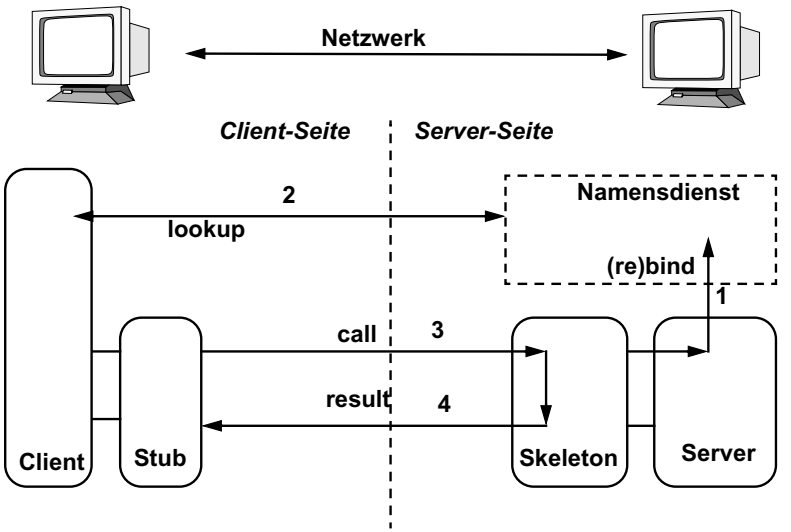
Remote Method Invocation

- **Spezielle Technik aus dem Java-Umfeld**
- **Ausführung von Methoden auf einem entfernten Rechner**
- **Analogon zum RPC (Remote Procedure Call)**
- **Zweck:**
  - **Objekte in verschiedenen Java-VMs**
  - **Aufruf einer Methode eines "remote"-Objektes**
  - **Programmieren wie mit lokalen Objekten**
  - **wenig Programmier-Overhead**
  - **verteilte Applikationen**

Architektur



Server und Client

Server:

- **stellt Objekte zum entfernten Aufruf bereit (Export)**
- **ist mit einem Ort (Internet-Adresse) assoziiert**

Client:

- **muss Typ der vom Server exportierten Objekte kennen**
- **muss Ort des Servers kennen**

Namensdienst

Gestattet Clients das Auffinden von Objekten,die von Servern bereitgestellt werden

- **Server-Seite:**

//entferntes Objekt anmelden (existierende Einträge überschreiben)
Naming.rebind("ObjectName", RemoteObject)

//entferntes Objekt abmelden
Naming.unbind("ObjectName", RemoteObject)

- **Client-Seite:**

//Referenz (Stub) des entfernten Objektes erfragen
Naming.lookup("rmi://some.server.com/ObjectName")

Namensdienst: Details

- Katalog von verfügbaren Remote-Objekten
- enthält Referenzen auf Remote-Objekte
- Referenzen über Namen identifizierbar
- Namen sind wie URLs zusammengesetzt → rmi://rechner:port/objektname
- Naming-Methoden:
  - void bind(String url, Remote reference)
  - void rebind(String url, Remote reference)
  - void unbind(String url)
  - String[] list(String url)
  - Remote lookup(String url)

RMI-Benutzung

- Schritt 1: Implementierung der Server-Seite
  - Basis-Interface und -Implementation
  - Server-Implementation
- (Schritt 2: Stub und Skeleton generieren)
  - automatische Generierung mittels *rmic*
  - Mit Java 1.5 und neuer nicht mehr nötig
- Schritt 3: Implementierung der Client-Seite
  - Anfrage im Repository nach dem entfernten Server
  - Methoden-Aufruf wie auf dem lokalen Objekt
- Schritt 4: Namensdienst aktivieren
  - Starten des Namensdienstes mit *rmiregistry*
- Schritt 6: Erst Server, dann Client starten

RMI-Beispiel: Objekt

Deklaration der entfernt zugreifbaren Methoden in einem Basis-Interface

```
// MyObject.java
import java.rmi.*;

interface MyObject extends Remote {
    public void do_something() throws RemoteException, UserException;
}
```

Implementation

```
// MyObjectImpl.java
import java.rmi.*;
import java.rmi.server.*;

public class MyObjectImpl extends UnicastRemoteObject implements MyObject {
    public MyObjectImpl() throws RemoteException {
        super();
    }
    public do_something() throws RemoteException, MyException {
        System.out.println("Hello RMI World!");
        throw new MyException("Hello RMI Execptions");
    }
}
```

Parallele Ausführung von Methoden mit Threads!

