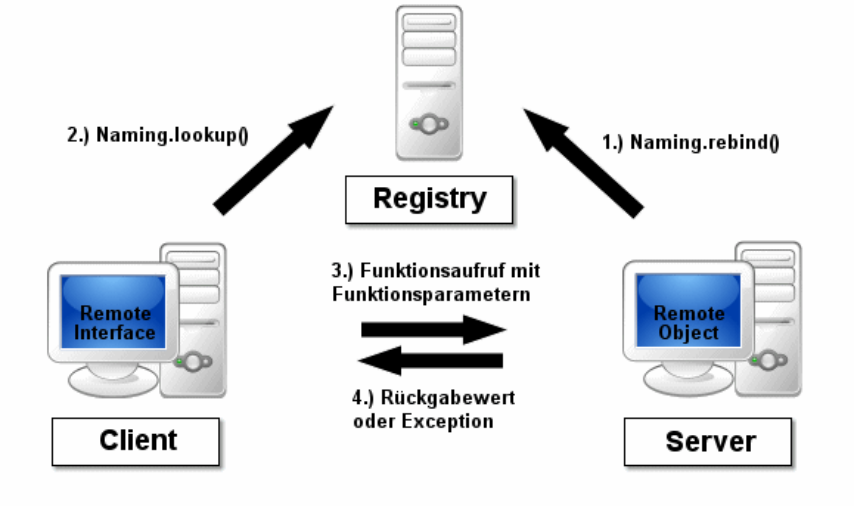
Protokoll

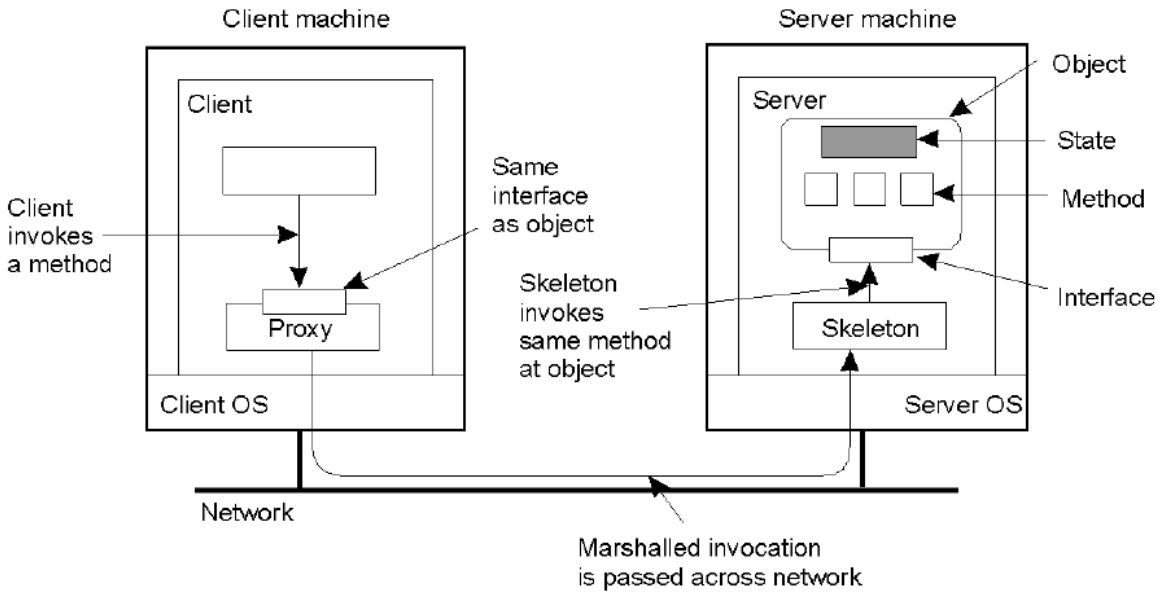
1)

