Uebung05 vesys

Daniel Gürber

4. Semester (FS 2013)

Inhaltsverzeichnis

1 Beschreibung

1.1 Kommunikation

Die Kommunikation läuft über RMI, dazu wurden die Interfaces RemoteBank, RemoteAccount und RemoteUpdateHandler erstellt. Die Bank und die Account Klasse implementieren diese Interfaces. Für den UpdateHandler wurrde eine Wrapper-Klasse geschrieben, da der vom GUI erstellte Handler kein Remote Objekt ist.

2 Code

2.1 Interfaces

Listing 1: RemoteBank

```java
package bank.rmi;

import java.io.IOException;
import java.rmi.Remote;

import bank.Bank;

public interface RemoteBank extends Bank, Remote {
  void registerUpdateHandler(RemoteUpdateHandler handler) throws IOException;
}
```

Listing 2: RemoteAccount

```java
package bank.rmi;

import java.rmi.Remote;

import bank.Account;

public interface RemoteAccount extends Account, Remote {}
```

Listing 3: RemoteUpdateHandler

```java
package bank.rmi;

import java.rmi.Remote;
import bank.BankDriver2;

public interface RemoteUpdateHandler extends Remote, BankDriver2.UpdateHandler {

}
```

2.2 Client

Listing 4: Driver

```java
package bank.rmi;

import java.io.IOException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import bank.Bank;
import bank.BankDriver2;
```

```java
public class Driver implements BankDriver2 {

  private RemoteBank bank;

  @Override
  public void connect(String[] args) throws IOException {
    try {
      bank = (RemoteBank)Naming.lookup(
          "rmi://localhost/bankService");
    } catch (NotBoundException e) {
      throw new IOException(e.getMessage());
    }

  }

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

  }

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

  @Override
  public void registerUpdateHandler(UpdateHandler handler) throws IOException {
    bank.registerUpdateHandler(new RemoteHandler(handler));
  }

  @SuppressWarnings("serial")
  static class RemoteHandler extends UnicastRemoteObject implements RemoteUpdateHandler {

    private UpdateHandler handler;

    public RemoteHandler(UpdateHandler handler) throws RemoteException {
      this.handler = handler;
    }

    @Override
    public void accountChanged(String id) throws IOException {
      handler.accountChanged(id);
    }

  }

}
```

2.3 Server

Listing 5: Server Klasse

```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 java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.UUID;
import java.util.concurrent.ConcurrentHashMap;

import bank.BankDriver2.UpdateHandler;
import bank.InactiveException;
import bank.OverdrawException;

public class Server {

  public static void main(String args[]) throws Exception {
    try {
      LocateRegistry.createRegistry(1099);
    }
    catch (RemoteException e) {
      System.out.println(">> registry could not be exported");
      System.out.println(">> probably another registry already runs on 1099");
    }

    RemoteBank bank = new Bank();
    Naming.rebind("bankService", bank);
  }

  @SuppressWarnings("serial")
  static class Bank extends UnicastRemoteObject implements RemoteBank {

    private final Map<String, Account> accounts = new ConcurrentHashMap<String, Account>();
    private final List<RemoteUpdateHandler> updateHandlers = new ArrayList<
        RemoteUpdateHandler>();

    public Bank() throws RemoteException {
    }

    @Override
    public void registerUpdateHandler(RemoteUpdateHandler handler) throws IOException {
      updateHandlers.add(handler);
    }

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

    @Override
    public String createAccount(String owner) throws IOException {
      Account newAccount = new Account(owner, updateHandlers);
      accounts.put(newAccount.number, newAccount);
      notifyHandlers(newAccount.number);
      return newAccount.number;
    }

    @Override
    public boolean closeAccount(String number) throws IOException {
      boolean ret=false;
      Account closeAccount = accounts.get(number);
      if (closeAccount != null ) {
        synchronized (closeAccount) {
          if (closeAccount.getBalance() == 0
              && closeAccount.isActive()) {
            closeAccount.active = false;
            ret = true;
          }
        }

        if (ret) {
          notifyHandlers(closeAccount.number);
        }
      }
      return ret;
    }
    @Override
    public Account getAccount(String number) {
      return accounts.get(number);
    }

    @Override
    public void transfer(bank.Account from, bank.Account to, double amount)
        throws IOException, InactiveException, OverdrawException {

      bank.Account first, second;
      if (from.getNumber().compareTo(to.getNumber())<0) {
        first = from;
        second = to;
      } else {
        first = to;
        second = from;
      }

      synchronized(first) {
        synchronized(second) {
          if (!from.isActive()) {
            throw new InactiveException("Source account is closed!");
          }

          if (!to.isActive()) {
            throw new InactiveException("Target account is closed!");
          }

          from.withdraw(amount);
          to.deposit(amount);
        }
      }
    }

    private void notifyHandlers(String id) throws IOException {
      for (UpdateHandler handler : updateHandlers) {
        handler.accountChanged(id);
      }
    }
  }

  @SuppressWarnings("serial")
  static class Account extends UnicastRemoteObject implements RemoteAccount {
    private final String number;
    private final String owner;
    private volatile double balance;
    private volatile boolean active = true;
    private final List<RemoteUpdateHandler> updateHandlers;

    Account(String owner, List<RemoteUpdateHandler> updateHandlers) throws IOException {
      this.owner = owner;
      this.number = UUID.randomUUID().toString();
      this.updateHandlers = updateHandlers;
    }

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

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

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

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

    @Override
    public void deposit(double amount) throws InactiveException, IllegalArgumentException,
        IOException {
      synchronized(this) {
        if (!this.active) {
          throw new InactiveException("Account is closed!");
        }

        if (amount < 0) {
          throw new IllegalArgumentException("Amount can't be negative!");
        }
        this.balance += amount;
      }
      notifyHandlers();
    }

    @Override
    public void withdraw(double amount) throws InactiveException, OverdrawException,
        IllegalArgumentException, IOException {
      synchronized(this) {
        if (!this.active) {
          throw new InactiveException("Account is closed!");
        }

        if (amount < 0) {
          throw new IllegalArgumentException("Amount can't be negative!");
        }

        if (amount > balance) {
          throw new OverdrawException("Not enough money on account!");
        }

        this.balance -= amount;
      }
      notifyHandlers();
    }

    private void notifyHandlers() throws IOException {
      for (UpdateHandler handler : updateHandlers) {
        handler.accountChanged(this.number);
      }
    }

  }
}
```