

Gruppe Pawlowsky, Sochovsky

# Distributed Pi Calculator

## RMI

VSDB 4BHITS 29.02.2012

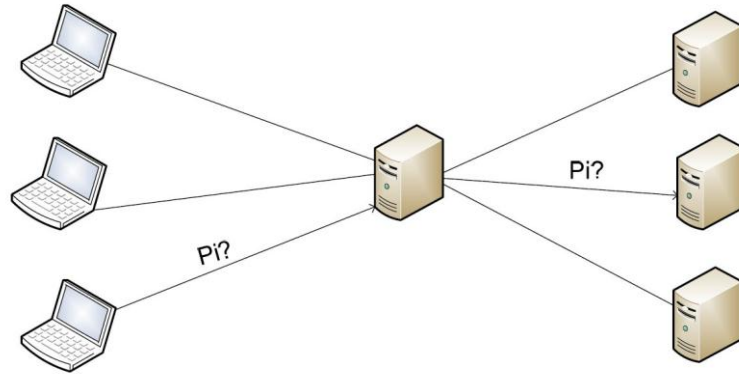
Pawlowsky, Sochovsky

## Inhalt

Aufgabenstellung.....	2
Distributed PI Calculator .....	2
<i>Zusätzliches</i> .....	3
Start der Applikation .....	4
Designüberlegung.....	4
Arbeitsteilung .....	6
Arbeitsdurchführung .....	6
Tests .....	8
Lokale Tests: .....	8
Großer Test lokal mit mehr und weniger stellen .....	12
Netzwerk Tests: .....	13
Quellen .....	16

## Aufgabenstellung

### *Distributed PI Calculator*



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 Balancier 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. Entwickeln Sie ein Serverprogramm, das eine CalculatorImpl-Instanz erzeugt und beim RMI-Namensdienst registriert. Entwickeln 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 Balancier, indem Sie eine Klasse CalculatorBalancer von Calculator ableiten und die Methode pi() entsprechend implementieren. Dadurch verhält sich der Balancier aus Sicht der Klienten genauso wie der Server, d.h. das Klientenprogramm muss nicht verändert werden. Entwickeln Sie ein Balancierprogramm, 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 Balancier übergibt, ohne es in die Registry zu schreiben. Verwenden Sie dabei ein eigenes Interface des Balancers, welches in die Registry gebündelt wird, um den Servern das Anmelden zu ermöglichen.
  - ~~Das Balancier-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 Balancier dadurch mehrere eintreffende Aufrufe parallel bearbeiten kann, andererseits müssen dadurch im Balancier 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 Balancier zwischen Start eines Servers und Abfragen des Namensdienstes einige Sekunden warten.~~

Testen Sie das entwickelte System, indem Sie den Balancier mit verschiedenen Serverpoolgrößen starten und mehrere Klienten gleichzeitig Anfragen stellen lassen. Wählen Sie die Anzahl der Iterationen bei der Berechnung von pi entsprechend gross, sodass eine Anfrage lang genug dauert um feststellen zu können, dass der Balancier 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 16.02.2012)

### Zusätzliches

Zusätzlich ergibt sich aus den Meta-Regeln, die besagen, dass man gelerntes, sofern sinnvoll, immer anwenden muss, dass man das Round-Robin-Verfahren, welches das Load-Balancing übernimmt, mithilfe des Strategy-Patterns implementieren muss, um die Erweiterbarkeit zu sichern.

## Start der Applikation

Um die Applikation zu starten muss zuerst einmal die „Balancer.jar“ gestartet werden, die keine zusätzlichen Konsolenargumente benötigt.

Dann muss/müssen auf demselben oder auch auf einem/mehreren anderen PC/s, einer oder mehrere „Server.jar“s gestartet werden, die beim Starten die IP-Adresse des Balancers als Konsolenargument benötigen, um sich zu eben diesem hinzufügen zu können. Wenn man den Server auf dem gleichen PC wie den Balancer gestartet hat kann man hier „localhost“ übergeben.

Zuletzt muss/müssen noch auf demselben oder auch auf einem/mehreren anderen PCs, einer oder mehrere „Client.jar“s gestartet werden. Diese benötigen als Konsolenargumente zum einen ebenfalls die IP-Adresse des Balancers, um eine Verbindung zu diesem aufbauen zu können und zum anderen die Anzahl an Stellen die das berechnete Pi hinter dem Komma haben soll.

Hat man diese Schritte alle richtig durchgeführt sollte man am Client Pi mit der gewünschten Genauigkeit ausgegeben bekommen.

## Designüberlegung

Grundsätzlich sieht unser Aufbau so aus:

