

**Laborprotokoll**

**Verteilte Objekte mit RMI**

**SYT Labor**

**4CHITT 2015/16**

**Gruppe B**

**Moritz Mühlehner**

|  |  |
| --- | --- |
|  | **Version 1.0** |
| **Note:** | **Begonnen am 11. März 2016** |
| **Betreuer: Borko Michael** | **Beendet am 17. März 2016** |

Inhaltsverzeichnis

[1 Einführung 3](#_Toc446016426)

[1.1 Ziele 3](#_Toc446016427)

[1.2 Voraussetzungen 3](#_Toc446016428)

[1.3 Aufgabenstellung 3](#_Toc446016429)

[1.4 Quellen 3](#_Toc446016430)

[2 Ergebnisse 4](#_Toc446016431)

[Java Policy File 4](#_Toc446016432)

[2.1 RMI Tutorial 4](#_Toc446016433)

[Server 5](#_Toc446016434)

[Client 5](#_Toc446016435)

[Aufgetretene Probleme 6](#_Toc446016436)

[2.2 RMI Command Pattern 6](#_Toc446016437)

[3 Aufwand 9](#_Toc446016438)

[3.1 Geschätzter Aufwand 9](#_Toc446016439)

[3.2 Tatsächlicher Aufwand 9](#_Toc446016440)

# Einführung

Verteilte Objekte haben bestimmte Grunderfordernisse, die mittels implementierten Middlewares leicht verwendet werden können. Das Verständnis hinter diesen Mechanismen ist aber notwendig, um funktionale Anforderungen entsprechend sicher und stabil implementieren zu können.

## Ziele

Diese Übung gibt eine einfache Einführung in die Verwendung von verteilten Objekten mittels Java RMI. Es wird speziell Augenmerk auf die Referenzverwaltung sowie Serialisierung von Objekten gelegt. Es soll dabei eine einfache verteilte Applikation in Java implementiert werden.

## Voraussetzungen

* Grundlagen Java und Software-Tests
* Grundlagen zu verteilten Systemen und Netzwerkverbindungen
* Grundlegendes Verständnis von nebenläufigen Prozessen

## Aufgabenstellung

Folgen Sie dem offiziellen Java-RMI Tutorial, um eine einfache Implementierung des PI-Calculators zu realisieren. Beachten Sie dabei die notwendigen Schritte der Sicherheitseinstellungen (SecurityManager) sowie die Verwendung des RemoteInterfaces und der RemoteException.

Implementieren Sie ein Command-Pattern [2] mittels RMI und übertragen Sie die Aufgaben/Berechnungen an den Server. Sie können am Client entscheiden, welche Aufgaben der Server übernehmen soll. Die Erweiterung dieser Aufgabe wäre ein Callback-Interface auf der Client-Seite, die nach Beendigung der Aufgabe eine entsprechende Rückmeldung an den Client zurück senden soll. Somit hat der Client auch ein RemoteObject, welches aber nicht in der Registry eingetragen wird sondern beim Aufruf mittels Referenz an den Server übergeben wird.

## Quellen

[1] "The Java Tutorials - Trail RMI"; online: http://docs.oracle.com/javase/tutorial/rmi/  
[2] "Command Pattern"; Vince Huston; online: http://vincehuston.org/dp/command.html  
[3] "Beispiel Konstrukt für Command Pattern mit Java RMI"; Michael Borko; online: https://github.com/mborko/code-examples/tree/master/java/rmiCommandPattern

# Ergebnisse

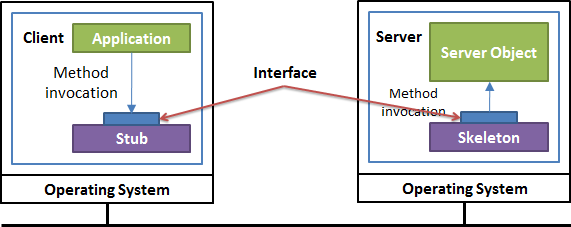


Abbildung 1 Client-Server Kommunikation

### Java Policy File

Das java.policy File gibt die Berechtigungen für jedes Java-Programm an. Für diese Aufgabe ist es wichtig die Berechtigungen im java.policy File für das entsprechende Verzeichnis zu ändern. Dazu muss folgendes im polify File eingetragen werden:



Das File findet man Windows (normalerweise) unter folgendem Pfad:  
C:\Program Files\Java\jdk1.8.0\_74\jre\lib\security\java.policy

## RMI Tutorial

Ziel dieser Aufgabe ist es, dass der Server eine Aufgabe für des Client ausführt. Die Aufgabe ist Pi zu berechnen.   
Das Ganze ist folgendermaßen aufgebaut:

Es gibt zwei Interfaces, Task und Compute. Das Interface Task ist generisch und beinhaltet eine Methode execute mit generischem Rückgabewert.  


Das Compute Interface erweitert java.rmi.remote und beinhaltet die Methode executeTask welche einen Task als Parameter bekommt und ebenfalls einen generischen Rückgabewert hat.



Beide Interfaces sind sowohl dem Client als auch dem Server bekannt.

### Server

Der Server beinhaltet eine einzige Klasse namens ComputeEngine.  
ComputeEngine implementiert Compute und muss daher eine Methode executeTask beinhalten. Die executeTask Methode ruft wiederum die Methode execute des Interfaces Task auf. Da die beiden Methoden generisch sind, kann jedes Objekt, dessen Klasse Task implementiert, an diese Methode als Parameter übergeben werden.

In der main Methode wird als erstes überprüft ob ein SecurityManager vorhanden ist, ist das nicht der Fall wird ein neuer erstellt. Der SecurityManager überprüft die Sicherheit gemäß dem **java.policy** File.

