Übung 2: Servlet Implementierung

# Grundidee

Die zweite Übung basiert mehr oder weniger auf den Prinzipien der ersten Übung:

1. Das Interface soll komplett objektbasiert gestaltet werden
2. Die Übertragung der Befehle soll mit einem Serializer und Deserializer über ein Netzwerk erfolgen. Es soll also kein zeichen- oder tokenbasiertes Protokoll erbastelt werden, da dieses schlecht wartbar, fehleranfällig und zu oft zu unsicher und nicht robust genug implementiert wird

Im Gegensatz zur ersten Übung soll in der zweiten Übung die Übertragung über ein Request an ein Servlet und der daraus resultierenden Response erfolgen. Vergleicht man die beiden Übungen, sollte ersichtlich werden, dass die Socket und Servlet Implementierungen bis auf die Übertragunslogik allen Code gemeinsam Teilen und deshalb die konkreten Klassen BankDriver, Bank und Account ausgelagert werden sollten und über eine situationspassende Sender Klasse ihre Übertragung vornehmen sollen.

Diese Idee wurde in der zweiten Übung umgesetzt und ich möchte auf ein paar Feedbackpunkte der ersten Übung eingehen:

1. Aller UI relevanter Code wurde in die Clientappikation verschoben
2. Eine Library mit dem Namen «shared» enthält jetzt alle möglichen Implementierung. Eine einzige Library wurde deshalb verwendet, um die Hin- und Herserialisierung betreffend Classloader und Abhängigkeiten in Maven zu vereinfachen
3. Die closeAccount Methode wurde entfern und die Funktionalität in die Bank Klasse verschoben, welche auch thematisch für den Account verantwortlich ist

Des Weiteren gab es diverse kleinere Verbesserungen bezüglich Accesslevel und anderen nicht erwähnenswerten Gründen.

# Implementierung

Die konkrete Implementierung der Bank Klasse wurde vereinheitlicht und mittels einer änderbaren Sender Klasse die in den Konstruktor injiziert wird umgesetzt. Zum Konvertieren zwischen lokalen und entfernten Profile welche durch das Hin- und Herkonvertieren entstehen, wurde eine Klasse Converter geschrieben:

|  |
| --- |
| package ch.fhnw.vesys.shared.core;  import ch.fhnw.vesys.shared.api.\*;  import ch.fhnw.vesys.shared.local.LocalDriver;  import java.io.IOException;  import java.io.Serializable;  import java.util.Set;  public abstract class Driver implements BankDriver {  private final InternalBank bank;  public Driver(Sender sender) {  this.bank = new InternalBank(sender);  }  abstract public void connect(String[] args);  abstract public void disconnect();  @Override  public Bank getBank() {  return bank;  }  private static class InternalBank implements Bank {  private final Converter converter;  private final Sender sender;  InternalBank(Sender sender) {  this.converter = new InternalConverter();  this.sender = sender;  }  @Override  public String createAccount(String owner) throws IOException {  return TaskRunner.launchCreateAccountTask(sender, owner);  }  @Override  public boolean closeAccount(String number) throws IOException {  return TaskRunner.launchCloseAccountTask(sender, number);  }  @Override  public Set<String> getAccountNumbers() throws IOException {  return TaskRunner.launchGetAccountNumbers(sender);  }  @Override  public Account getAccount(String number) throws IOException {  return TaskRunner.launchGetAccountTask(converter, sender, number);  }  @Override  public void transfer(Account from, Account to, double amount) throws IOException, IllegalArgumentException, OverdrawException, InactiveException {  TaskRunner.launchTransferTask(converter, sender, from, to, amount);  }  }  private static class InternalAccount implements Account, Serializable {  private final Sender sender;  private final String number;  private final String owner;  InternalAccount(Sender sender, String number, String owner) {  this.sender = sender;  this.number = number;  this.owner = owner;  }  @Override  public String getNumber() throws IOException {  return number;  }  @Override  public String getOwner() throws IOException {  return owner;  }  @Override  public boolean isActive() throws IOException {  return TaskRunner.launchIsActiveTask(sender, number);  }  @Override  public void deposit(double amount) throws IOException, IllegalArgumentException, InactiveException {  TaskRunner.launchDepositTask(sender, number, amount);  }  @Override  public void withdraw(double amount) throws IOException, IllegalArgumentException, OverdrawException, InactiveException {  TaskRunner.launchWithdrawTask(sender, number, amount);  }  @Override  public double getBalance() throws IOException {  return TaskRunner.launchGetBalanceTask(sender, number);  }  }  private static class InternalConverter implements Converter {  @Override  public Account fromLocalToRemoteAccount(Sender sender, Account localaccount) throws IOException {  if (localaccount == null) {  return null;  }  return new InternalAccount(sender, localaccount.getNumber(), localaccount.getOwner());  }  @Override  public Account fromRemoteToLocalAccount(Account remoteaccount) throws IOException, InactiveException {  if (remoteaccount == null) {  return null;  }  LocalDriver.LocalAccount localaccount = new LocalDriver.LocalAccount(remoteaccount.getOwner(), remoteaccount.getNumber());  localaccount.active = remoteaccount.isActive();  if (localaccount.active) {  localaccount.deposit(remoteaccount.getBalance());  }  return localaccount;  }  }  } |

Der Sender ist als Interface realisiert und wird durch den entsprechenden Server implementiert:

|  |
| --- |
| package ch.fhnw.vesys.shared.core;  public interface Sender {  Task sendTask(Task task);  } |