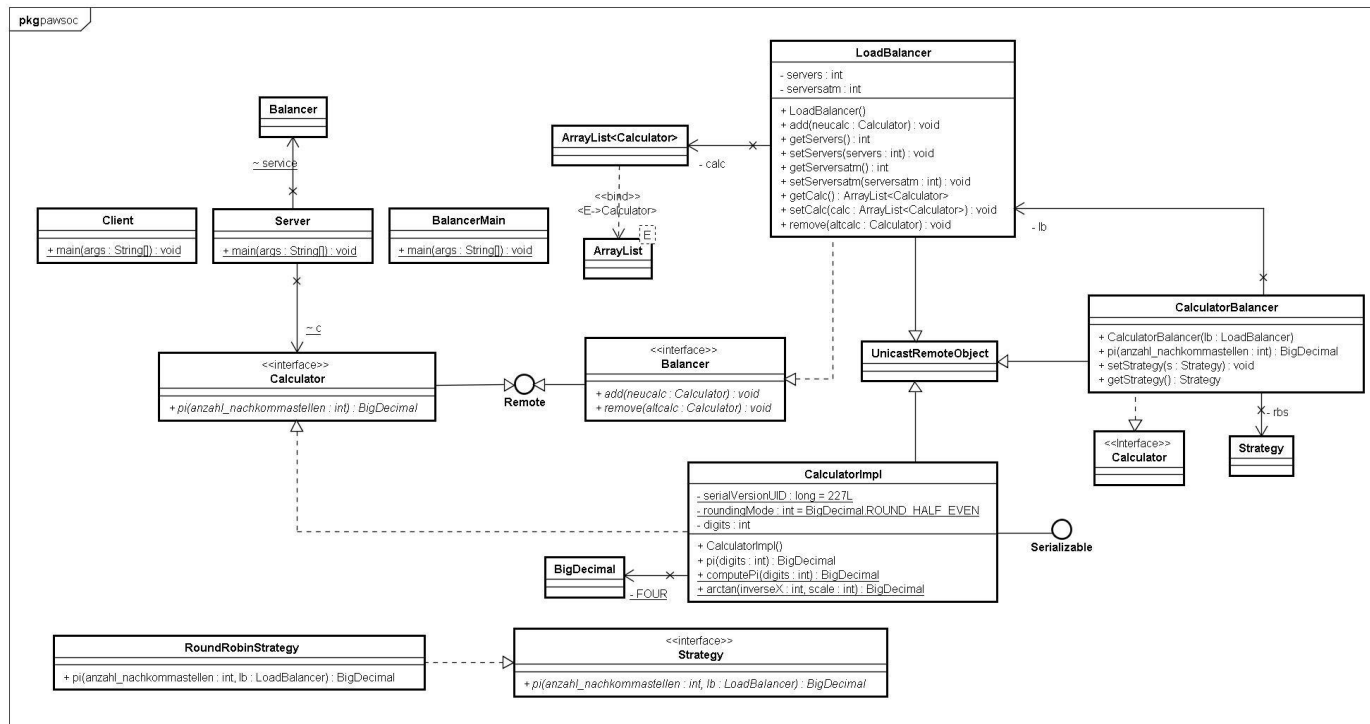
Wir haben zwei Balancer einer macht das Load-Balancing über eine ausgelagerte Strategie. Zurzeit ist nur die Round-Robin-Strategie implementiert, allerdings ist der Balancer erweiterbar und es kann auch zur Laufzeit eine neue Strategie gesetzt werden. Der zweite Balancer übernimmt das entfernen und Hinzufügen von Servern zu einer ArrayList, die er selbst verwaltet. Außerdem speichert dieser die derzeit angemeldete Anzahl an Servern und den zuletzt benutzten Server, um die Strategien zu vereinfachen. Beide Balancer sich UnicastRemoteObjects, das heißt sie können an den RMI Namensdienst gebunden werden. Das binden übernimmt eine Balancer-Main-Klasse die auch eine Verbindung zwischen den zwei Balancern herstellt, indem dem CalculatorBalancer(der zuerst erwähnte) den LoadBalancer(der zweite) als Parameter übergibt, damit der Calculator-Balancer und dessen Strategien auf die Serverliste zugreifen kann.

Wenn nun ein Server gestartet wird sucht dieser im RMI-Namensdienst unter der übergebenen(Konsolenargument) IP-Adresse des Balancers nach einer Referenz auf den LoadBalancer, um dessen add-Methode auszuführen die den Server beim Namensdienst registriert. Pi-Berechnungen führt der Server über eine Recheneinheit namens CalculatorImpl aus, welche mittels des Arcus-Tangens-Verfahrens nach Michin's Formel Pi auf beliebig viele Stellen genau berechnen kann. Eine Referenz auf diese Berechnungseinheit wird vom Server beim Aufruf der add-Methode an den Balancer übergeben. Wenn ein Server geschlossen wird, wird er über einen shutdownHook, der beim Beenden einer main-Methode automatisch über einen Thread noch eine Methode ausführen kann, automatisch aus dem Namensdienst entfernt und die Berechnungseinheit wird unexported.

Wenn ein Client nun über RMI eine Anfrage an den CalculatorBalancer stellt um sich Pi berechnen zu lassen, so gibt ihm dieser über Load-Balancing das berechnete Pi von dem ihm am sinnvollsten erscheinenden Server zurück. Für den Client sieht der Calculator-Balancer wie ein einfacher Calculator aus und die Auslagerung der Berechnung an verschiedene Server bleibt für den Client unsichtbar also transparent, da Transparenz ein Grundprinzip von RMI ist.

Die Strategy haben wir über das Strategy-Pattern ausgelagert, um bei den Load-Balancing Algorithmen Erweiterbarkeit sicherstellen zu können.

Als UML-Klassendiagramm dargestellt sehen die Verbindungen zwischen den Klassen in etwa so aus:



## Aufwandsabschätzung

RMI-Verbindung implementieren: 2h  
Round-Robin implementieren: 1h  
Shutdown-Hook implementieren: 1h  
Umgang mit Benutzereingaben + Synopsis: 0,5h  
Feinschliffe für Zusatzpunkte: 3h  
Tests: 1h

**Gesamt: 9,5h**

## Arbeitsteilung

**Pawlowsky:**

RMI-Verbindung implementiert  
Umgang mit Benutzereingaben + Synopsis  
Shutdown-Hook implementiert  
Feinschliffe

**Sochovsky:**

RMI-Verbindung implementiert  
Round-Robin implementiert  
Strategy-Pattern implementiert  
Tests

## Arbeitsdurchführung

**Sochovsky:**

Freitag 20.01.2012: RMI-Verbindung + Round-Robin implementiert 1h  
Sonntag 26.02.2012: Strategy-Pattern implementiert 1h  
Donnerstag 01.03.2012: Feinschliffe + Tests 2.5h  
**Gesamt: 4.5h**

**Pawlowsky:**

Freitag 24.02.2012: RMI-Verbindung implementiert 1h  
Montag 27.02.2012: Feinschliffe 1.5h  
Mittwoch 29.02.2012: Feinschliffe + Shutdown-Hook + Benutzereingaben + Synopsis 2h  
Donnerstag 01.03.2012: Tests 1h  
**Gesamt: 5.5h**

**Gesamt: 10h**

## Resultate/Niederlagen

### Sochovsky:

Da ich zuerst 1. der Aufgabenstellung erledigt habe, hatte ich ein Problem beim Einfügen eines zweiten Objekts, das sich am RMI-Namensdienst registriert, da ich vergessen habe für dieses ein zweites Remote-Interface zu erzeugen. Dadurch bekam ich eine ClassCastException: \$Proxy0, als ich den zweiten LoadBalancer auf das Remote-Interface casten wollte.

#### Lösung:

Ich habe ein neues Remote-Interface erzeugt, welches eine add-Methode für den Server beinhaltet und im LoadBalancer implementiert. Dadurch konnte ich im Server nun den Service der die add-Methode anbietet auf dieses Interface casten und die Exception ist nicht mehr aufgetreten.

### Pawlowsky:

Anfangs habe ich das Policy-File über ein Konsolenkommando hinzugefügt. Nach einigem Suchen konnte ich dann ein Kommando finden, welches mir das Policy-File auch während der Laufzeit automatisch einliest. Diese Variante funktioniert allerdings nicht wenn das Policy-File innerhalb der exportierten jar-Datei liegt.

