Benennung der Objekte

Grundsätzlich zwei Möglichkeiten:

- Nutzung der schon vorhandenen anwendungsspezifischen Namen (Banknummer, Kontonummer)
    - Nachteil: Abhängigkeit vom Anwendungsszenario
    - Vorteil: keine zusätzlichen Namen
- Einführung eines separaten Namensschemas ("Objekt1", "Objekt2", ...)
    - Nachteil: zusätzliche Namen
    - Vorteil:Unabhängigkeit vom Anwendungsszenario

Protokoll

- Request (Anfrage):
  - Objektname
  - Methodenname
  - Parameter des Methodenaufrufes
- Response (Antwort):
  - Exception-Flag
  - Falls Exception: Name und Datenelemente der Exception
  - Sonst: Ausgabeparameter und Ergebnis

Implementierung des Protokolls

Grundsätzlich zwei Möglichkeiten:

- Automatisch mittels `Serializable` und `Objekt[Input|Output]Stream`:

```
public class Request implements Serializable {
    String object_name;
    ...
};
ObjectOutputStream ostr = ...;
Request req = ...
ostr.writeObjekt (req);
```

- Von Hand mittels `Data[Input|OutputStream]`:

```
DataOutputStream ostr = ...;
Request req = ...;
ostr.writeUTF (req.object_name);
```

ClientStubs 1

Basisklasse `ClientStubBase` verwaltet Socket:

```
public class ClientStubBase {
    static Socket socket;

    public ClientStubBase (String rechner, int port) {
        if (socket == null)
            socket = new Socket (rechner, port);
    }
    public DataOutputStream output_stream () {
        return new DataOutputStream (socket.getOutputStream());
    }
    public DataOutputStream input_stream () {
        return new DataInputStream (socket.getInputStream());
    }
};
```

ClientStubs 2

Beispiel einer ClientStub-Methode:

```
public class BankClientStub extends ClientStubBase implements Bank {
    String object_name;
    public Konto erzeuge (String knr) throws BankException {
        DataInputStream istr = input_stream();
        DataOutputStream ostr = output_stream();

        ostr.writeUTF (object_name);
        ostr.writeUTF ("erzeuge");
        ostr.writeUTF (knr);
        ostr.flush();

        if (istr.readByte() == 0) {
            String konto_object_name = istr.readUTF();
            return new KontoClientStub (konto_object_name);
        }
        String exception_name = istr.readUTF();
        String error_msg = istr.readUTF();
        throw new BankException (error_message);
    } ...
```

ServerStubs 1

Abstrakte Basisklasse `ServerStubBase` deklariert `dispatch()`:

```
abstract public class ServerStubBase {
    abstract public void dispatch (String methode,
        DataInputStream istr,
        DataOutputStream ostr);
}
```

ServerStubs 2

Beispiel eines ServerStubs:

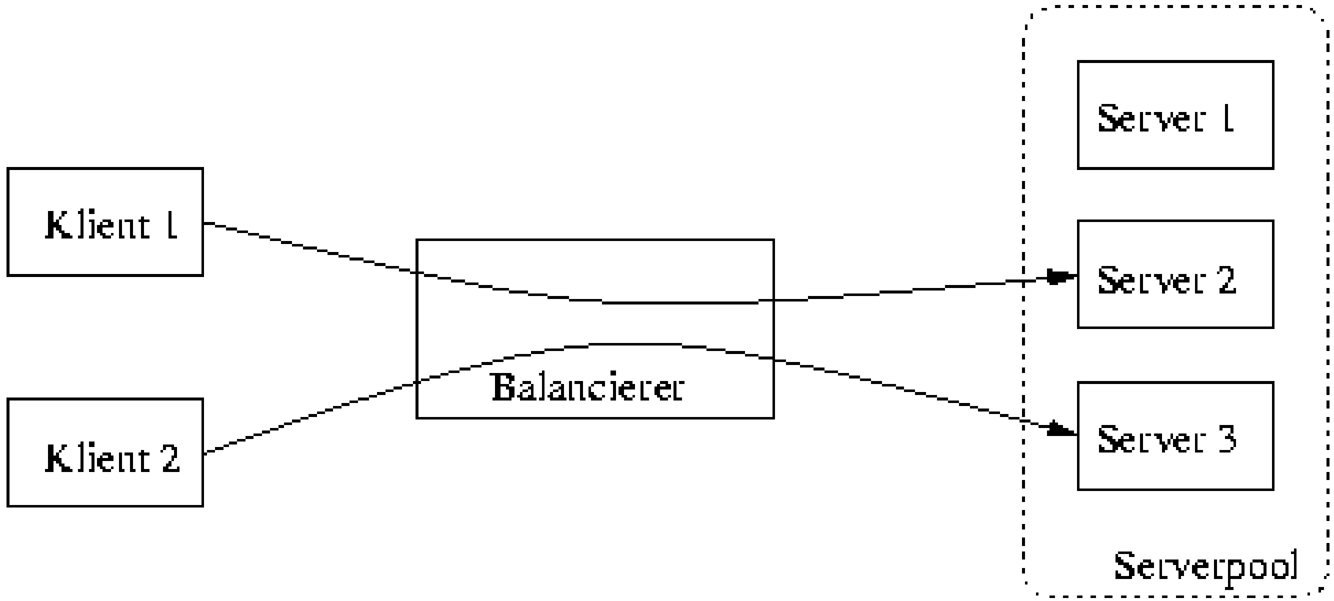
```
public class BankServerStub extends ServerStubBase {
    BankImpl bank_impl;
    void dispatch (String method, DataInputStream istr,
                   DataOutputStream ostr)
    {
        if (method.equals ("erzeuge")) {
            String knr = istr.readUTF();
            try {
                Konto konto = bank_impl.erzeuge (knr);
                ostr.writeByte (0);
                ostr.writeUTF ("konto" + knr);
            } catch (BankException e) {
                ostr.writeByte (1);
                ostr.writeUTF (e.getMessage());
            }
        } else if (method.equals (...)) {
        ...
```

Dispatcher 1

```
public class Dispatcher {
    Hashtable stubs;
    void connect (String name, ServerStubBase stub) {
        stubs.put (name, stub);
    }
    void disconnect (String name) {
        stubs.remove (name);
    }
    void run () {
        ServerSocket server = new ServerSocket (4444);
        while (true) {
            Socket socket = server.accept();
            DataInputStream istr =
                new DataInputStream (socket.getInputStream());
            DataOutputStream ostr =
                new DataOutputStream (ssocket.getOutputStream());
          process (istr, ostr);
      }
    }
```

```
void process (DataInputStream istr, DataOutputStream ostr) {
    while (true) {
        try {
            String object_name = istr.readUTF();
            ServerStubBase stub =
                (ServerStubBase)stubs.get (object_name);
            String methode = istr.readUTF();
            stub.dispatch (methode, istr, ostr);
        } catch (EOFException e) {
            return;
        }
    }
}
...
```

Lastbalancierung mittels RMI



- Round-Robin Verfahren im Balancierer