# PI-Calculator

## Schuljahr 2014/15

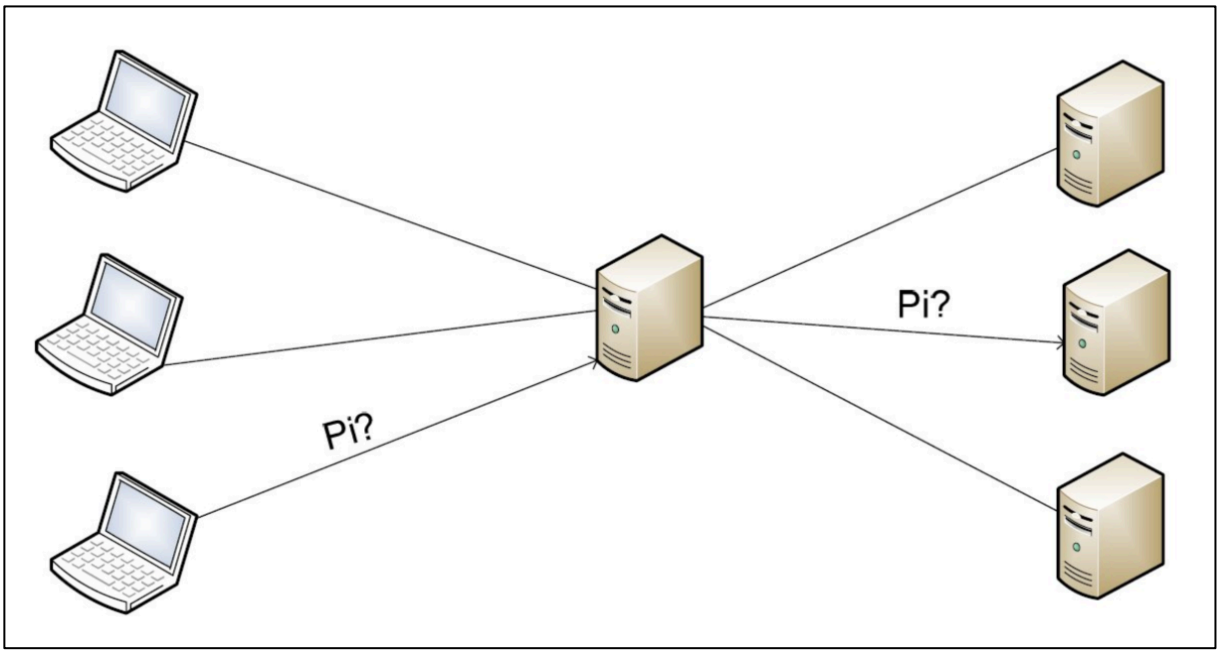
Christoph Hackenberger
Burkhard Hampl

tgm
Die Schule der Technik

## Inhaltsverzeichnis

1. **Aufgabenstellung** ..... 3
2. **Zeitabschätzung** ..... 5
3. **Requirements Analyse** ..... 5
4. **Designüberlegung** ..... 5
5. **Lessions Learned** ..... 5
6. **Testprotokoll** ..... 5
7. **Quellen** ..... 6

# 1. Aufgabenstellung




Als Dienst soll hier die beliebig genaue Bestimmung von pi betrachtet werden. Der Dienst stellt folgendes Interface bereit:

```
1  // Calculator.java
2  public interface Calculator {
3          public BigDecimal pi (int anzahl_nachkommastellen);
4  }
```

Ihre Aufgabe ist es nun, zunächst mittels Java-RMI die direkte Kommunikation zwischen Klient und Dienst zu ermöglichen und in einem zweiten Schritt den Balancierer zu implementieren und zwischen Klient(en) und Dienst(e) zu schalten. Gehen Sie dazu folgendermassen vor:

1. Ändern Sie Calculator und CalculatorImpl so, dass sie über Java-RMI von aussen zugreifbar sind. Entwicklen Sie ein Serverprogramm, das eine CalculatorImpl-Instanz erzeugt und beim RMI-Namensdienst registriert. Entwicklen Sie ein Klientenprogramm, das eine Referenz auf das Calculator-Objekt beim Namensdienst erfragt und damit pi bestimmt. Testen Sie die neu entwickelten Komponenten.

2. Implementieren Sie nun den Balancierer, indem Sie eine Klasse CalculatorBalancer von Calculator ableiten und die Methode pi() entsprechend implementieren. Dadurch verhält sich der Balancierer aus Sicht der Klienten genauso wie der Server, d.h. das Klientenprogramm muss nicht verändert werden. Entwickeln Sie ein Balanciererprogramm, das eine CalculatorBalancer-Instanz erzeugt und unter dem vom Klienten erwarteten Namen beim Namensdienst registriert. Hier ein paar Details und Hinweise:

- Da mehrere Serverprogramme gleichzeitig gestartet werden, sollten Sie das Serverprogramm so erweitern, dass man beim Start auf der Kommandozeile den Namen angeben kann, unter dem das CalculatorImpl-Objekt beim Namensdienst registriert wird. dieses nun seine exportierte Instanz an den Balancierer übergibt, ohne es in die Registry zu schreiben. Verwenden Sie dabei ein eigenes Interface des Balancers, welches in die Registry gebinded wird, um den Servern das Anmelden zu ermöglichen.