#### Lösung:

Mit einer URL kann man die aktuelle Klasse und deren Ressourcen getten. Wenn man an diese URL, dann noch einen Slash und den Namen des Policy-Files anhängt, dann ist es möglich das Policy-File auch innerhalb der jar-Datei auszulesen.

### Pawlowsky:

Anfangs wollte ich den Server durch Eingabe von ,q' schließbar machen, um ihn perfekt beim Schließen aus der Liste entfernen zu können, dies ergab dich allerdings als aufwendig, da die add-Methode, die der Server aufruft, um sich beim Balancer zu registrieren wartet, bis der Server geschlossen wird und daher ist eine Eingabe nicht möglich und man hätte die add-Methode in einen Thread auslagern müssen.

#### Lösung:

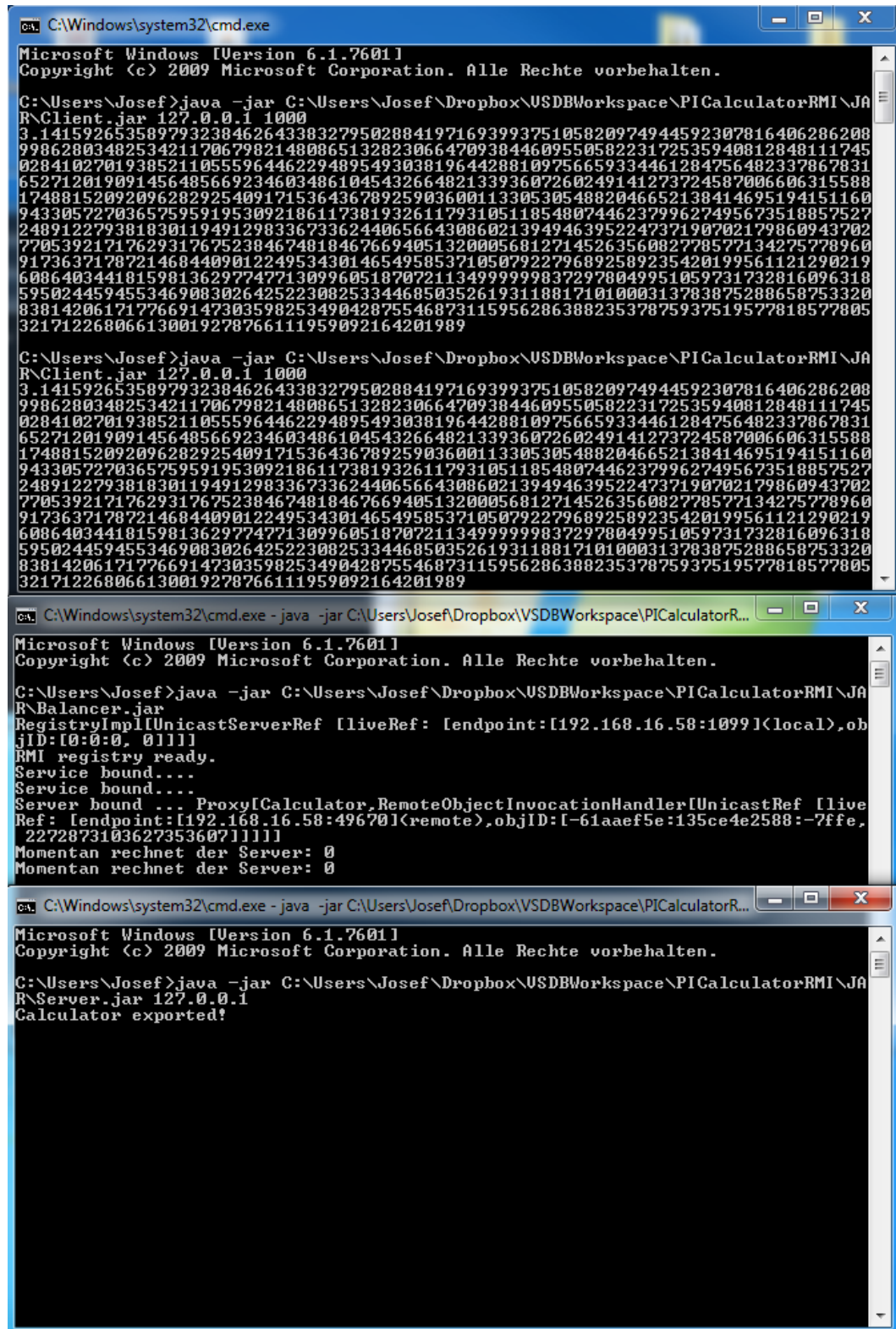
Ich habe nun mithilfe eines Shutdown-Hooks das Schließen der jar mittels Ctrl+c abgefangen und vor der Beendigung der JVM eine remove-Methode aufgerufen, um den Server aus dem Balancer zu entfernen und die übergebene CalculatorImpl-Instanz unexported.



## Tests

### Lokale Tests:

Durchführung mit 1 Client, 1 Server und dem Balancer: es soll 2-mal Pi berechnet werden



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
Client.jar 127.0.0.1 1000
3.141592653589793238462643383279502884197169399375105820974944592307816406286208
99862803482534211706798214808651328230664709384460955058223172535940812848111745
02841027019385211055596446229489549303819644288109756659334461284756482337867831
65271201909145648566923460348610454326648213393607260249141273724587006606315588
17488152092096282925409171536436789259036001133053054882046652138414695194151160
94330572703657595919530921861173819326117931051185480744623799627495673518857527
24891227938183011949129833673362440656643086021394946395224737190702179860943702
77053921717629317675238467481846766940513200056812714526356082778577134275778960
91736371787214684409012249534301465495853710507922796892589235420199561121290219
60864034418159813629774771309960518707211349999998372978049951059731732816096318
59502445945534690830264252230825334468503526193118817101000313783875288658753320
83814206171776691473035982534904287554687311595628638823537875937519577818577805
321712268066130019278766111959092164201989

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Balancer.jar
RegistryImpl[UnicastServerRef [liveRef: [endpoint:[192.168.16.58:1099]<local>,obj
ID:[0:0:0, 0]]]]
RMI registry ready.
Service bound....
Service bound....
Server bound ... Proxy[Calculator.RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[192.168.16.58:49670]<remote>,objID:[-61aaef5e:135ce4e2588:-7ffe,
2272873103627353607]]]]
Momentan rechnet der Server: 0
Momentan rechnet der Server: 0

