Remote Method Invocation

- Aufruf von Methoden über die Grenzen der VM hinweg.
- Javaprozesse der rufenden und gerufenen Methode können auf verschiedenen Hosts laufen.
- Eine RMI-Applikation besteht aus dem Client, der den Aufruf einer remote Method enthält und der Implementation, die die Methode bereitstellt sowie einem Interface, in dem die remote Methoden deklariert sind.
- Über einen bind-Mechnismus erfolgt die Bindung zur Laufzeit mit Hilfe eines rmiregistry Prozesses, der implementationsseitig (Server) laufen muss.

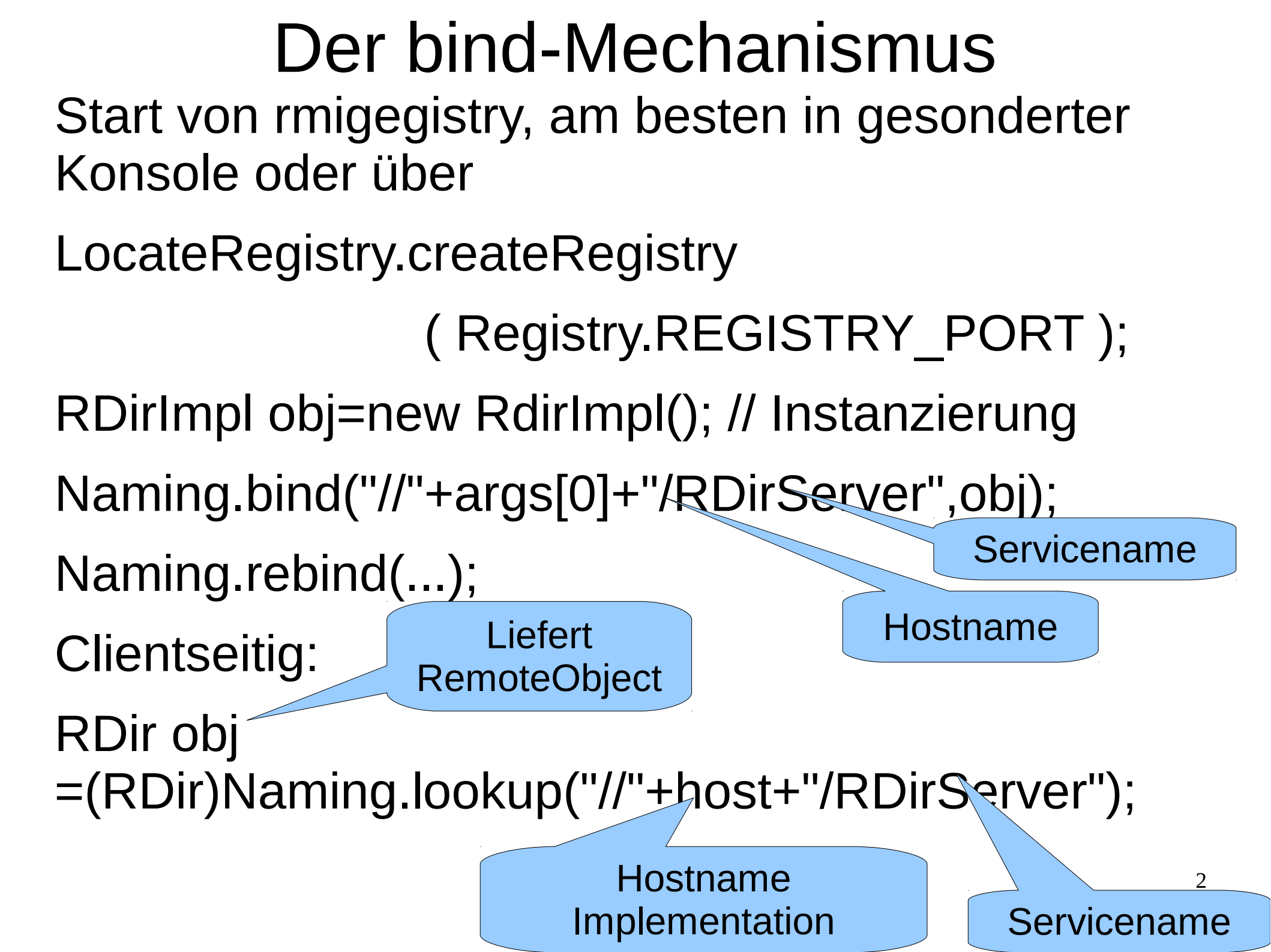
Der bind-Mechanismus

Start von rmigegistry, am besten in gesonderter Konsole oder über

```
LocateRegistry.createRegistry
                    ( Registry.REGISTRY_PORT );
RDirImpl obj=new RdirImpl(); // Instanzierung
Naming.bind("//"+args[0]+"/RDirServer",obj);
Naming.rebind(...);
```

Clientseitig:

```
RDir obj
=(RDir)Naming.lookup("//"+host+"/RDirServer");
```



Rmiregistry

- Ab Vesion 1.6 kann die registry auch aus aus dem Server (Implementation) gestartet werden
- LocateRegistry.createRegistry()
- try-catch erforderlich
- Import java.rmi.registry.*;

Das Remote Object

- Das remote Object auf der Serverseite implementiert das Remote Interface, das alle Funktionen, die remote ausgeführt werden sollen deklariert.

```
import java.util.*;
public interface RDir extends java.rmi.Remote
{
  public Vector<String> getDir(String Dir)
                  throws java.rmi.RemoteException;
}
```

