DEZSYS07

Pi Calculator

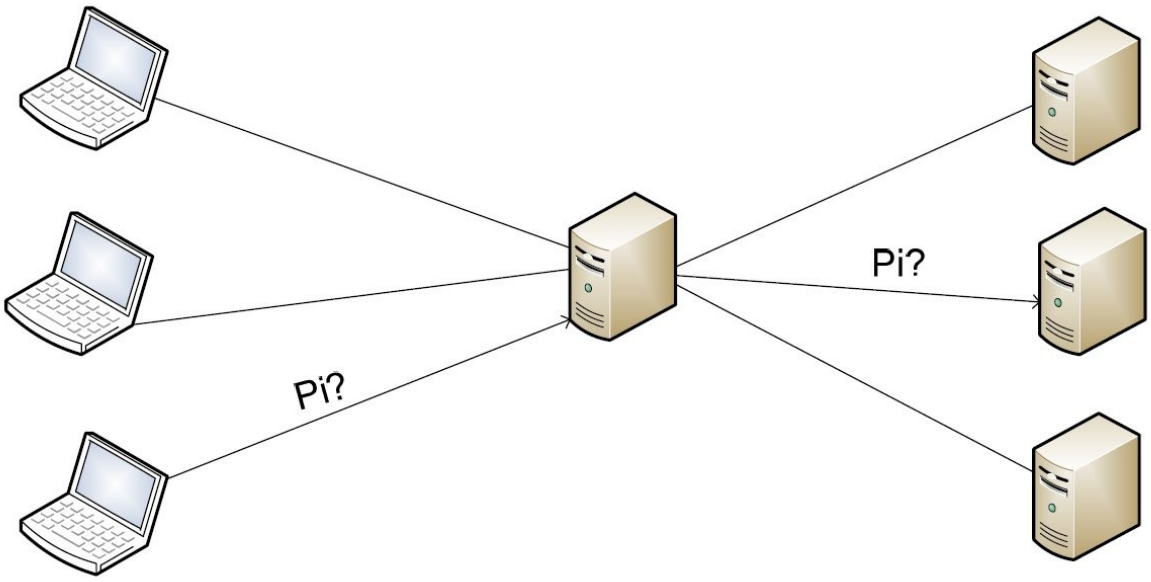
von René Hollander und Paul Kalauner 4AHIT

14.12.14

GitHub-Repolink: https://github.com/ReneHollander/dezsys07-picalc

Table of Contents

Angabe



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

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

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:

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.

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.

Gruppenarbeit

Die Arbeit ist als 2er-Gruppe zu lösen und über das Netzwerk zu testen! Nur localhost bzw. lokale Testzyklen sind unzulässig und werden mit 6 Minuspunkten benotet!

Benotungskriterien

o 12 Punkte: Java RMI Implementierung (siehe Punkt 1)
o 12 Punkte: Implementierung des Balancers (siehe Punkt 2)

o davon 6 Punkte: Balancer
o davon 2 Punkte: Parameter - Name des Objekts
o davon 2 Punkte: Listing der Server (dyn. Hinzufügen und Entfernen)
o davon 2 Punkte: Testprotokoll mit sinnvollen Werten für Serverpoolgröße und Iterationen

Quellen

An Overview of RMI Applications, Oracle Online Resource,
http://docs.oracle.com/javase/tutorial/rmi/overview.html (last viewed 28.11.2014)