C:\Windows\system32\cmd.exe - java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorR...
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Server.jar 127.0.0.1
Calculator exported!
```

Durchführung mit 2 Client, 2 Server und dem Balancer: es soll 4-mal Pi berechnet werden

```
C:\Windows\system32\cmd.exe
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Client.jar 127.0.0.1 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Client.jar 127.0.0.1 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Client.jar 127.0.0.1 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Client.jar 127.0.0.1 10
3.1415926536

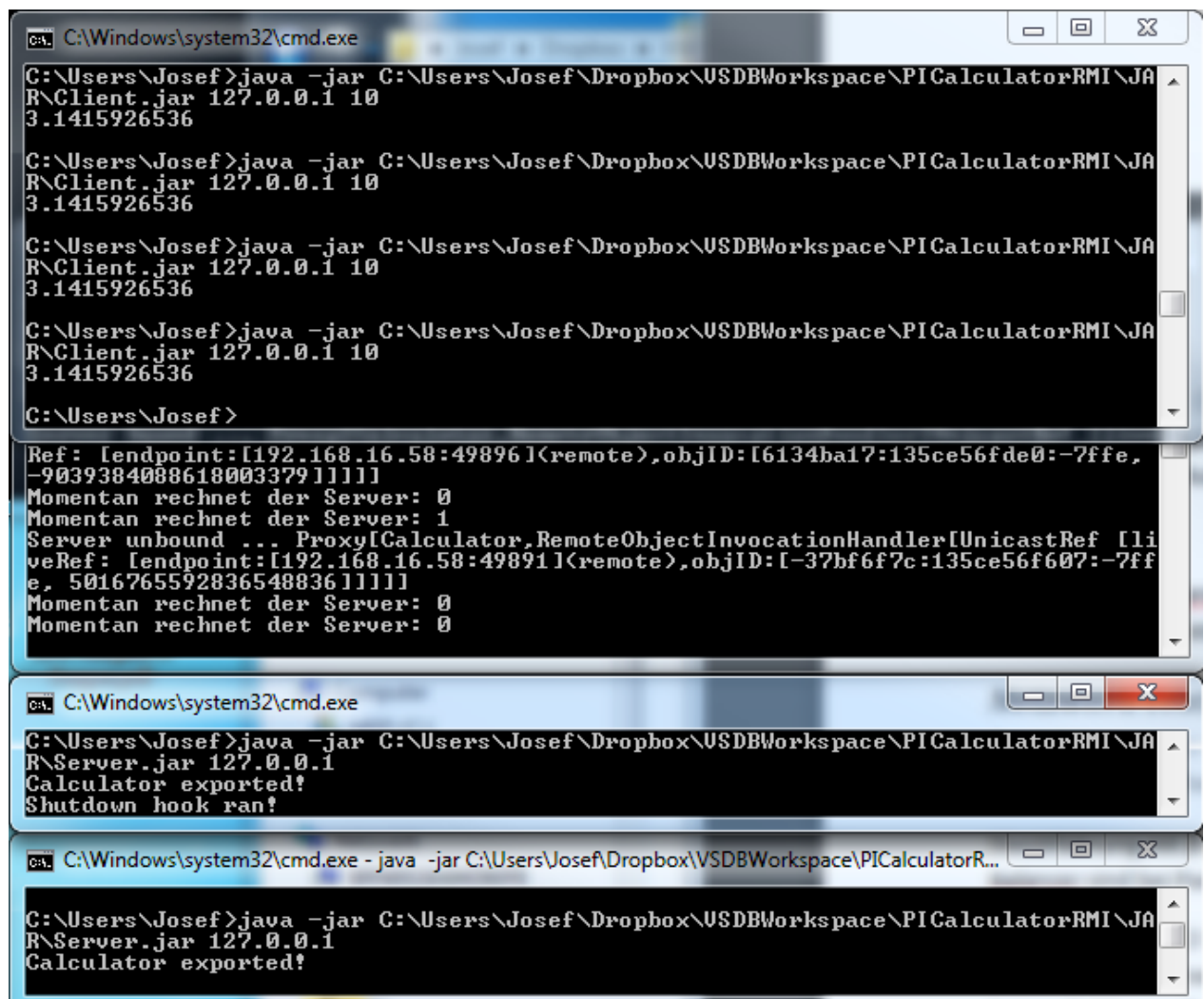
C:\Users\Josef>

C:\Windows\system32\cmd.exe - java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorR...
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Balancer.jar
RegistryImpl[UnicastServerRef [liveRef: [endpoint:[192.168.16.58:10991]<local>,obj
jID:[0:0:0, 0]]]]
RMI registry ready.
Service bound....
Service bound....
Server bound ... Proxy[Calculator.RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[192.168.16.58:49837]<remote>,objID:[20e4de95:135ce54909d:-7ffe,
3958127653829759115]]]]
Server bound ... Proxy[Calculator.RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[192.168.16.58:49841]<remote>,objID:[-54d43eca:135ce5499f0:-7ffe,
-3108568369240827585]]]]
Momentan rechnet der Server: 0
Momentan rechnet der Server: 1
Momentan rechnet der Server: 0
Momentan rechnet der Server: 1

C:\Windows\system32\cmd.exe - java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorR...
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Server.jar 127.0.0.1
Calculator exported!

C:\Windows\system32\cmd.exe - java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorR...
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Server.jar 127.0.0.1
Calculator exported!
```

Durchführung mit 2 Client, 2 Server und dem Balancer: es soll 4-mal Pi berechnet werden – Der zuerst erstellte Server wird geschlossen und es soll kein Fehler auftreten (RoundRobin soll hier keinen Fehler produzieren)



```
C:\Windows\system32\cmd.exe
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Client.jar 127.0.0.1 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Client.jar 127.0.0.1 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Client.jar 127.0.0.1 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Client.jar 127.0.0.1 10
3.1415926536