**Wichtig:** Das java.policy File muss geändert werden (Siehe: [Java Policy File](#_Java_Policy_File))

Anschließend wird ein neues ComputeEngine Objekt und ein stub dieses Objektes erzeugt. (Über den der Client dann auf den Server zugreifen kann)



Danach wird eine neue Registry erstellt und der stub in der Registry mit einem Namen vermerkt, sodass der Client später über den Namen an den stub kommt.  


Ist das ganze erfolgreich wird ein „ComputeEngine bound“ ausgegeben.

### Client

Der Client hat zwei Klassen, Pi und ComputePi.

Pi implementiert Task und Serializable und ist daher für die Übertragung an den Server geeignet. Pi bekommt im Konstruktor einen Integer welcher die Genauigkeit für die Berechnung von Pi angibt. Da die Klasse das Interface Task implementiert, beinhaltet sie auch eine Methode execute, welche den berechneten Wert für Pi als BigDecimal zurückgibt.

ComputePi beinhaltet nur die main Methode. Wie auch schon beim Server wird hier als erstes überprüft ob ein SecurityManager vorhanden ist. Ist keiner vorhanden wird ein neuer erzeugt. Danach holt man sich die Registry des Servers und speichert diese. Als Parameter muss die Adresse des Servers angegeben werden. (Hier: localhost)

Über die Registry des Servers bekommt man nun das Compute Objekt des Servers, welches in der Registry mit einem bestimmten Namen „gebindet“ wurde.

Über dieses Objekt kann man durch die Methode executeTask, der man ein Objekt der Klasse Pi als Parameter übergibt (da diese ja Task implementiert), Pi auf dem Server berechnen lassen.



Zuletzt wird der berechnete Wert von Pi noch durch ein System.out.println(pi) ausgegeben.

### Aufgetretene Probleme

Da auf meinem Gerät mehrere Java Versionen installiert sind, gab es beim Erzeugen und Abfragen der Registry Probleme. Diese Probleme wurden durch die Verwendung des Standard Ports 1099 behoben.

## RMI Command Pattern

Der Aufbau dieser Aufgabe ist ähnlich wie bei dem RMI Tutorial.  
Der Client bekommt ein Remote Objekt über die Registry des Servers.



Über dieses Objekt ruft der Client die Methode doSomething auf welcher ein Command als Parameter übergeben wird. doSomething ist eine Methode der Klasse ServerService welche doSomethingService implementiert. (doSomethingService gibt diese Methode vor) doSomething führt die Methode execute auf die übergebene Command Referenz aus.

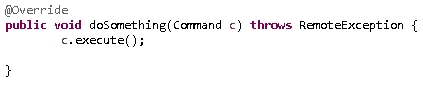


Abbildung 2 Implementierung der doSomething Methode in ServerService



Abbildung 3 doSomething wird von Client über Remote Objekt aufgerufen

Command ist ein Interface und beinhaltet die Methode execute. Es gibt eine Klasse CalculationCommand welche dieses Interface implementiert und daher der Methode doSomething als Parameter übergeben werden kann.

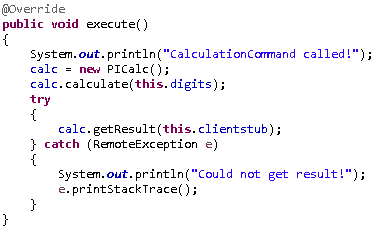


Abbildung 4 execute Methode der Klasse CalculationCommand

Der Konstruktor von CalculationCommand bekommt als Parameter die Anzahl an Nachkommastellen die Pi haben soll und eine Referenz auf den stub des Clients, sodass der Server den berechneten Wert später an den Client zurückliefern kann. (Callback)

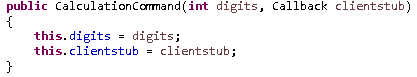


Abbildung 5 Konstruktor von CalculationCommand



Abbildung 6 Erstellen des Client-Stub für Callback  
  


Abbildung 7 Erzeugen eines CalculationCommand Objekts

In der execute Methode (siehe Abbildung 4) der CalculationCommand Klasse wird Pi durch aufrufen einer Methode der PICalc Klasse berechnet und anschließend durch aufrufen der getResult Methode der Klasse PICalc, welcher der Client-Stub als Parameter übergeben wird, auf dem Client ausgegeben. (getResult ruft eine weitere Methode get, aus der ClientCallback Klasse, auf welche dann Pi auf dem Client ausgibt)

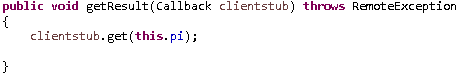


Abbildung 8 getResult Methode der Klasse PICalc



Abbildung 9 get Methode des Client

# Aufwand

## Geschätzter Aufwand

Der geschätzte Aufwand ist wie folgt aufgeschlüsselt:

|  |  |  |
| --- | --- | --- |
| **Durchgeführte Arbeit** | **Datum** | **geschätzter Aufwand** |
| RMI Tutorial | 11.03.2016 | 2 h |
| RMI Command Pattern | 11.03.2016-17.03.2016 | 4 h |
| Protokoll | 17.03.2016 | 2 h |
| **Gesamt** | **11. - 17.03.2016** | **8 h** |

## Tatsächlicher Aufwand

Der tatsächliche Aufwand ist wie folgt aufgeschlüsselt:

|  |  |  |
| --- | --- | --- |
| **Durchgeführte Arbeit** | **Datum** | **tatsächlicher Aufwand** |
| RMI Tutorial | 11.03.2016 | 4 h |
| RMI Command Pattern | 17.03.2016 | 3 h |
| Protokoll | 17.03.2016 | 3 h |
| **Gesamt** | **11. – 17.03.2016** | **10 h** |