Planung

Aufwandsschätzung

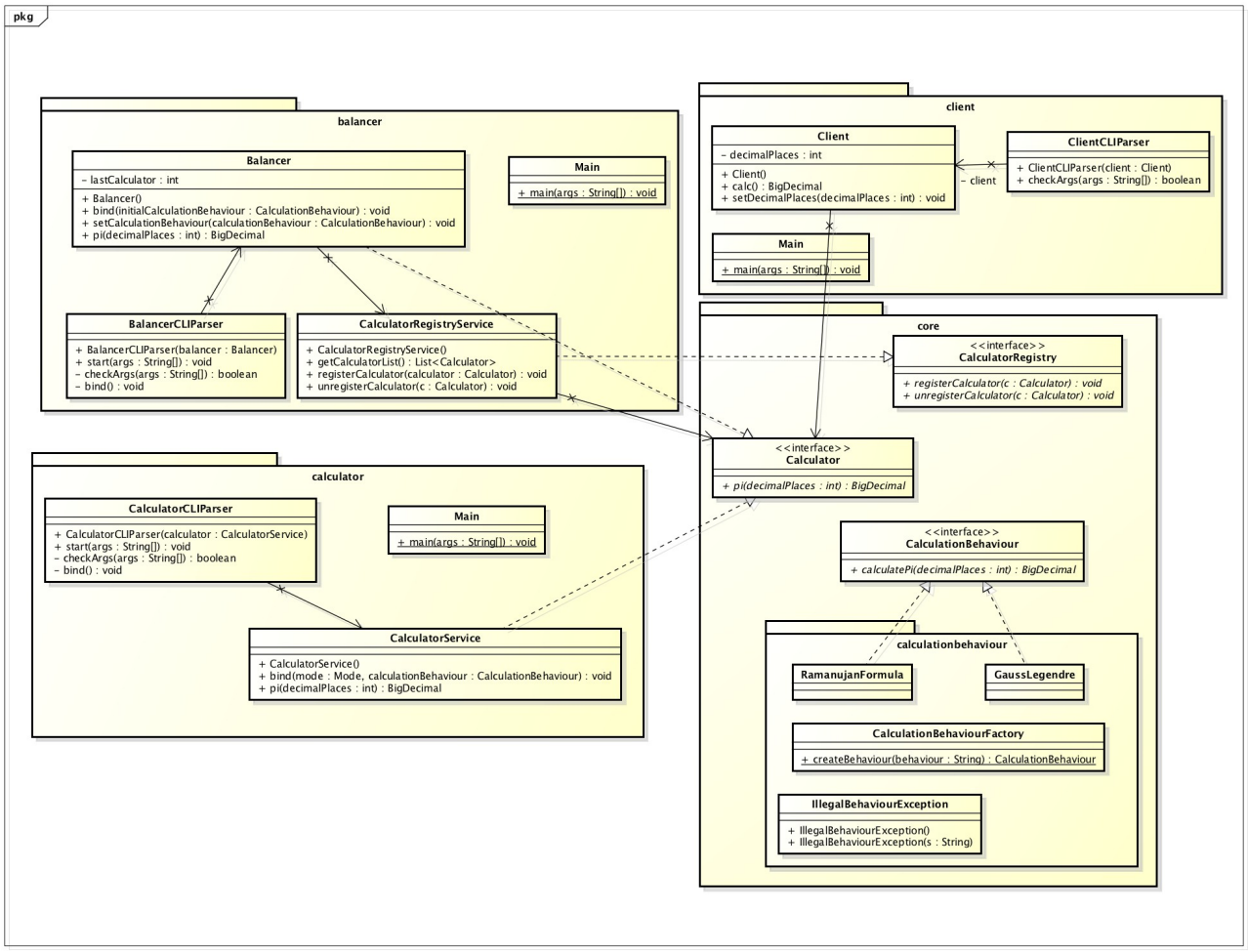
Arbeitspaket	**geschätzte Dauer**	**Person/en**
Designüberlegung	50 Minuten	Hollander, Kalauner
Definieren der Core-Interfaces	10 Minuten	Hollander
Definieren der Behaviours	5 Minuten	Hollander
Implementierung der Behaviours	10 Minuten	Kalauner
Implementierung des Servers	60 Minuten	Hollander
Implementierung des Clients	60 Minuten	Kalauner
Implementierung des Balancers	30 Minuten	Hollander, Kalauner
Testing	60 Minuten	Hollander, Kalauner
Gesamt	**4 Stunden, 45 Minuten**	-