- Das Balancierer-Programm sollte nun den Namensdienst in festgelegten Abständen abfragen um herauszufinden, ob neue Server Implementierungen zur Verfügung stehen.

- Java-RMI verwendet intern mehrere Threads, um gleichzeitig eintreffende Methodenaufrufe parallel abarbeiten zu können. Das ist einerseits von Vorteil, da der Balancierer dadurch mehrere eintreffende Aufrufe parallel bearbeiten kann, andererseits müssen dadurch im Balancierer änderbare Objekte durch Verwendung von synchronized vor dem gleichzeitigen Zugriff in mehreren Threads geschützt werden.

- Beachten Sie, dass nach dem Starten eines Servers eine gewisse Zeit vergeht, bis der Server das CalculatorImpl-Objekt erzeugt und beim Namensdienst registriert hat sich beim Balancer meldet. D.h. Sie müssen im Balancierer zwischen Start eines Servers und Abfragen des Namensdienstes einige Sekunden warten.

Testen Sie das entwickelte System, indem Sie den Balancierer mit verschiedenen Serverpoolgrössen starten und mehrere Klienten gleichzeitig Anfragen stellen lassen. Wählen Sie die Anzahl der Iterationen bei der Berechung von pi entsprechend gross, sodass eine Anfrage lang genug dauert um feststellen zu können, dass der Balancierer tatsächlich mehrere Anfragen parallel bearbeitet.

## 2. Zeitabschätzung

| | Geplant | Aktuell |
|---|---|---|
| **Design** | 3:00 | 3:00 |
| **Implementierung** | 2:30 | 2:00 |
| **Testen** | 1:30 | 2:00 |
| **Dokumentation** | 2:00 | 2:30 |
| **Summe** | 9:00 | 9:30 |

## 3. Requirements Analyse

- Lokale Implementierung (CLI Arguments etc.)
- RMI Implementierung
- Client muss eine richtige Antwort für PI direkt vom CalculationServer bekommen
- Client muss eine richtige Antwort für PI über den Balancer bekommen
- Balancer muss Last an Server verteilen
- Alles muss sowohl als Localhost als auch über das Netzwerk funktionieren

## 4. Designüberlegung

UML: siehe Pi_Calculator_UML.pdf

## 5. Lessions Learned

- Maven richtig konfigurien, Tutorial: http://www.patrick-gotthard.de/maven-tutorial-fuer-anfaenger

- Java Policy File (jre/lib/security/java.policy) angepasst:

  Folgende Zeile: permission java.net.SocketPermission "localhost:0", "listen";
  Durch folgende ersetzt: permission java.net.SocketPermission "*", "connect,accept,listen";

## 6. Testprotokoll

- ✓ 1 Balancer, 1 Server, 1 Client auf Localhost
- ✓ 1 Server, 1 Client auf Localhost
- ✓ 1 Balancer, 1 Server, 1 Client über Netzwerk verteil
- ✓ 1 Server, 1 Client über Netzwerk verteil
- ✓ 1 Balancer, 5 Server, 8 Client über Netzwerk verteilt

Für alle durchgeführten Tests wurden 100.000 Stellen von Pi berechnet was jeweils etwa 20 Sekunden Rechenzeit in Anspruch genommen hat.

**UTA**

- ✓ Lokale Implementierung (CLI Arguments etc.)
- ✓ RMI Implementierung
- ✓ Client muss eine richtige Antwort für PI direkt vom CalculationServer bekommen
- ✓ Client muss eine richtige Antwort für PI über den Balancer bekommen
- ✓ Balancer muss Last an Server verteilen
- ✓ Alles muss sowohl als Localhost als auch über das Netzwerk funktionieren

## 7. Quellen


- Sourcecode zur PI Berechnung, http://homepage.uibk.ac.at/~csag8802/client/Pi.java, abgerufen am 7.1.2015
- Java RMI Tutorial, http://docs.oracle.com/javase/tutorial/rmi/overview.html, abgerufen am 5.1.2015
- Distributed Objects RMI M. Borko T. Micheler, https://elearning.tgm.ac.at/mod/resource/view.php?id=31094, abgerufen am 6.1.2015
- Maven Tutorial Patrick Gotthard 2011, http://www.patrick-gotthard.de/maven-tutorial-fuer-anfaenger, abgerufen am 6.1.2015