Uebung04 vesys

Daniel Gürber

4. Semester (FS 2013)

Inhaltsverzeichnis

1 Beschreibung

1.1 Architektur

Die Bank wird als REST Service implementiert, wobei GET, POST, PUT, HEAD und DELETE Anfragen für die accounts Ressource angeboten werden.
Transfers werden in einer separaten Ressource verwaltet, damit sie am Stück ausgeführt werden.

2 Code

2.1 Transfer Types

Listing 1: AccountURLs

```
1  package bank.types;

   import java.util.ArrayList;
   import java.util.List;

6  import javax.xml.bind.annotation.XmlRootElement;

   @XmlRootElement
   public class AccountURLs {
     private List<String> url;
11
     public List<String> getUrl() {
       return url;
     }

16   public void setUrl(List<String> url) {
       this.url = url;
     }

     public AccountURLs() {
21     url = new ArrayList<String>();
     }
   }
```

Listing 2: AccountData

```
   package bank.types;
2
   import javax.xml.bind.annotation.XmlRootElement;

   @XmlRootElement(name = "account")
   public class AccountData {
7    private String number;
     private String owner;
     private double balance;
     private boolean active;
     private int modified;
12
     public String getNumber() {
       return number;
     }
     public void setNumber(String number) {
17     this.number = number;
     }
     public String getOwner() {
       return owner;
     }
22   public void setOwner(String owner) {
       this.owner = owner;
     }
```

```
    public double getBalance() {
      return balance;
27  }
    public void setBalance(double balance) {
      this.balance = balance;
    }
    public boolean isActive() {
32    return active;
    }
    public void setActive(boolean active) {
      this.active = active;
    }
37  public int getModified() {
      return modified;
    }
    public void setModified(int modified) {
      this.modified = modified;
42  }


  }
```

Listing 3: TransactionData

```
  package bank.types;

  import javax.xml.bind.annotation.XmlRootElement;

5 @XmlRootElement(name = "TransactionData")
  public class TransactionData {
    private double fromAmount;
    private double toAmount;
    private String fromNumber, toNumber, fromETag, toEtag;
10  public double getFromAmount() {
      return fromAmount;
    }
    public void setFromAmount(double fromAmount) {
      this.fromAmount = fromAmount;
15  }
    public double getToAmount() {
      return toAmount;
    }
    public void setToAmount(double toAmount) {
20    this.toAmount = toAmount;
    }
    public String getFromNumber() {
      return fromNumber;
    }
25  public void setFromNumber(String fromNumber) {
      this.fromNumber = fromNumber;
    }
    public String getToNumber() {
      return toNumber;
30  }
    public void setToNumber(String toNumber) {
      this.toNumber = toNumber;
    }
    public String getFromETag() {
35    return fromETag;
    }
    public void setFromETag(String fromETag) {
      this.fromETag = fromETag;
    }
40  public String getToEtag() {
      return toEtag;
    }
    public void setToEtag(String toEtag) {
      this.toEtag = toEtag;
45  }
  }
```