Zeitaufzeichnung

Arbeitspaket	**Datum**	**von**	**bis**	**Dauer**	**Person/en**
Designüberlegung	12/05/14	12:45:00	13:20:00	35 Minuten	Hollander, Kalauner
Definieren der Core-Interfaces	12/12/14	11:40:00	11:45:00	5 Minuten	Hollander
Implementierung des Clients	12/12/14	11:40:00	12:00:00	20 Minuten	Kalauner
Implementierung des Servers	12/12/14	11:50:00	12:15:00	25 Minuten	Hollander
Fehlersuche	12/12/14	12:20:00	12:40:00	20 Minuten	Hollander, Kalauner
Definieren der Behaviours	12/12/14	12:40:00	12:55:00	15 Minuten	Hollander
Implementierung der Behaviours	12:12.14	13:00:00	13:10:00	10 Minuten	Kalauner
Implementierung des Balancers	12/12/14	13:10:00	13:45:00	35 Minuten	Hollander, Kalauner
Parsen der CLI Argumente	12/13/14	09:00:00	10:00:00	60 Minuten	Kalauner
Verbesserungen Calculator, Client, Balancer	12/13/14	10:00:00	10:45:00	45 Minuten	Hollander
Fehlendes Javadoc hinzufügen	12/13/14	14:45:00	11:15:00	30 Minuten	Hollander
Testing	12/14/14	10:10:00	11:20:00	70 Minuten	Hollander
Testing	12/14/14	10:10:00	11:20:00	70 Minuten	Kalauner
Integrationtesting	12/15/14	x	x	x	Hollander

Gesamt: x Stunden, x Minuten

Klassendiagramm



Designüberlegung

Wir haben uns dazu entschlossen das Projekt in 4 Module zu unterteilen:

- Core
- Calculator
- Balancer
- Client

Durch die Verwendung von Maven (siehe Arbeitsdurchführung) erleichtert uns diese Aufteilung das Deployment.

Core:

Im Core Modul befinden sich alle Interfaces und die Implementierungen der Algorithmen für die Berechnung von Pi. Diese Algorithmen implementieren das Interface `CalculationBehaviour`. Dies macht es mittels Strategy Pattern möglich, die Algorithmen während der Laufzeit zu wechseln. Außerdem können leicht neue hinzugefügt werden.

Die Methode `createBehaviour(String behaviour)` der `CalculationBehaviourFactory` liefert je nach Benutzeingabe ein Behaviour zurück. Bei ungültiger Eingabe wird die `IllegalBehaviourException` geworfen.

Calculator:

Der CalculatorService implementiert das Interface `Calculator` und ruft die calculateMethode() des jeweiligen Behaviours, welches in der Registry gespeichert ist, auf wenn er vom Balancer dazu aufgefordert wird.

Balancer:

Der Balancer implementiert ebenfalls das Interface `Calculator`. So macht es für den Client keinen Unterschied, ob er sich direkt mit einem Service verbindet oder nur indirekt über den Balancer. Mithilfe des Round-Robin Verfahren verteilt der Balancer die Anfragen an die registrierten CalculatorServices.

Client:

Der Client stellt Anfragen an den Balancer. Er kann Pi mit unterschiedlichen Dezimalstellen von den Services berechnen lassen.

Arbeitsdurchführung

Projektaufbau

Das Projekt wurde in mehrere Module aufgeteilt:

- Core: Interfaces und Helferklassen die von mehreren Modulen benötigt werden
- Balancer: Verteilt Anfragen auf mehrere Calculators ohne das der Client etwas bemerkt
- Calculator: Die eigentliche Berechnung von Pi übernimmt der Calculator
- Client: Sendet Anfragen an einen Calculator oder Balancer und gibt des ausgerechneten Wert aus

Durch die Aufteilung des Projektes in mehrere Module wurde eine logische und übersichtliche Projektstruktur geschaffen.

Build Tool

Als Built Tool wurde Maven verwendet. Das einfache Dependency Management hat es uns ermöglicht, das Core Modul einfach in die anderen Module einzubinden. Auch werden Dependecies wie Log4J2 und Apache Commons CLI zur verfügung gestellt.

Core

Das Core Modul enthält die RMI Schnittstelle, Helferklassen und Algorithmen für die Berechnung von Pi. Das Core Module wird beim Erstellen der Jar Archive für Balancer, Calculator und Client in das Jar File eingebunden.