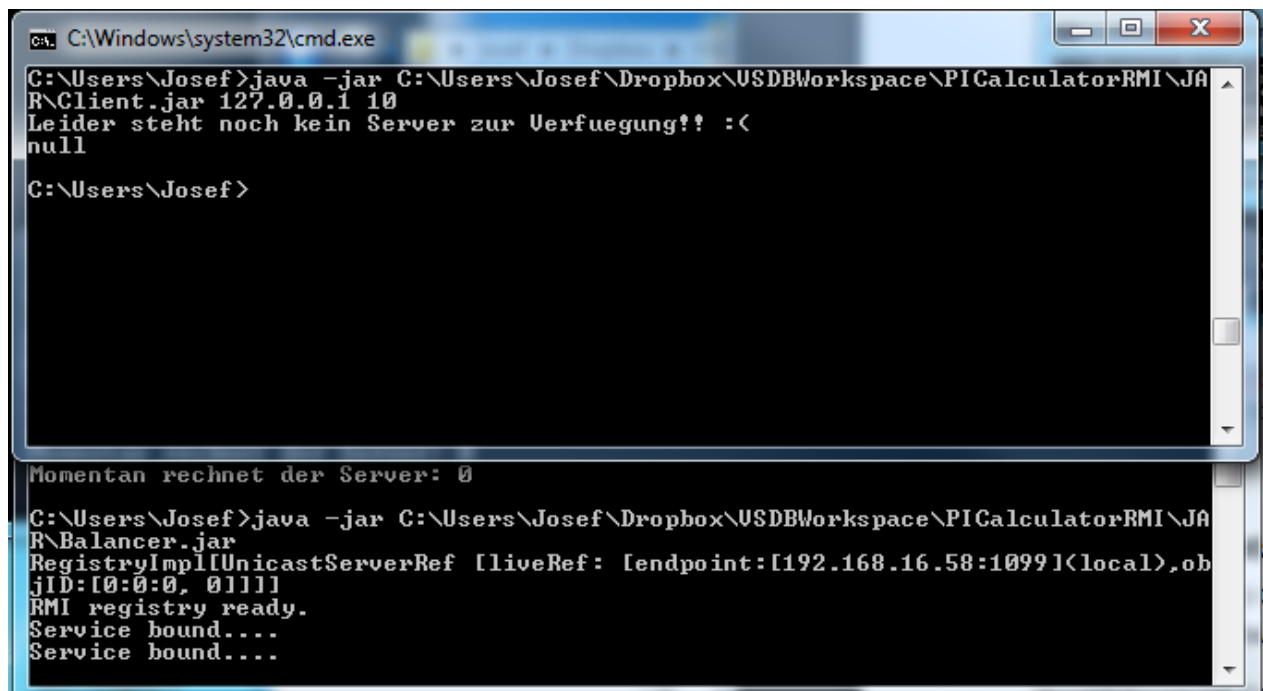
C:\Users\Josef>

Ref: [endpoint:[192.168.16.58:49896]<remote>,objID:[6134ba17:135ce56fde0:-7ffe,
-903938408861800337911111]
Momentan rechnet der Server: 0
Momentan rechnet der Server: 1
Server unbound ... Proxy[Calculator.RemoteObjectInvocationHandler[UnicastRef [li
veRef: [endpoint:[192.168.16.58:49891]<remote>,objID:[-37bf6f7c:135ce56f607:-7ff
e, 50167655928365488361111]
Momentan rechnet der Server: 0
Momentan rechnet der Server: 0

C:\Windows\system32\cmd.exe
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Server.jar 127.0.0.1
Calculator exported!
Shutdown hook ran!

C:\Windows\system32\cmd.exe - java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorR...
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Server.jar 127.0.0.1
Calculator exported!
```

Erstellen eines Balancers und eines Clients, der Client oder Balancer darf nicht abstürzen wenn kein Server vorhanden ist:



```
C:\Windows\system32\cmd.exe

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
Client.jar 127.0.0.1 10
Leider steht noch kein Server zur Verfuegung!! :<
null

