

Aufgabenblatt 4

(Zu bearbeiten bis 20.11.)

Vorlesung Verteilte Systeme Wintersemester 2025/26

Aufgabe 1: Simulator für Verteilte Systeme

Im weiteren Verlauf der Übung werden Sie einige Aufgaben mit Hilfe eines einfachen Simulators für verteilte Systeme lösen, **darunter auch einige Pflichtaufgaben**.

Laden Sie sich zunächst den Simulator vom [Git-Server des Lehrstuhls](#)¹ herunter (Zugriff nur aus dem Uni-VPN!) und installieren Sie ihn auf Ihrem PC oder Laptop². Eine Installationsanleitung finden ebenfalls auf dem Git-Server.

Sehen Sie sich dann die [Kurzbeschreibung des Simulators](#)³ im Wiki an.

Zur praktischen Einarbeitung bearbeiten Sie anschließend die [Hello World](#)⁴ Aufgabe im Wiki.

Aufgabe 2: Programmierung: Java-RMI - Folgebeispiel

Pflichtaufgabe, Abgabe über moodle!

Erstellen Sie eine verteilte Anwendung, die ein entferntes Objekt mit einer Methode beinhaltet, welche zwei Ganzzahlen multipliziert. Gehen Sie wie folgt vor: Erstellen Sie

- zunächst die Schnittstelle `Numbers`, die die Methode `multiply()` definiert,
- dann eine Klasse `NumbersImpl`, die `Numbers` implementiert,
- eine Server-Anwendung in der Klasse `Server`,
- eine Client-Anwendung, die die entfernte Methode im Server aufruft und das Ergebnis der Multiplikation ausgibt.

Versuchen Sie, die RMI-Registry direkt im Programmcode des Servers zu starten, so dass auf der Kommandozeile kein Aufruf von `rmiregistry` notwendig ist. Verwenden Sie die Klassen `java.rmi.registry.LocateRegistry` und `java.rmi.registry.Registry` (siehe [Java API Dokumentation](#)).

Aufgabe 3: Programmierung: Generische Proxy-Objekte

Seit JDK 1.5 kann bei der Nutzung von Java RMI auf die Verwendung des RMI-Compilers `rmic` zur Erzeugung der Client-Stub-Klassen verzichtet werden. Stattdessen werden die notwendigen Stub-Klassen zur Laufzeit mit Hilfe von Java Reflection und der Klasse `java.lang.reflect.Proxy` gearbeitet. Um diesen Mechanismus näher kennenzulernen, sollen Sie für die Java-Klasse `CalculatorImpl`, die Sie im Archiv [u04Files.zip](#)⁵ auf der Vorlesungswebseite finden, einen generischen Proxy erzeugen, der alle Methodenaufrufe an das eigentliche Objekt delegiert und sie zusätzlich protokolliert.

Informieren Sie sich zunächst über folgende Methoden:

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

²Im Moment werden leider nur Linux und Windows unterstützt. Sollten Sie keinen geeigneten Computer besitzen, können Sie auch die Rechner im Labor H-A 4111 nutzen. In diesem Fall stellen Sie bitte einen [Chipschlüsselantrag](#), damit Sie Zugang zu dem Raum haben. Den Antrag geben Sie bitte im Sekretariat H-B 8403 oder einfach nach der Vorlesung oder Übung ab.

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

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

⁵<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u04Files.zip>

- `java.lang.reflect.Proxy.newProxyInstance()` – Erzeugt dynamisch ein Proxy-Objekt
- `java.lang.reflect.InvocationHandler.invoke()` – An diese Methode werden alle Methodenauf-rufe auf dem Proxy-Objekt delegiert
- `java.lang.Object.getClass()` – Liefert die Klasse eines Objekts
- `java.lang.reflect.Class.getClassLoader()` – Liefert den Klassenlader zu einer Klasse
- `java.lang.reflect.Class.getInterface()` – Liefert alle Schnittstellen, die die Klasse implementiert

Ergänzen Sie dann die Klasse `CalculatorProxy`, die das Interface `InvocationHandler` implementiert. Die Me-thode `invoke()` soll die aufgerufene Methode und deren Argumente ausgeben, den Aufruf dann an das „echte“ Objekt delegieren und anschließend den Rückgabewert ausgeben. Ersetzen Sie in dann der Klasse `CalculatorExample` das „echte“ `Calculator`-Objekt (Referenz in der Variable `calc`) durch das Proxy-Objekt und testen Sie Ihr Programm.

Aufgabe 4: Parameterübergabe

Betrachten Sie die Prozedur `incr`, die zwei ganzzahlige Parameter verwendet. Diese Prozedur addiert zu jedem Paramter eine 1.

Angenommen, sie wird mit der selben Variable zweimal aufgerufen, beispielsweise `incr(i, i)`. Wenn `i` anfänglich 0 ist, welchen Wert hat es, anschließend, wenn *call-by-reference* verwendet wird? Welchen Wert hätte `i`, wenn *call-by-value* genutzt wird? Wie sieht es bei *call-by-copy/result* aus?

Aufgabe 5: Transparenz von Java RMI

Pflichtaufgabe, Abgabe über moodle!

Ein unerfahrener Programmierer hat die Aufgabe bekommen, das Sortieren einer Liste, das bisher lokal in einem Cli-ent vorgenommen wurde, ein einen Server auszulagern. Da dabei die Schnittstelle absolut unverändert bleiben mus-te (Teile des Clients liegen nicht im Quelltext vor), hat der Programmierer eine Wrapper-Klasse implementiert, die das Server-Objekt beim Namensdienst aufsucht und die auftretenden Exceptions behandelt. Eine (vereinfachte) Versi-on seines Codes finden Sie im Archiv [u04Files.zip](http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u04Files.zip)⁶ auf der Vorlesungswebseite. Dabei ist `LocalSorter` (in `Client.java`) die Originalklasse, `RemoteSorter` die neue Wrapper-Klasse. Der Servercode besteht aus der Remote-Schnittstelle `SortServer` und der Implementierungsklasse `Server`. Um die Kompatibilität zwischen den Klassen `RemoteSorter` und `LocalSorter` zu testen, wird die Methode `testSorter()` in `Client.java` jeweils mit ei-ner Instanz der beiden Klassen aufgerufen. Wie die Ausgabe des Clients zeigt, scheint das Sortieren mit Hilfe des Servers jedoch nicht zu funktionieren.

„Reparieren“ Sie die Remote-Implementierung des Sortierens! Beachten Sie, daß die Schnittstelle von `RemoteSorter` nicht verändert werden darf. Wo liegt das Problem? Wie kann es gelöst werden?

⁶<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u04Files.zip>