- Programm-Beispiel wurde nicht programmiert. Die Theorie ist vollständig.
- Remote Method Invocation (RMI), ist der Aufruf einer Methode eines entfernten Java-Objekts und realisiert die Java-eigene Art des Remote Procedure Call.
- „Entfernt“ bedeutet dabei, dass sich das Objekt in einem anderen Java Virtual Machine befinden kann (entfernten Rechner oder auch lokalen Rechner).
- Es müssen jedoch besondere Ausnahmen (Exceptions) abgefangen werden, die zum Beispiel einen Verbindungsabbruch signalisieren können.
- Auf der Client-Seite kümmert sich der sogenannte Stub um den Netzwerktransport.
- Der Stub muss entweder lokal oder über das Netz für den Client verfügbar sein.
- RMI bezeichnet außerdem ein auf TCP/IP basierendes Kommunikationsprotokoll.
- Komponente:
    - Remote Interface:
        - Beschreibt die Funktionen, die auf dem Server zur Verfügung stehen, und definiert damit das Verhalten des entfernten.
    - Remote Object:
        - Stellt das entfernte Objekt dar und liegt auf dem Server.
        - Es implementiert das Remote Interface und das Verhalten der für die Clients zur Verfügung stehenden entfernten Methoden.
        - Vom Server können eine oder mehrere Instanzen des Remote-Objekts erstellt werden.
        - Ein Remote Object muss von UnicastRemoteObject abgeleitet sein und muss einen parameterlosen Konstruktor haben, denn dieser ruft nur den Konstruktor von UnicastRemoteObject auf und könnte sonst eine RemoteException auslösen.
        - Jede Methode muss eine RemoteException deklarieren, auch der parameterlose Konstruktor.
    - Remote Reference:
        - Ist eine Referenz auf Remote Objects.
        - Die Clients bekommen die Remote Reference von der RMI Registry.
- Ablauf:
    1. Der Server registriert ein Remote Object bei der RMI-Registry unter einem eindeutigen Namen.
    2. Der Client sieht bei der RMI-Registry unter diesem Namen nach und bekommt eine Objektreferenz, die seinem Remote Interface entsprechen muss.
    3. Der Client ruft eine Methode aus der Objektreferenz auf. Dabei kann ein Objekt einer Klasse X übergeben werden, die der JVM des Servers bisher nicht bekannt ist (das ist möglich, wenn X ein dem Server bekanntes Interface implementiert). In diesem Fall lädt die Server-JVM die Klasse X dynamisch nach, beispielsweise vom Webserver des Client.
    4. Die Server-JVM führt die Methode auf dem Remote Object aus, wobei evtl. dynamisch geladener Fremdcode benutzt wird (z. B. Methoden von X), der

im Allgemeinen Sicherheitsrestriktionen unterliegt. Dem Client werden die Rückgabewerte dieses Aufrufes gesendet, oder der Client bekommt eine Fehlermeldung (z. B. bei einem Verbindungsabbruch).



Quelle: https://de.wikipedia.org/wiki/Remote_Method_Invocation



- Ein **Stub** bezeichnet in der Softwareentwicklung einen Programmcode, der anstelle eines anderen Programmcodes steht.
    - Dabei ist der Programmcode, den der Stub ersetzt, entweder noch nicht entwickelt (Top-Down) oder er ist auf einem anderen Rechner oder in einem anderen Speicherbereich.
- **Skeleton** wird im Bereich Programmierung für eine automatisch generierte Struktur (häufig Quelltext) verwendet, die ein Programmierer oder Benutzer dann ausbauen kann.

- Ein Skeleton wird auch als Stub bezeichnet und bildet den Gegenpunkt zu dem Stub auf der Client-Seite (auch Proxy genannt).
- Codebase:
    - Datenbank, welche Implementierungen von Klassen austeilt.
    - Der Client enthält nur die Definition der Interfaces.
- Hostname:
    - Ein String, welcher zusammen mit den Stubs die Methoden remote ausführbar macht.
- useCodebaseOnly:
    - Wird genutzt, wenn Server oft geupdatet werden muss,
        - Clients müssen so nicht neu ausgeteilt werden
- Java Policy File:
    - Beschreibt die benötigten Berechtigungen, z.B. Netzwerk für RMI.
    - Der SecurityManager bietet Einstellungen zu Berechtigungen und Zugang.
- SecurityManagers:
    - Vorteile:
        - Sicherheit vor fremdem Code
        - Sicherheit vor unbefugtem Zugriff
    - Nachteile:
        - Konfigurationsaufwand
- Ausführung:
    1. RMI-Registry ausführen
    2. Server starten
    3. Client starten

2)

- Public Interface Remote:
    - Das Interface ist leer
    - Dient zur Markierung der Klassen welche RMI nutzen
- Bei der Aufgabe 2a wird für die Klasse ein Interface definiert, in Aufgabe 2b ist das Interface generisch.
- Da es nicht gefragt worden ist, kann der Server nur ein Client auf einmal abarbeiten.
    - Mit Tasks könnte man mehrere Clients abarbeiten.
- Note: 8

3)

- Callback:
    - Verwendet gleiche Mechanismen wie RMI
    - Client deklariert Objekt mit der Callback-methode
        - Implementiert Remote
        - Extends UnicastRemoteObject
        - Nichts wird auf eine Registry veröffentlicht
    - Client sendet Objekt mit Callback an den Server
        - Server speichert Objekt
    - Server:
        - Verwendet Callback wenn‘s braucht
    - Vorteile:
        - Symmetrisch
        - Nicht extra Anmeldung des Callback-Objekts an Registry
    - Nachteile:
        - Komplizierter als eine „Method-Call“
        - Braucht extra Struktur auf dem Server
        - Rechte müssen neu vergeben werden, da beide Peers als Server und Client fungieren
- Code:

```
public class AntwortImpl extends UnicastRemoteObject implements Antwort {

    public AntwortImpl() throws RemoteException {...}

    public void Antwort(String i, Client c) {
        try {
            Thread.sleep(10); // 10 nur für Tests verwändet.

        } catch (Exception e) {
        }
        c.Ausgeben("Die Antwort auf  <" + i + "> ist wahrscheinlich 61!");
    }

}
```

➔ Thread.sleep muss geändert werden, damit die Aufgabe vollständig ist.

- Note: 8