Uebung01 vesys

Daniel Gürber

4. Semester (FS 2013)

Inhaltsverzeichnis

1 Beschreibung

1.1 Architektur

Als Client wird die zur Verfügung gestellte Java-Applikation mit eigenem Driver verwendet. Der Server ist in C# geschrieben.

1.2 Kommunikation

Die Kommunikation verläuft so, das der Client eine Linie als Nachricht sendet und eine Linie als Antwort zurückerhält. Die einzelnen Argumente werden mit einem Doppelpunkt getrennt, vorhandene Doppelpunkte werden escaped als "[colon]". Wenn die Argumente nicht den Vorbedingungen entsprechen wird trotzdem eine Nachricht mit einem Fehler gesendet.

2 Code

2.1 Client

Listing 1: Driver

```
1  package bank.sockets;

   import java.io.BufferedReader;
   import java.io.IOException;
   import java.io.InputStreamReader;
6  import java.io.OutputStreamWriter;
   import java.io.PrintWriter;
   import java.net.Socket;
   import java.nio.charset.Charset;
   import java.util.HashSet;
11 import java.util.Set;

   import bank.InactiveException;
   import bank.OverdrawException;
   import bank.sockets.Driver.NetworkHandler;
16
   public class Driver implements bank.BankDriver {
     private Socket s;
     private PrintWriter out;
     private BufferedReader in;
21   private Bank bank;

     @Override
     public void connect(String[] args) throws IOException {
       s = new Socket(args[0], Integer.parseInt(args[1]), null, 0);
26     out = new PrintWriter(new OutputStreamWriter(s.getOutputStream(),Charset.forName("UTF
           -16")));
       in = new BufferedReader(new InputStreamReader(s.getInputStream(),Charset.forName("UTF
           -16")));
       NetworkHandler handler = new NetworkHandler(out, in);
       bank = new Bank(handler);
     }
31
     @Override
     public void disconnect() throws IOException {
       bank = null;
       out.close();
36     out = null;
       in.close();
       in = null;
       s.close();
       s = null;
41   }
```

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

    static class NetworkHandler {
      private PrintWriter out;
      private BufferedReader in;

      public NetworkHandler(PrintWriter out, BufferedReader in) {
        this.out = out;
        this.in = in;
      }

      public String[] sendMessage(String command, String... args) throws IOException {
        StringBuilder sb = new StringBuilder(escape(command));
        for (int i = 0; i < args.length; i++) {
          sb.append(":");
          sb.append(escape(args[i]));
        }
        out.println(sb.toString());
        out.flush();

        String[] message = in.readLine().split(":");
        for (int i = 0; i < message.length; i++) {
          message[i] = unescape(message[i]);
        }

        if (message.length == 0 || message[0] == null) {
          throw new IOException("Illegal message recieved!");
        }

        return message;
      }

      private static String escape(String s) {
        return s.replace("\n","").replace(":", "[colon]");
      }

      private static String unescape(String s) {
        return s.replace("\n","").replace("[colon]", ":");
      }
    }

    static class Bank implements bank.Bank {
      private NetworkHandler handler;

      public Bank(NetworkHandler handler) {
        this.handler = handler;
      }

      @Override
      public Set<String> getAccountNumbers() throws IOException {
        String[] message = handler.sendMessage("get-acc-numbers");

        Set<String> accountNumbers = new HashSet<String>();
        int count = Integer.parseInt(message[0]);
        if (message.length < count + 1) {
          throw new IOException("Invalid message!");
        }

        for (int i = 1; i <= count; i++) {
          accountNumbers.add(message[i]);
        }
        return accountNumbers;
      }

      @Override
      public String createAccount(String owner) throws IOException {
        String[] message = handler.sendMessage("create", owner);
        if (message[0].equals("ok") && message.length > 1) {
```

```
            return message[1];
          } else {
            return null;
          }
        }

        @Override
        public boolean closeAccount(String number) throws IOException {
          String[] message = handler.sendMessage("close", number);
          if (message[0].equals("ok")) {
            return true;
          } else {
            return false;
          }
        }


        @Override
        public bank.Account getAccount(String number) throws IOException {
          String[] message = handler.sendMessage("get-acc", number);
          if (message[0].equals("ok") && message.length > 1) {
            return new Account(message[1], handler);
          } else {
            return null;
          }

        }

        @Override
        public void transfer(bank.Account from, bank.Account to, double amount)
            throws IOException, InactiveException, OverdrawException {
          String[] message = handler.sendMessage("transfer", from.getNumber(), to.getNumber(),
              String.valueOf(amount));

          String status = message[0];
          if (!status.equals("ok")) {
            switch (status) {
              case "InactiveException":
                throw new InactiveException();
              case "OverdrawException":
                throw new OverdrawException();
              case "ArgumentException":
                throw new IllegalArgumentException();
              default:
                throw new IOException("Illegal status recieved!");
            }
          }
        }