2.2 Server

2.2.1 Ressourcen

Listing 4: Accounts

```
package bank.resources;

import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.HEAD;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.EntityTag;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Request;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import javax.ws.rs.core.Response.Status;
import javax.ws.rs.core.UriInfo;

import bank.implementation.Account;
import bank.implementation.Bank;
import bank.types.AccountURLs;

import com.sun.jersey.spi.resource.Singleton;

@Path("/accounts")
@Singleton
public class Accounts {


  @GET
  @Produces(MediaType.APPLICATION_XML)
  public Response getAccountURLs(@Context Request request, @Context UriInfo uri) {
    EntityTag modified = new EntityTag(String.valueOf(Bank.modified));

    ResponseBuilder builder = request.evaluatePreconditions(modified);
    if (builder!= null) {
      return builder.build();
    }

    AccountURLs urls = new AccountURLs();
    for (Account account : Bank.accounts.values()) {
      if (account.isActive()) {
        urls.getUrl().add(uri.getRequestUri().toString() + "/" + account.getNumber());
      }
    }

    builder = Response.ok(urls);
    builder.tag(modified);
    return builder.build();
  }

  @POST
  @Consumes("text/plain")
  public Response createAccount( @Context UriInfo uri, String owner) {
    Account account = new Account(owner);
    Bank.modified++;
    Bank.accounts.put(account.getNumber(), account);
    ResponseBuilder builder;
    builder = Response.created(uri.getRequestUri().resolve(account.getNumber()));
    return builder.build();
  }

  @GET
```

```
  @Path("{id}")
  public Response getAccount(@PathParam("id") String number) {
    Account account = Bank.accounts.get(number);

    if (account==null) {
      return Response.status(Status.NOT_FOUND).build();
    }

    else return Response.ok(account.getAccountData()).build();
  }

  @DELETE
  @Path("{id}")
  public Response deleteAccount(@Context Request request, @PathParam("id") String number) {
    Account account = Bank.accounts.get(number);

    if (account==null) {
      return Response.status(Status.NOT_FOUND).build();
    }
    synchronized(account) {
      if (account.getBalance() <= 0.0) {
        account.setActive(false);
        Bank.modified++;
      } else {
        return Response.status(Status.CONFLICT).entity("Balance is not 0").build();
      }
    }

    return Response.status(Status.NO_CONTENT).build();
  }

  @HEAD
  @Path("{id}")
  public Response isActive(@PathParam("id") String number) {
    Account account = Bank.accounts.get(number);

    if (account==null) {
      return Response.status(Status.NOT_FOUND).build();
    }

    if (account.isActive()) {
      return Response.ok().build();
    } else {
      return Response.status(Status.GONE).build();
    }
  }

  @PUT
  @Path("{id}")
  @Consumes("text/plain")
  public Response setBalance(@Context Request request, @PathParam("id") String number,
      String sAmount) {

    double amount;

    amount = Double.parseDouble(sAmount);

    Account account = Bank.accounts.get(number);

    if (account==null) {
      return Response.status(Status.NOT_FOUND).build();
    }

    synchronized (account) {
      ResponseBuilder builder = request.evaluatePreconditions(new EntityTag(String.valueOf(
          account.getModified())));
      if (builder != null) {
        builder.build();
      }

      account.setBalance(amount);
    }
```

```
      return Response.status(Status.NO_CONTENT).build();
    }
}
```

Listing 5: Transfer

```
package bank.resources;

import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;
import javax.ws.rs.core.UriInfo;

import bank.implementation.Account;
import bank.implementation.Bank;
import bank.types.TransactionData;

import com.sun.jersey.spi.resource.Singleton;

@Path("/transfer")
@Singleton
public class Transfer {

  @POST
  public Response doTransfer(@Context UriInfo uri, TransactionData data) {
    Account fromAccount = Bank.accounts.get(data.getFromNumber());
    Account toAccount = Bank.accounts.get(data.getToNumber());

    if (fromAccount==null || toAccount==null) {
      return Response.status(Status.NOT_FOUND).build();
    } else {
      Account a;
      Account b;
      if (fromAccount.getNumber().compareTo(toAccount.getNumber())> 0) {
        a = fromAccount;
        b = toAccount;
      } else {
        b = fromAccount;
        a = toAccount;
      }

      synchronized (a) {
        synchronized (b) {
          if (!String.valueOf(fromAccount.getModified()).equals(data.getFromETag()) ||
              !String.valueOf(toAccount.getModified()).equals(data.getToEtag())) {
            return Response.status(Status.CONFLICT).build();
          }

          fromAccount.setBalance(data.getFromAmount());
          toAccount.setBalance(data.getToAmount());

          return Response.noContent().build();
        }
      }
    }

  }
}
```