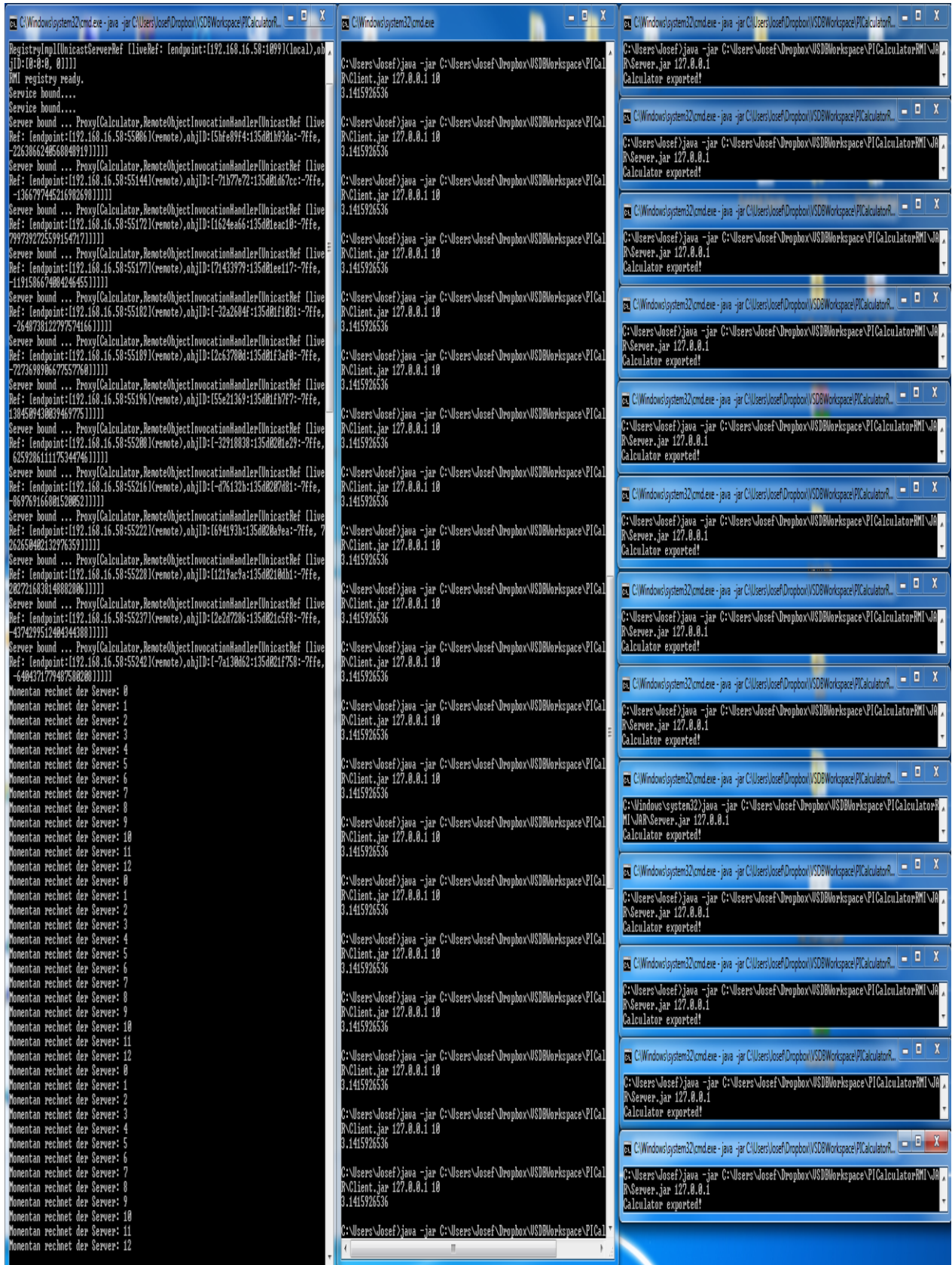
C:\Users\Josef>

Momentan rechnet der Server: 0

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Balancer.jar
RegistryImpl[UnicastServerRef [liveRef: [endpoint:[192.168.16.58:1099]<local>,ob
jid:[0:0:0, 0]]]]
RMI registry ready.
Service bound....
Service bound....
```

# Großer Test lokal mit mehr und weniger stellen

Es wurden 13 Server erstellt und einige Clients, es wird dabei nur ein Balancer benutzt



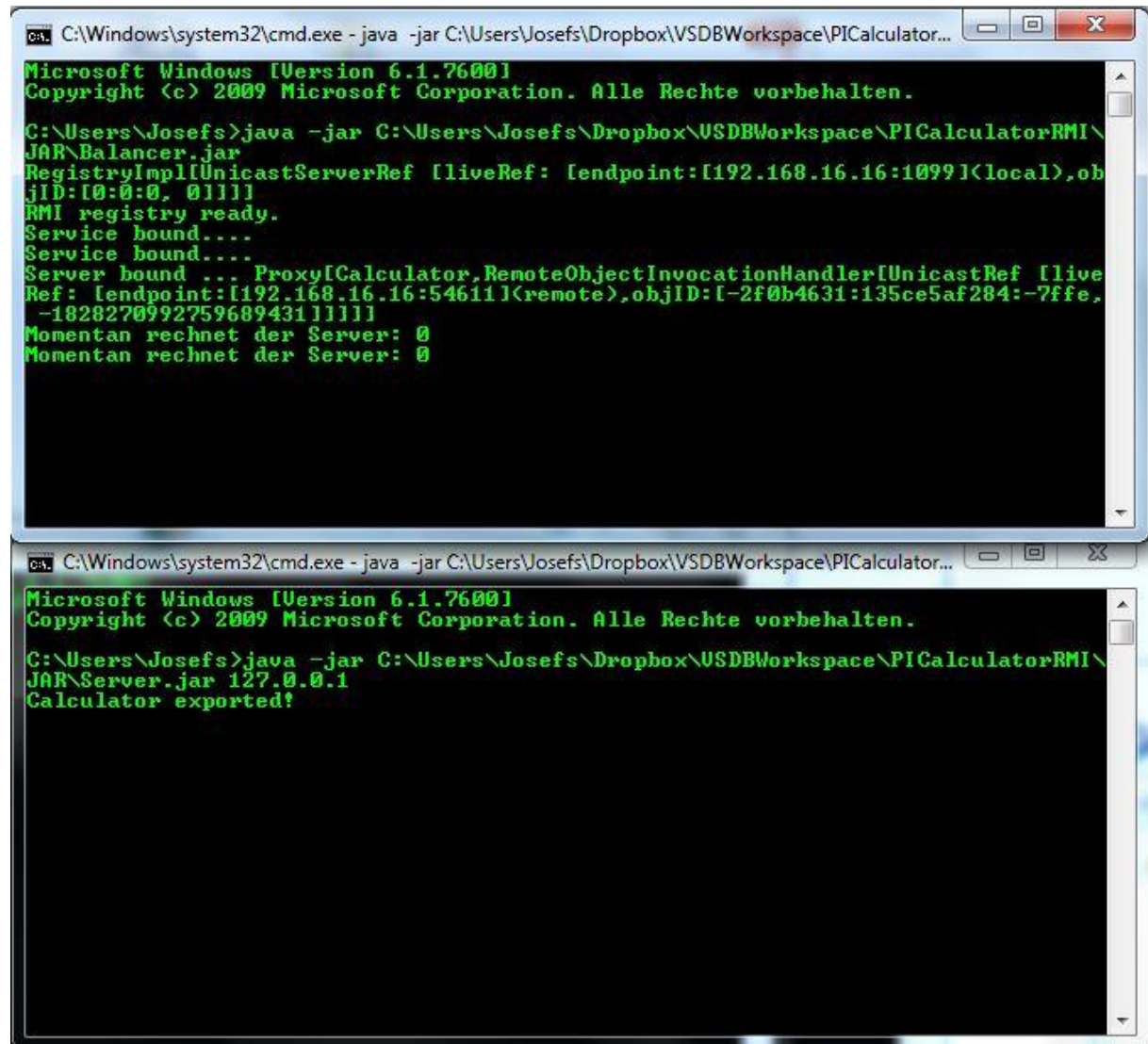


## Netzwerk Tests:

Durchführung mit 1 Client, 1 Server und dem Balancer: es soll 2-mal Pi berechnet werden Server und Balancer sind bei Sochovsky Laptop und die Client bei Sochovsky PC