      }

      static class Account implements bank.Account {
        private String number;
        private NetworkHandler handler;

        Account(String number, NetworkHandler handler) {
          this.number = number;
          this.handler = handler;
        }

        @Override
        public double getBalance() throws IOException {
          String[] message = handler.sendMessage("get-balance", number);
          return Double.parseDouble(message[0]);
        }

        @Override
        public String getOwner() throws IOException {
          String[] message = handler.sendMessage("get-owner", number);
          return message[0];
        }
```

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

    @Override
    public boolean isActive() throws IOException {
      String[] message = handler.sendMessage("get-active", number);
      return message[0].equals("active");
    }

    @Override
    public void deposit(double amount) throws InactiveException, IllegalArgumentException,
        IOException {
      String[] message = handler.sendMessage("deposit", number, String.valueOf(amount));
      String status = message[0];
      if (!status.equals("ok")) {
        switch (status) {
          case "InactiveException":
            throw new InactiveException();
          case "ArgumentException":
            throw new IllegalArgumentException();
          default:
            throw new IOException("Illegal status recieved!");
        }
      }
    }

    @Override
    public void withdraw(double amount) throws InactiveException, OverdrawException,
        IllegalArgumentException, IOException {
      String[] message = handler.sendMessage("withdraw", number, String.valueOf(amount));
      String status = message[0];
      if (!status.equals("ok")) {
        switch (status) {
          case "InactiveException":
            throw new InactiveException();
          case "ArgumentException":
            throw new IllegalArgumentException();
          case "OverdrawException":
            throw new OverdrawException();
          default:
            throw new IOException("Illegal status recieved!");
        }
      }
    }

  }
}
```

2.2 Server

Listing 2: BankSever Klasse

```
using System;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;

namespace BankServer
{
    class BankServer
    {
        private readonly TcpClient _client;
        private readonly Bank _bank;
        private readonly StreamReader _reader;
```

```
        private readonly StreamWriter _writer;

        public BankServer(TcpClient client, Bank bank)
        {
            _client = client;
            _bank = bank;
            _reader = new StreamReader(_client.GetStream(), Encoding.Unicode);
            _writer = new StreamWriter(_client.GetStream(), Encoding.Unicode);
        }

        private static String Escape(String s)
        {
            return s.Replace("\n", "").Replace(":", "[colon]");
        }

        private static String Unescape(String s)
        {
            return s.Replace("\n", "").Replace("[colon]", ":");
        }

        public void SendMessage(String command, params string[] args) {
      var sb = new StringBuilder(Escape(command));
      for (var i = 0; i < args.Count(); i++) {
        sb.Append(":");
        sb.Append(Escape(args[i]));
      }
      _writer.WriteLine(sb.ToString());
      _writer.Flush();
    }

        public void Read()
        {

            String input;
            while ((input = _reader.ReadLine()) != null)
            {
                Account account;
                var message = input.Split(':').Select(Unescape).ToArray();

                switch (message[0])
                {
                    case "get-acc-numbers":
                        var accNumbers = _bank.GetAccountNumbers().ToArray();
                        SendMessage(accNumbers.Length.ToString(CultureInfo.InvariantCulture
                            ), accNumbers);
                        break;
                    case "create":
                        if (message.Length < 2)
                        {
                            SendMessage("error");
                        }
                        else
                        {
                            var number = _bank.CreateAccount(message[1]);
                            if (!String.IsNullOrEmpty(number))
                            {
                                SendMessage("ok", number);
                            }
                            else
                            {
                                SendMessage("error");
                            }
                        }
                        break;
                    case "close":
                        if (message.Length < 2)
                        {
                            SendMessage("error");
                        }
                        else
                        {
                            SendMessage(_bank.CloseAccount(message[1]) ? "ok" : "error");
```

```
                }
                break;
            case "get-acc":
                if (message.Length < 2)
                {
                    SendMessage("error");
                }
                else
                {
                    account = _bank.GetAccount(message[1]);
                    if (account != null)
                    {
                        SendMessage("ok", account.Number);
                    }
                    else
                    {
                        SendMessage("error");
                    }
                }
                break;
            case "transfer":
                if (message.Length < 4)
                {
                    SendMessage("error");
                }
                else
                {
                    var fromAccount = _bank.GetAccount(message[1]);
                    var toAccount = _bank.GetAccount(message[2]);

                    if (fromAccount == null && toAccount == null)
                    {
                        SendMessage("ArgumentException");
                    }
                    else
                    {
                        try
                        {
                            _bank.Transfer(fromAccount, toAccount, Double.Parse(
                                message[3]));
                            SendMessage("ok");
                        }
                        catch (Exception e)
                        {
                            SendMessage(e.GetType().Name);
                        }
                    }
                }
                break;
            case "get-balance":
                if (message.Length < 2)
                {
                    SendMessage(Double.NaN.ToString(CultureInfo.InvariantCulture));
                }
                else
                {
                    account = _bank.GetAccount(message[1]);
                    SendMessage((account != null ? account.Balance : Double.NaN)
                        .ToString(CultureInfo.InvariantCulture));
                }
                break;
            case "get-owner":
                if (message.Length < 2)
                {
                    SendMessage("");
                }
                else
                {
                    account = _bank.GetAccount(message[1]);
                    SendMessage(account != null ? account.Owner : "");
                }
                break;
```

```
159                     case "get-active":
                            if (message.Length < 2)
                            {
                                SendMessage(false.ToString());
                            }
164                         else
                            {
                                account = _bank.GetAccount(message[1]);
                                SendMessage((account != null && account.IsActive).ToString());
                            }
169                         break;
                        case "deposit":
                            if (message.Length < 3)
                            {
                                SendMessage("error");
174                         }
                            else
                            {
                                account = _bank.GetAccount(message[1]);
                                if (account != null)
179                             {
                                    try
                                    {
                                        account.Deposit(Double.Parse(message[2]));
                                        SendMessage("ok");
184                                 }
                                    catch (Exception e)
                                    {
                                        SendMessage(e.GetType().Name);
                                    }
189                             }
                                else
                                {
                                    SendMessage("ArgumentException");
                                }
194                         }
                            break;
                        case "withdraw":
                            if (message.Length < 3)
                            {
199                             SendMessage("error");
                            }
                            else
                            {
                                account = _bank.GetAccount(message[1]);
204                             if (account != null)
                                {
                                    try
                                    {
                                        account.Withdraw(Double.Parse(message[2]));
209                                     SendMessage("ok");
                                    }
                                    catch (Exception e)
                                    {
                                        SendMessage(e.GetType().Name);
214                                 }
                                }
                                else
                                {
                                    SendMessage("ArgumentException");
219                             }
                            }
                            break;
                        default:
                            SendMessage("error");
224                         break;
                    }
                }
            }

229 // ReSharper disable FunctionNeverReturns
            static void Main()
```

```
        {
            var bank = new Bank();
            var listener = new TcpListener(IPAddress.Any, 5678);
            listener.Start();

            while (true)
            {
                var client = listener.AcceptTcpClient();
                var thread = new Thread(new BankServer(client, bank).Read);
                thread.Start();
            }
        }
// ReSharper restore FunctionNeverReturns
    }
}
```