RMI-Beispiel: Exception / Server

Benutzer-definierte Exception

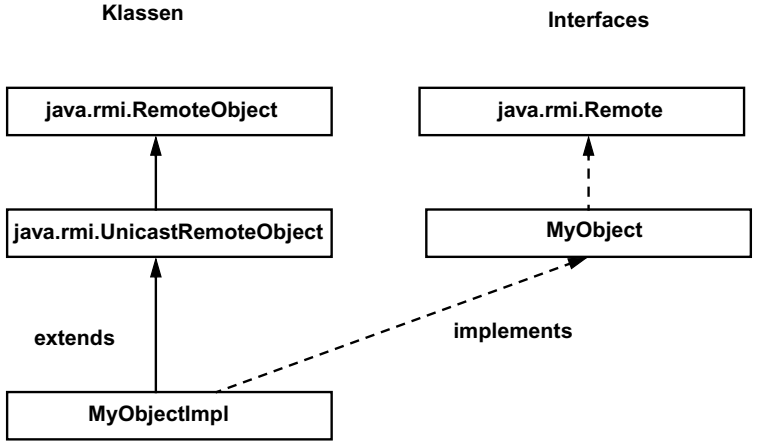
```
// MyException.java
public class MyException extends Exception {
    public MyException(String text) {
        super(text);
    }
};
```

Server

```
// Server.java
import java.rmi.*;
import java.rmi.server.*;

public class Server {
    public static void main (String args[]) {
        try {
            MyObjectImpl obj = new MyObjectImpl();
            Naming.rebind("MyObject", obj);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

RMI-Klassen und -Interfaces



(RMI-Beispiel: Stub und Skeleton)

Für ältere Java-Versionen (vor 1.5) werden Stub und Skeleton wie folgt erzeugt:

- **ServerImpl.java compilieren**
- **Für automatische Erzeugung wird Tool *rmic* verwendet**

  > rmic ServerImpl

- **Dabei entstehen folgende Klassen:**

  ServerImpl_Stub.class
  - Stub für die Client-Seite

  ServerImpl_Skel.class
  - Skeleton für die Server-Seite

RMI-Beispiel: Namensdienst

- **Verzeichnis der zur Verfügung gestellten entfernten Objekte**
- **spezielles Tool *rmiregistry*:**

  > rmiregistry <port> &

- **Namensdienst wird gestartet**
- **Portnummer ist ein optionaler Parameter (per Definition wird 1099 eingestellt).**
- **Fehlermeldung falls Port bereits von einem anderen Prozess verwendet wird**

RMI-Beispiel: Client

- **Keine zusätzliche Vererbung von Interfaces und Objektklassen nötig**
- **Client findet den Server, indem er sich an den Namensdienst in dem Server-Rechner wendet**
- **Bei Implementation assoziierter Name wird für die Nachfrage benutzt**

```
import java.rmi.*;

public class Client {
    public static void main (String args[]) {
        try {
            MyObject obj = (MyObject)Naming.lookup("rmi://some.server.com:port/MyObject");
            object.do_something();
        }
        catch (Exception e) {
            System.out.println(e)
        }
    }
}
```

Zusammenfassung

- **RMI bietet Methode zum Verstecken von Verteilung**
- **vereinfacht das Programmieren**
- **aber: neue Fehlerquellen + Exceptions**
- **(fast) gleiche Behandlung von lokalen und Remote-Objekten**
- **Remote-Objekt fungiert als Server (exportiert Methoden)**
- **Client kann über ein Interface auf das Remote-Objekt zugreifen**
- **RMI-Paket bietet einfachen Naming-Service**
- **Parallele Ausführung von Methoden mittels Threads (Synchronisation!)**