java` is called with one instance of each class. As the output of the client shows, sorting using the server does not seem to work.

“Repair” the remote implementation of sorting! Note that the `RemoteSorter` interface must not be changed. What is the problem? How can it be solved?

---

<sup>6</sup><http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u04eFiles.zip>