- Für den Client ist das remote Object vom Typ des remote Interfaces.

Eine RMI-Applikation step by step
1. Das remote Interface

```
import java.util.*;

public interface RmiCalc extends java.rmi.Remote
{
  public double calc(String Calculation)
                  throws java.rmi.RemoteException;
}
```

Das Remote Interface muss Client- und Implementationsseitig als .class vorhanden sein.

2. Die Implementation (1)

```
import java.util.*;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.*;
Import java.rmi.registry.*;

public class RmiCalcImpl extends UnicastRemoteObject
                         implements RmiCalc
{
 public RmiCalcImpl() throws RemoteException
 {
   super();
 }
 public double calc(String Calculation)
                                 throws RemoteException
 {
   return new ExprCalc(Calculation).getResult();
 }
  . . .
}
```

DefaultConstructor muss sein, auch wenn er nix macht!

Hier wird gerechnet!!

2. Die Implementation (2)

```
public static void main(String args[])
{
 System.setSecurityManager(new RMISecurityManager()
 {
  public void checkConnect(String host,int port){}
  public void checkAccept(String host, int port){}
 });
 try
 {
  LocateRegistry.createRegistry()
  RmiCalcImpl obj=new RmiCalcImpl();
  Naming.rebind("//"+args[0]+"/RmiCalcImpl",obj);
  System.out.println("RmiCalcImpl bound in registry");
 }
 catch (Exception e)
 {
  System .out.println("HelloImpl err: "+e.getMessage());
  e.printStackTrace();
 }
}
```

