Verteilte Systeme FS 13
Übung 5

Thomas Baumann

20. Mai 2013

1 Beschreibung

Ich habe zwei neue Interfaces eingeführt, das einte Interface *IRemoteAccount* erweitert das Interface *IAccount* um das Marker Interface *java.rmi.Remote* und dementsprechend für die Bank das Interface *IRemoteBank*.

Die lokale Bank habe ich so abgeändert, dass beide Klassen die neu erstellten Interfaces implementieren und der Rückgabetyp von *getAccount(String number)* ebenfalls dem neuen Interface entspricht. Als letzte Änderung habe ich Zeile 45 eingefügt, damit alle Anforderungen für RMI abgedeckt sind.

Wie bereits die vorherigen Übungen, ist diese Lösung nicht Thread-Safe.

2 Code

Listing 1: Interface des Accounts für RMI

```java
package bank;

import java.rmi.Remote;

/**
 * This interface extends the IAccount Interface with the marker interface java.rmi.Remote
 *
 * @see IAccount, Remote
 * @author Thomas Baumann
 * @version 1.0
 */
public interface IRemoteAccount extends IAccount, Remote {

}
```

Listing 2: Interface der Bank für RMI

```java
package bank;

import java.rmi.Remote;

/**
 * This interface extends the IBank Interface with the marker interface java.rmi.Remote
 *
 * @see IBank, Remote
 * @author Thomas Baumann
 * @version 1.0
 */
public interface IRemoteBank extends IBank, Remote {

}
```

Listing 3: Lokale Bank

```java
package bank.local;

import java.io.IOException;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import bank.IAccount;
import bank.IBank;
import bank.IRemoteAccount;
import bank.IRemoteBank;
import bank.InactiveException;
import bank.OverdrawException;

/**
 * Implementation of the IBank interface with full functionality and the inner class
 * Account with the implementation of the IAccount interface.
 *
 * @see IBank
 * @see IAccount
 * @author Thomas Baumann
 * @version 1.1
 */
public class Bank implements IRemoteBank {

    private Map<String, Account> accounts = new HashMap<String, Account>();

    @Override
    public Set<String> getAccountNumbers() {
        Set<String> set = new HashSet<>();
        for (Account a : this.accounts.values()) {
            if (a.isActive()) {
                set.add(a.getNumber());
            }
        }
        return set;
    }

    @Override
    public String createAccount(String owner) throws RemoteException {
        Account ac = new Account(owner);
        UnicastRemoteObject.exportObject(ac, 0);
        this.accounts.put(ac.getNumber(), ac);
        return ac.getNumber();
    }

    @Override
    public boolean closeAccount(String number) {
        Account a = this.accounts.get(number);
        if (a != null && a.isActive() && a.getBalance() == 0.0) {
            a.active = false;
            return true;
        }
        return false;
    }

    @Override
    public IRemoteAccount getAccount(String number) {
        return this.accounts.get(number);
    }

    @Override
    public void transfer(IAccount from, IAccount to, double amount) throws IOException,
            InactiveException, OverdrawException {
        from.withdraw(amount);
        try {
            to.deposit(amount);
        } catch (Exception e) {
            from.deposit(amount);
            throw e;
        }
    }

    static class Account implements IRemoteAccount {
        private static int accountNumbers;
        private String number;
        private String owner;
        private double balance;
        private boolean active = true;

        Account(String owner) {
            this.owner = owner;
            this.number = Integer.toString(++Account.accountNumbers);

        }

        @Override
        public double getBalance() {
            return this.balance;
        }

        @Override
        public String getOwner() {
            return this.owner;
        }

        @Override
        public String getNumber() {
            return this.number;
        }

        @Override
        public boolean isActive() {
            return this.active;
        }

        @Override
        public void deposit(double amount) throws InactiveException {
            if (amount < 0) {
                throw new IllegalArgumentException("Amount can not be less then 0.");
            }
            if (!this.isActive()) {
                throw new InactiveException("Account is inactive.");
            }
            this.balance += amount;
        }

        @Override
        public void withdraw(double amount) throws IllegalArgumentException,
                InactiveException, OverdrawException {
            if (amount < 0) {
                throw new IllegalArgumentException("Amount can not be less then 0.");
            }
            if (!this.isActive()) {
                throw new InactiveException("Account is inactive.");
            }
            if (amount > this.getBalance()) {
                throw new OverdrawException("The account has to less amount.");
            }
            this.balance -= amount;
        }

    }
}
```

Listing 4: Server Driver

```java
package bank.rmi;

import java.io.IOException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

import bank.IRemoteBank;
import bank.IServerDriver;
import bank.StartServer;
import bank.local.Bank;

public class ServerDriver implements IServerDriver {

    private IRemoteBank bank = new Bank();

    @Override
    public void start(String[] args) throws IOException {
        if (args.length < 2) {
            System.out.println("Usage: java " + StartServer.class.getName() + " "
                    + ServerDriver.class.getName() + " <host> <portnumber>");
            System.exit(1);
        }
        String host = args[0];
        int port = 0;
        try {
            port = Integer.parseInt(args[1]);
        } catch (NumberFormatException e) {
            System.out.println("Port must be a number");
            System.exit(1);
        }

        try {
            LocateRegistry.createRegistry(port);
        } catch (RemoteException e) {
            System.out.println("registry could not be exported");
            System.exit(1);
        }

        UnicastRemoteObject.exportObject(this.bank, 0);

        Naming.rebind("rmi://" + host + ":" + port + "/Bank", this.bank);
    }

}
```

Listing 5: Client Driver

```java
package bank.rmi;

import java.io.IOException;
import java.rmi.Naming;
import java.rmi.NotBoundException;

import bank.IBank;
import bank.IBankDriver;
import bank.IRemoteBank;
import bank.StartClient;

public class ClientDriver implements IBankDriver {

    private IRemoteBank bank;

    @Override
    public void connect(String[] args) throws IOException {
        if (args.length < 2) {
            System.out.println("Usage: java " + StartClient.class.getName() + " "
                    + ClientDriver.class.getName() + " <host> <port>");
            System.exit(1);
        }
        try {
            bank = (IRemoteBank)Naming.lookup("rmi://"+args[0]+":"+args[1]+"/Bank");
        } catch (NotBoundException e) {
            System.out.println("Bank can not be found");
            System.exit(1);
        }
    }

    @Override
    public void disconnect() throws IOException {
        bank = null;
    }


    @Override
    public IBank getBank() {
        return bank;
    }

}
```