Algorithmen die zur Verfügung stehen:

- GaussLegendre [1]
- RamanujanFormula [2]

Balancer

Der Balancer hat die Aufgabe, Anfragen auf mehrere Rechner zu verteilen, ohne das der Client etwas davon mitbekommt. Als Distributionsalgorithmus wurde Round Robin verwendet. Dieser Algorithmus schickt eine Anfrage an den 1. Calculator, die nächste Anfrage an den 2. Calculator. Nach dem letzten Calculator kommt wieder der erste an die Reihe.
Dieser Algorithmus ist leicht zu implementieren und kann die Last gut verteilen, wenn alle Anfragen gleich komplex sind. Sollte ein Anfrage Pi auf zum Beispiel 100000 Nachkommastellen genau haben wollen, wird ein Server damit voll Belastet. Wenn eine weitere Anfrage auf diesen Server zugeteilt wird, verlangsamt sich die Berechnung. Wenn zu viele komplexe Berechnungen auf einen Server zugeteilt werden, kann es passieren das dieser abstürzt. Diesem Problem kann man mit einer Last basierten Verteilung lösen, die aber schwerer zu implementieren ist.

Damit sich ein Calculator beim Balancer anmelden kann, wird ein CalculatorRegistry Service zur Verfügung gestellt. Dieser Service fügt den Calculator dann zu einer Liste hinzu aus der der Round Robin Algorithmus einen Calculator nimmt und die Anfrage an den Calculator schickt.

Auch stellt der Balancer den Algorithmus der für die Berechnung von Pi verwendet werden soll über die RMI Registry zur verfügung.

Calculator

Der Calculator hat zwei Betriebsmodi:

- Standalone
- Behind Balancer

Im “Standalone” Modus, erzeugt der Calculator eine RMI Registry und stellt seinen Service zur Berechnung direkt zur verfügung. Der Algorithmus für die Berechnung von Pi kann über die Kommandozeilenargumente spezifiziert werden.
Im “Behind Balancer” Modus meldet sich der Calculator an dem spezifizierten Balancer an und steht dann für Anfragen zu verfügung. Der Algorithmus für die Berechnung von Pi wird bei jeder Anfrage von der RMI Registry abgefragt.

Client

Der Client ist ein einfaches Programm, dass sich über den spezifizierten Hostname und Port am Calculator/Balancer anmeldet und Anfragen für die Berechnung von Pi verschickt. Die Genauigkeit kann über ein Argument an das Programm übergeben werden. Das Ergebnis wird auf der Kommandozeile ausgegeben.

Lessons Learned

- Verwendung von Java RMI [3]

Code Ausschnitt:

Policy File und Registry setzen:

```
public static void setupPolicy() {
	LOG.info("Setting up Policy");
	if (System.getSecurityManager() == null) {
		System.setProperty("java.security.policy",
System.class.getResource("/policy/java.policy").toString());
		System.setSecurityManager(new SecurityManager());
	}
}

public static void setupRegistry() throws RemoteException {
	LOG.info("Setting up Registry");
	LocateRegistry.createRegistry(1099);
}
```

Service bei Registry registrieren:

```
public void bind() {
	RMIUtil.setupPolicy();
	RMIUtil.setupRegistry();
	Naming.bind("NameDesServices", this);
}
```

Lookup des Services am Client:

```
public void connect(String host, int port) {
    RMIUtil.setupPolicy();
    service = (Calculator) Naming.lookup("rmi://localhost:10999/NameDesServices");
}
```

Methode des Dienstes aufrufen:

```
 return service.pi(10000);
```

Testprotokoll

Gestartete Server: 3
Anzahl der Anfragen: 3
Iterationsgröße: 100 000
Erwartetes Ergebnis: Die Server sollten parallel jeweils eine Anfrage verarbeiten.

Für den Test ist die Iterationsgröße für jede Anfrage gleich, es ist aber auch möglich Anfragen mit unterschiedlich großen Iterationsgrößen zu stellen.