```
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Client.jar 192.168.16.16 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JA
R\Client.jar 192.168.16.16 10
3.1415926536
```



The image shows two screenshots of a Windows command prompt window. The title bar of the window is "C:\Windows\system32\cmd.exe - java -jar C:\Users\Josefs\Dropbox\USDBWorkspace\PICalculator...".

The first screenshot shows the output of running the balancer jar:

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Alle Rechte vorbehalten.

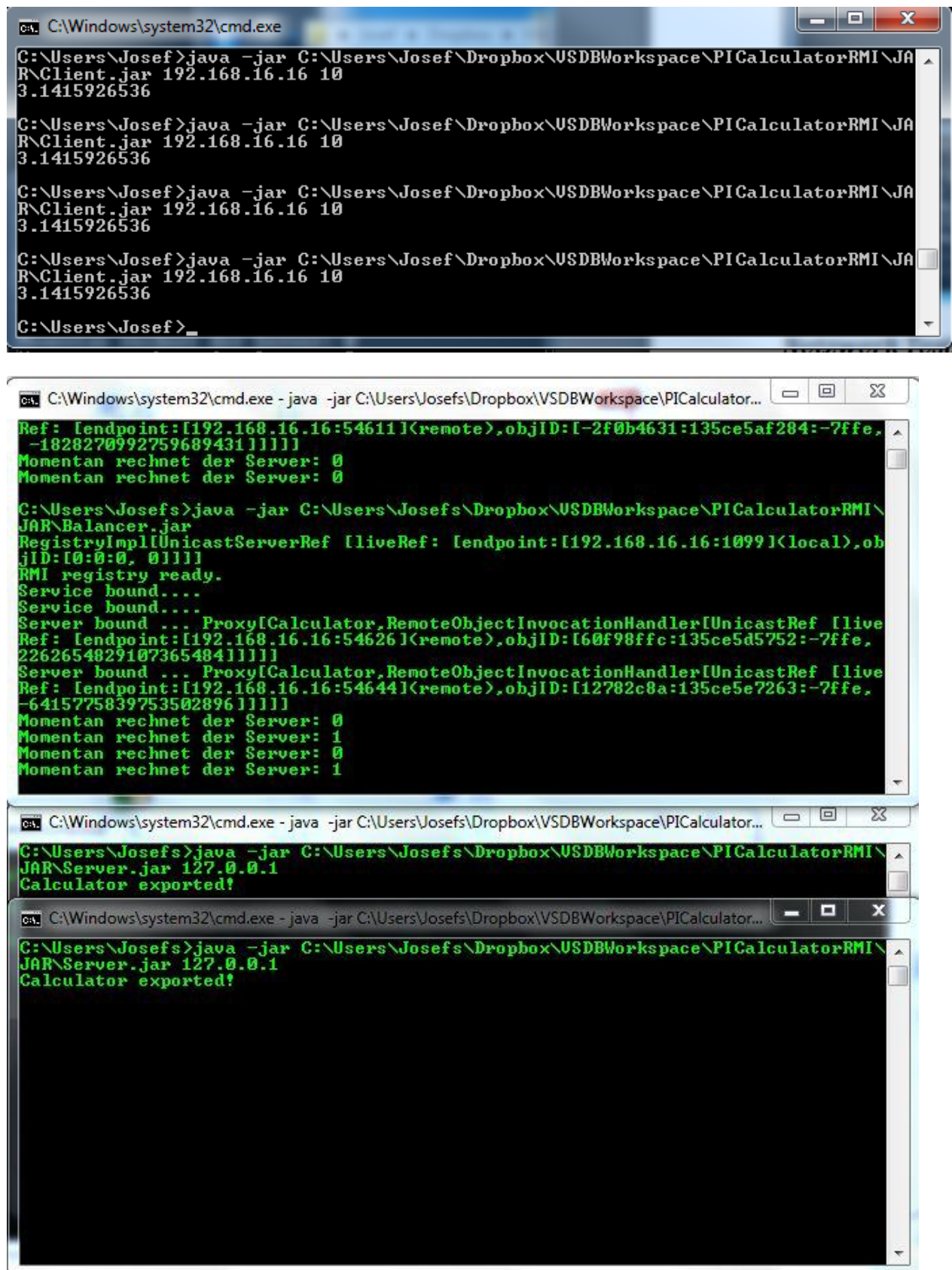
C:\Users\Josefs>java -jar C:\Users\Josefs\Dropbox\USDBWorkspace\PICalculatorRMI\
JAR\Balancer.jar
RegistryImpl[UnicastServerRef [liveRef: [endpoint:[192.168.16.16:1099]<local>,ob
jID:[0:0:0, 0]]]]
RMI registry ready.
Service bound....
Service bound....
Server bound ... Proxy[Calculator.RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[192.168.16.16:54611]<remote>,objID:[-2f0b4631:135ce5af284:-7ffe,
-1828270992759689431]]]]]
Momentan rechnet der Server: 0
Momentan rechnet der Server: 0
```

The second screenshot shows the output of running the server jar:

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Josefs>java -jar C:\Users\Josefs\Dropbox\USDBWorkspace\PICalculatorRMI\
JAR\Server.jar 127.0.0.1
Calculator exported!
```

Durchführung mit 2 Client, 2 Server und dem Balancer: es soll 4-mal Pi berechnet werden Server und Balancer sind bei Sochovsky Laptop und die Client bei Sochovsky PC



```
C:\Windows\system32\cmd.exe
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\Client.jar 192.168.16.16 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\Client.jar 192.168.16.16 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\Client.jar 192.168.16.16 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\Client.jar 192.168.16.16 10
3.1415926536

C:\Users\Josef>_

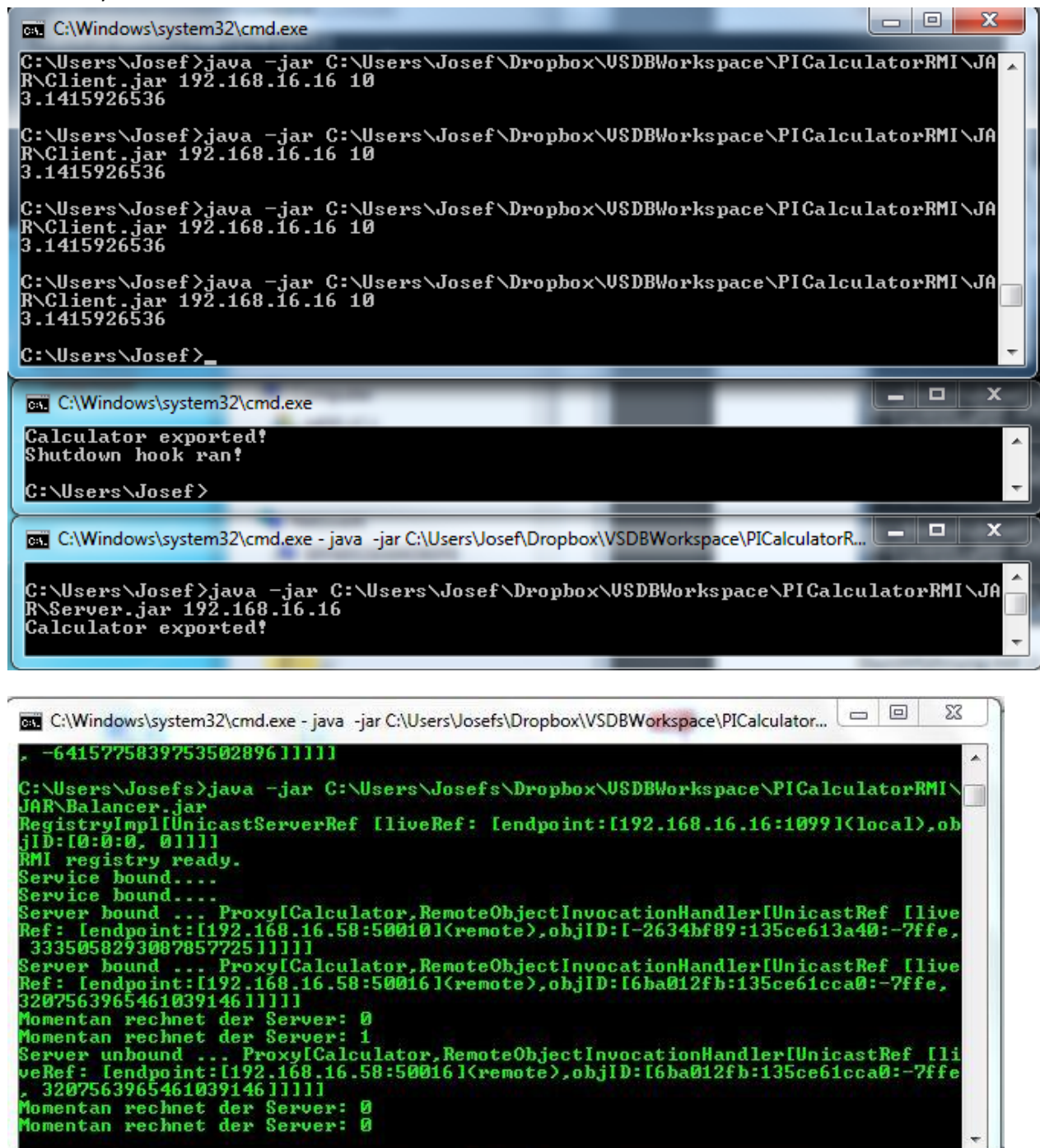
C:\Windows\system32\cmd.exe - java -jar C:\Users\Josefs\Dropbox\VSDBWorkspace\PICalculator...
Ref: [endpoint:[192.168.16.16:54611]<remote>],objID:[-2f0b4631:135ce5af284:-7ffe,
-18282709927596894311111]
Momentan rechnet der Server: 0
Momentan rechnet der Server: 0