Der Konverter ist auch als Interface realisiert. Der Grund ist der, dass die interne Implementierung nicht in die Tasks gestreut werden soll (Die interne Implementierung kann als private deklariert werden):

|  |
| --- |
| package ch.fhnw.vesys.shared.core;  import ch.fhnw.vesys.shared.api.Account;  import ch.fhnw.vesys.shared.api.InactiveException;  import java.io.IOException;  interface Converter {  Account fromLocalToRemoteAccount(Sender sender, Account localaccount) throws IOException;  Account fromRemoteToLocalAccount(Account remoteaccount) throws IOException, InactiveException;  } |

Die Tasks, respektive deren ausführenden Einheiten sind analog zu der in Übung 1 implementiert und werden folglich hier nicht noch einmal aufgeführt (Siehe Source Code).

Die Implementierung für einen Servlet Driver sieht durch die Verallgemeinerung des Drivers wie folgt aus (Diese Implementierung wird durch den Bankenclient geladen) – einzig der Sender muss noch neu implementiert werden:

|  |
| --- |
| package ch.fhnw.vesys.shared.servlet;  import ch.fhnw.vesys.shared.core.Driver;  import ch.fhnw.vesys.shared.core.Sender;  import ch.fhnw.vesys.shared.core.Task;  import java.io.ObjectInputStream;  import java.io.ObjectOutputStream;  import java.net.HttpURLConnection;  import java.net.URL;  public class ServletDriver extends Driver {  private static final ServletSender sender = new ServletSender();  public ServletDriver() {  super(sender);  }  @Override  public void connect(String[] args) {  ServletSender.hostname = args[0];  ServletSender.port = Integer.parseInt(args[1]);  System.out.println("Connected to servlet driver.");  }  @Override  public void disconnect() {  System.out.println("Disconnected from servlet driver");  }  private static class ServletSender implements Sender {  static String hostname;  static int port;  @Override  public Task sendTask(Task task) {  try {  URL url = new URL("http://" + hostname + ":" + port);  HttpURLConnection connection = (HttpURLConnection) url.openConnection();  connection.setDoOutput(true);  connection.setRequestMethod("POST");  ObjectOutputStream outputstream = new ObjectOutputStream(connection.getOutputStream());  outputstream.writeObject(task);  ObjectInputStream inputstream = new ObjectInputStream(connection.getInputStream());  task = (Task) inputstream.readObject();  return task;  } catch (Exception exception) {  return new Task.InvalidTask();  }  }  }  } |

Auf der Gegenseite wird für den Servlet Server ein integrierter Tomcat Server verwendet, da ich hier keinen eigenständigen Application Server verwenden möchte:

|  |
| --- |
| package ch.fhnw.vesys.serverservlet;  import ch.fhnw.vesys.shared.api.BankDriver;  import ch.fhnw.vesys.shared.local.LocalDriver;  import org.apache.catalina.Context;  import org.apache.catalina.startup.Tomcat;  import java.io.File;  class ServletServer {  private ServletServer(int port) throws Exception {  BankDriver bankdriver = new LocalDriver();  Tomcat tomcat = new Tomcat();  tomcat.setPort(port);  Context context = tomcat.addContext("", new File(".").getAbsolutePath());  Tomcat.addServlet(context, "ServletServer", new RequestHandler(bankdriver));  context.addServletMappingDecoded("/\*", "ServletServer");  tomcat.start();  tomcat.getServer().await();  }  public static void main(String[] args) throws Exception {  new ServletServer(1234);  }  } |

Das durch den Server geladene Servlet sieht wie folgt aus:

|  |
| --- |
| package ch.fhnw.vesys.serverservlet;  import ch.fhnw.vesys.shared.api.BankDriver;  import ch.fhnw.vesys.shared.core.Task;  import javax.servlet.ServletException;  import javax.servlet.ServletOutputStream;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import java.io.IOException;  import java.io.ObjectInputStream;  import java.io.ObjectOutputStream;  class RequestHandler extends HttpServlet {  private final BankDriver bankdriver;  RequestHandler(BankDriver bankdriver) {  this.bankdriver = bankdriver;  }  @Override  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  ServletOutputStream outputstream = response.getOutputStream();  outputstream.write("Please use the servlet sender implementation and POST HTTP method to access this servlet!".getBytes());  outputstream.flush();  outputstream.close();  }  @Override  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  try (ObjectInputStream inputstream = new ObjectInputStream(request.getInputStream()); ObjectOutputStream outputstream = new ObjectOutputStream(response.getOutputStream())) {  Task task = (Task) inputstream.readObject();  task.executeHandledTask(bankdriver);  outputstream.writeObject(task);  response.getOutputStream().close();  } catch (Exception exception) {  exception.printStackTrace();  }  }  } |

# Verbesserungsmöglichkeiten

Mit der aktuellen Implementierung bin ich wie schon in der ersten Version sehr zufrieden. Ich habe diese Woche durch den Input von Sandro Schwager ein Lambda und BiFunction Implementierung getestet. Diese ist kürzer, durch die Serialisierung aber circa 10 Mal langsamer. Man müsste hier also auf einen alternativen Serializer wie Kryo ausweichen.

# Source Code

Der Source Code ist hier zu finden: <https://github.com/swaechter/fhnw/tree/master/Module/vesys/bank>