Screenshots

Balancer:

```
12:44:51:955 [main] INFO  at.hollanderkalauner.picalc.balancer.Balancer - Binding Balancer
12:44:51:994 [main] INFO  at.hollanderkalauner.picalc.core.RMIUtil - Setting up Policy
12:44:51:995 [main] INFO  at.hollanderkalauner.picalc.core.RMIUtil - Setting up Registry
12:44:52:001 [main] INFO  at.hollanderkalauner.picalc.balancer.Balancer - Setting Calculation Behaviour to at.hollanderkalauner.picalc.core.calculationbehaviour.C
12:44:52:385 [main] INFO  at.hollanderkalauner.picalc.balancer.Balancer - Successfully bound Balancer
12:44:58:691 [RMI TCP Connection(4)-10.0.0.1] INFO  at.hollanderkalauner.picalc.balancer.CalculatorRegistryService - Adding Proxy[Calculator,RemoteObjectInvocatic
12:45:02:236 [RMI TCP Connection(6)-10.0.0.1] INFO  at.hollanderkalauner.picalc.balancer.CalculatorRegistryService - Adding Proxy[Calculator,RemoteObjectInvocatic
12:45:04:846 [RMI TCP Connection(8)-10.0.0.1] INFO  at.hollanderkalauner.picalc.balancer.CalculatorRegistryService - Adding Proxy[Calculator,RemoteObjectInvocatic
12:45:22:676 [RMI TCP Connection(10)-10.0.0.1] INFO  at.hollanderkalauner.picalc.balancer.Balancer - Routing request with 100000 decimal places to service Proxy[C
12:45:25:437 [RMI TCP Connection(13)-10.0.0.1] INFO  at.hollanderkalauner.picalc.balancer.Balancer - Routing request with 100000 decimal places to service Proxy[C
12:45:32:564 [RMI TCP Connection(16)-10.0.0.1] INFO  at.hollanderkalauner.picalc.balancer.Balancer - Routing request with 100000 decimal places to service Proxy[C
```

Server 1:

```
12:44:58:619 [main] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Initializing CalculatorService
12:44:58:628 [main] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Binding Service CalculatorService{} in balancer mode!
12:44:58:629 [main] INFO  at.hollanderkalauner.picalc.core.RMIUtil - Setting up Policy
12:44:58:691 [main] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Service CalculatorService{} successfully bound and listening for Requests through the balancer!
12:45:22:677 [RMI TCP Connection(2)-10.0.0.1] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Calculating Pi to 100000 decimal places
```

Server 2:

```
12:45:02:177 [main] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Initializing CalculatorService
12:45:02:185 [main] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Binding Service CalculatorService{} in balancer mode!
12:45:02:187 [main] INFO  at.hollanderkalauner.picalc.core.RMIUtil - Setting up Policy
12:45:02:236 [main] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Service CalculatorService{} successfully bound and listening for Requests through the balancer!
12:45:25:439 [RMI TCP Connection(2)-10.0.0.1] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Calculating Pi to 100000 decimal places
```

Server 3:

```
12:45:04:778 [main] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Initializing CalculatorService
12:45:04:788 [main] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Binding Service CalculatorService{} in balancer mode!
12:45:04:789 [main] INFO  at.hollanderkalauner.picalc.core.RMIUtil - Setting up Policy
12:45:04:846 [main] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Service CalculatorService{} successfully bound and listening for Requests through the balancer!
12:45:32:566 [RMI TCP Connection(2)-10.0.0.1] INFO  at.hollanderkalauner.picalc.calculator.CalculatorService - Calculating Pi to 100000 decimal places
```

Anhand der Zeitausgabe ist ersichtlich, dass die Anfragen parallel verarbeitet werden.

Client 1:

```
12:45:22:572 [main] INFO  at.hollanderkalauner.picalc.client.Client - Initialising Client
12:45:22:576 [main] INFO  at.hollanderkalauner.picalc.core.RMIUtil - Setting up Policy
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482
```

Client 2:

```
12:45:25:319 [main] INFO  at.hollanderkalauner.picalc.client.Client - Initialising Client
12:45:25:323 [main] INFO  at.hollanderkalauner.picalc.core.RMIUtil - Setting up Policy
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482
```

Client 3:

```
12:45:32:003 [main] INFO  at.hollanderkalauner.picalc.client.Client - Initialising Client
12:45:32:095 [main] INFO  at.hollanderkalauner.picalc.core.RMIUtil - Setting up Policy
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482
```

Quellen

[1]: Gauss-Legendre Algorithmus von Gaute Lykkenborg:
http://java.lykkenborg.no/2005/03/computing-pi-using-bigdecimal.html
Zuletzt abgerufen: 12.12.2014

[2]: Ramanujan Formula Algorithmus:
http://cronodon.com/Programming/Pi.html
Zuletzt abgerufen: 12.12.2014

[3]: An Overview of RMI Applications, Oracle Online Resource,
http://docs.oracle.com/javase/tutorial/rmi/overview.html
Zuletzt abgerufen: 12.12.2014