RMISecurityManager muss sein

Anmelden bei rmiregistry
bind: einmalig, rebind: wiederholt

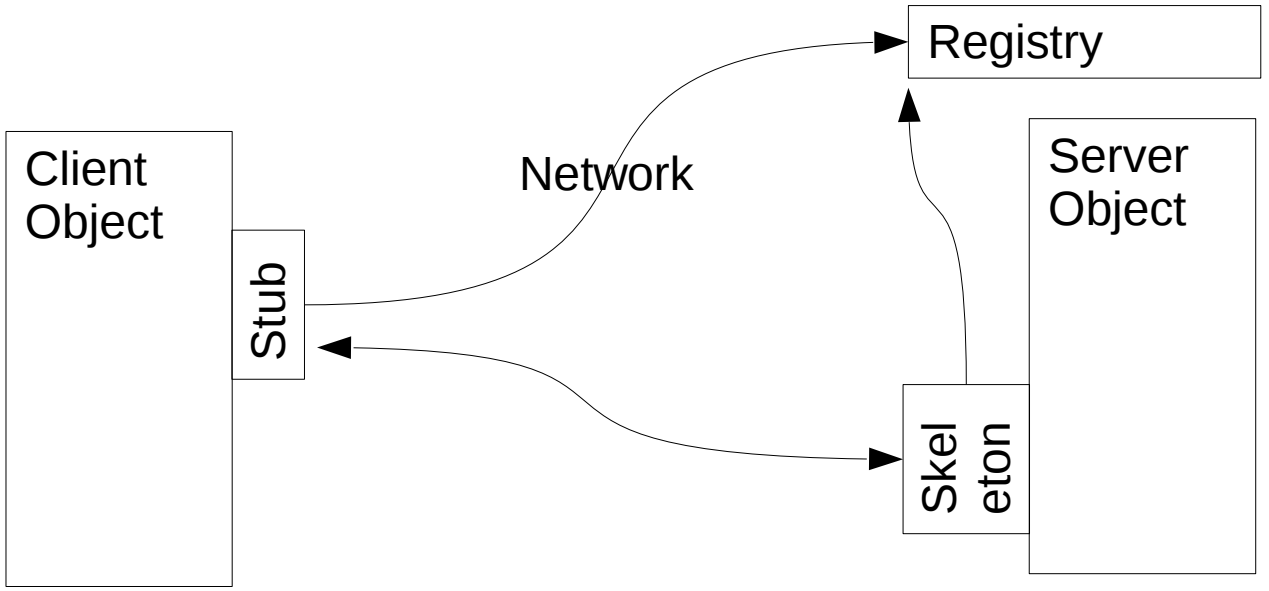
Der Client

```
import java.io.*;
import java.util.*;
import java.awt.*;
import java.rmi.*;

public class RmiCalcCl
{
  public RmiCalcCl(String host, String Calculation)
  {
     try
     {
        RmiCalc obj =(RmiCalc)Naming.lookup("//"+host+"/RmiCalcImpl");
        double d=obj.calc(Calculation);
          System.out.println("-> "+d);
     }
     catch(Exception e)
     {
        System.out.println("RDirException:\n"+e);
        e.printStackTrace();
     }
  }
  public static void main(String args[])
  {
     RmiCalcCl H=new RmiCalcCl(args[0],args[1]);
  }
}
```

Remote binding
liefert ein Objekt des Remote Interface

Und wie funktioniert's?



Parameterübergabe / Returnwert

- Parameter/Returnwerte werden bei RMI immer als Kopie übergeben, im Gegensatz zu gewöhnlichen Methodenaufrufen, bei denen nur die Referenzen von Objekten übergeben werden.
- Parameter/Returnwerte werden serialisiert und als Datenstrom über das Netzwerk übertragen. Dies erfolgt nicht über Port 1099, der dient nur der Vermittlung über rmiregistry.
- Setzt voraus, dass Client und Implementation unter einer verträglichen java-Version laufen (dieselbe Major Version)

RMI und Firewall

- das Öffnen der Ports 1099 und 1098 reicht nicht.
- Port 1099 ist der Port für die RMIRegistry, der Datenaustausch geschieht über einen anderen Port.
- Mittels super(port); im Construktor des RMI-Servers kann dieser andere Port festgeschrieben werden, zB. 9000.
- Dieser Port kann dann in der Firewall freigeschaltet werden.
- In /etc/hosts müssen die Hostnamen mit der IP-Adresse korrekt eingetragen sein, nicht als localhost (127.0.0.1).