C:\Users\Josefs>java -jar C:\Users\Josefs\Dropbox\USDBWorkspace\PICalculatorRMI\
JAR\Balancer.jar
RegistryImpl[UnicastServerRef [liveRef: [endpoint:[192.168.16.16:1099]<local>],ob
jID:[0:0:0, 0111]
RMI registry ready.
Service bound....
Service bound....
Server bound ... Proxy[Calculator.RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[192.168.16.16:54626]<remote>],objID:[60f98ffc:135ce5d5752:-7ffe,
22626548291073654841111]
Server bound ... Proxy[Calculator.RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[192.168.16.16:54644]<remote>],objID:[12782c8a:135ce5e7263:-7ffe,
-64157758397535028961111]
Momentan rechnet der Server: 0
Momentan rechnet der Server: 1
Momentan rechnet der Server: 0
Momentan rechnet der Server: 1

C:\Windows\system32\cmd.exe - java -jar C:\Users\Josefs\Dropbox\VSDBWorkspace\PICalculator...
C:\Users\Josefs>java -jar C:\Users\Josefs\Dropbox\USDBWorkspace\PICalculatorRMI\
JAR\Server.jar 127.0.0.1
Calculator exported!

C:\Windows\system32\cmd.exe - java -jar C:\Users\Josefs\Dropbox\VSDBWorkspace\PICalculator...
C:\Users\Josefs>java -jar C:\Users\Josefs\Dropbox\USDBWorkspace\PICalculatorRMI\
JAR\Server.jar 127.0.0.1
Calculator exported!
```

Durchführung mit 2 Client, 2 Server und dem Balancer: es soll 4-mal Pi berechnet werden – Der zuerst erstellte Server wird geschlossen und es soll kein Fehler auftreten (RoundRobin soll hier keinen Fehler produzieren) ... Balancer sind bei Sochovsky Laptop und die Clients und Server bei Sochovsky PC



```
C:\Windows\system32\cmd.exe
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Client.jar 192.168.16.16 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Client.jar 192.168.16.16 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Client.jar 192.168.16.16 10
3.1415926536

C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Client.jar 192.168.16.16 10
3.1415926536

C:\Users\Josef>_

C:\Windows\system32\cmd.exe
Calculator exported!
Shutdown hook ran!
C:\Users\Josef>

C:\Windows\system32\cmd.exe - java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorR...
C:\Users\Josef>java -jar C:\Users\Josef\Dropbox\USDBWorkspace\PICalculatorRMI\JAR\
R\Server.jar 192.168.16.16
Calculator exported!

C:\Windows\system32\cmd.exe - java -jar C:\Users\Josefs\Dropbox\USDBWorkspace\PICalculator...
-641577583975350289611111
C:\Users\Josefs>java -jar C:\Users\Josefs\Dropbox\USDBWorkspace\PICalculatorRMI\
JAR\Balancer.jar
RegistryImpl[UnicastServerRef [liveRef: [endpoint:[192.168.16.16:10991]local],obj
jid:[0:0:0, 0111]
RMI registry ready.
Service bound....
Service bound....
Server bound ... Proxy[Calculator,RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[192.168.16.58:50010]remote],objID:[-2634bf89:135ce613a40:-7ffe,
33350582930878577251111]
Server bound ... Proxy[Calculator,RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[192.168.16.58:50016]remote],objID:[6ba012fb:135ce61cca0:-7ffe,
32075639654610391461111]
Momentan rechnet der Server: 0
Momentan rechnet der Server: 1
Server unbound ... Proxy[Calculator,RemoteObjectInvocationHandler[UnicastRef [li
veRef: [endpoint:[192.168.16.58:50016]remote],objID:[6ba012fb:135ce61cca0:-7ffe,
32075639654610391461111]
Momentan rechnet der Server: 0
Momentan rechnet der Server: 0
```



## Quellen

An Overview of RMI Applications, Oracle Online Resource,  
<http://docs.oracle.com/javase/tutorial/rmi/overview.html> (last viewed 16.02.2012)

Creating a Client Program  
<http://docs.oracle.com/javase/tutorial/rmi/client.html> (last viewed 01.03.2012)

Calling function when program exits in java  
<http://stackoverflow.com/questions/63687/calling-function-when-program-exits-in-java> (last viewed 01.03.2012)

RMI – Verteilte Programmierung unter JAVA  
<http://www.cul.de/data/jbuilderpr.pdf> (last viewed 01.03.2012)