Listing 3: Bank Klasse

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace BankServer
{
    class Bank
    {
        private readonly Dictionary<string, Account> _accounts = new Dictionary<string,
            Account>();

    public IEnumerable<string> GetAccountNumbers() {
      return _accounts.Values.Where(account => account.IsActive).Select(element => element.
          Number);
    }

    public string CreateAccount(string owner) {
      var newAccount = new Account(owner);
      _accounts.Add(newAccount.Number, newAccount);
      return newAccount.Number;
    }

    public bool CloseAccount(string number)
    {
        Account closeAccount;
        if (!_accounts.TryGetValue(number, out closeAccount) || !closeAccount.Balance.
            Equals(0) || !closeAccount.IsActive)
        {
            return false;
        }
        closeAccount.IsActive = false;
        return true;
    }


        public Account GetAccount(String number)
    {
        Account account;
        _accounts.TryGetValue(number, out account);
        return account;
    }


    public void Transfer(Account from, Account to, double amount) {

      if (!from.IsActive || !to.IsActive) {
        throw new InactiveException();
      }

      from.Withdraw(amount);
      to.Deposit(amount);
    }
```

```
50

    }
}
```

Listing 4: Account Klasse

```
using System;
using System.Globalization;

namespace BankServer
{
    public class Account
    {

        private static int _sequence;

        private readonly string _owner;

        public string Owner
        {
            get { return _owner; }
        }

        private readonly string _number;

        public string Number
        {
            get { return _number;  }
        }

        public double Balance { get; private set; }

        public bool IsActive { get; set; }

    public Account(String owner) {
        Balance = 0;
        _owner = owner;
        _number = _sequence++.ToString(CultureInfo.InvariantCulture);
        IsActive = true;
    }


    public void Deposit(double amount) {
            if (!IsActive)
            {
        throw new InactiveException();
      }

      if (amount < 0) {
        throw new ArgumentException();
      }

      Balance += amount;
    }

    public void Withdraw(double amount) {
      if (!IsActive) {
        throw new InactiveException();
      }

      if (amount < 0) {
        throw new ArgumentException("");
      }

      if (amount > Balance) {
        throw new OverdrawException();
      }

      Balance -= amount;
```

```
        }

      }
67 }
```