2.2.2 Implementierung

Listing 6: Bank

```
package bank.implementation;
```

```
import java.util.concurrent.ConcurrentHashMap;

public class Bank {
  public static ConcurrentHashMap<String, Account> accounts = new ConcurrentHashMap<>();
  public static volatile int modified = 0;
}
```

Listing 7: Account

```
package bank.implementation;

import java.util.UUID;

import bank.types.AccountData;

public final class Account {
  private final String number;
  private final String owner;
  private volatile double balance;
  private volatile boolean active = true;
  private volatile int modified;

  public Account(final String owner) {
    this.owner = owner;
    this.number = UUID.randomUUID().toString();
    this.modified = 0;
  }

  public double getBalance() {
    return balance;
  }

  public String getOwner() {
    return owner;
  }

  public String getNumber() {
    return number;
  }

  public int getModified() {
    return modified;
  }

  public boolean isActive() {
    return active;
  }

  public synchronized void setActive(final boolean active) {
    this.modified++;
    this.active = active;
  }

  public synchronized void setBalance(final double amount)  {

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

    this.modified++;
    this.balance = amount;
  }

  public synchronized AccountData getAccountData() {
    AccountData data = new AccountData();
    data.setNumber(number);
    data.setOwner(owner);
    data.setBalance(balance);
    data.setActive(active);
    data.setModified(modified);
```

```
62      return data;
    }
  }
```

2.3 Client

Listing 8: Driver

```
  package bank.rest;

  import java.io.IOException;
4 import java.util.HashSet;
  import java.util.Set;

  import javax.ws.rs.core.EntityTag;
  import javax.ws.rs.core.MediaType;
9
  import bank.InactiveException;
  import bank.OverdrawException;
  import bank.types.AccountData;
  import bank.types.AccountURLs;
14 import bank.types.TransactionData;

  import com.sun.jersey.api.client.Client;
  import com.sun.jersey.api.client.ClientResponse;
  import com.sun.jersey.api.client.WebResource;
19
  public class Driver implements bank.BankDriver {
    private String url;
    private Bank bank;

24  @Override
    public void connect(String[] args) throws IOException {
      url = args[0];
      bank = new Bank(url);
    }
29
    @Override
    public void disconnect() throws IOException {
      bank = null;
    }
34
    @Override
    public Bank getBank() {
      return bank;
    }
39
    static class Bank implements bank.Bank {
      private String url;
      Client c;
      WebResource r;
44    WebResource rTransfer;
      private Set<String> accountNumbers;
      private EntityTag modified;

      public Bank(String url) {
49      this.url = url;
        c = Client.create();
        r = c.resource(url + "accounts/");
        rTransfer = c.resource(url + "transfer/");
      }
54
      @Override
      public Set<String> getAccountNumbers() throws IOException {

        ClientResponse response;
59      if (accountNumbers==null) {
          response = r.get(ClientResponse.class);
        } else {
          response = r.header("If-None-Match", modified).get(ClientResponse.class);
```

```
    }

    switch(response.getClientResponseStatus()) {
      case OK:
        accountNumbers = new HashSet<String>();
        for (String url : response.getEntity(AccountURLs.class).getUrl()) {
          accountNumbers.add(url.substring(url.lastIndexOf('/')+1));
        }
        modified = response.getEntityTag();
        return accountNumbers;
      case NOT_MODIFIED:
        return accountNumbers;
      default:
        throw new IOException("Error connecting to the server");
    }

  }

  @Override
  public String createAccount(String owner) throws IOException {
    ClientResponse response = r.type(MediaType.TEXT_PLAIN).post(ClientResponse.class,
        owner);
    if (response.getClientResponseStatus() == ClientResponse.Status.CREATED) {
      String accUrl = response.getLocation().toString();
      return accUrl.substring(accUrl.lastIndexOf('/')+1);
    } else {
      throw new IOException(response.getClientResponseStatus().getReasonPhrase());
    }
  }

  @Override
  public boolean closeAccount(String number) throws IOException {
    WebResource r = c.resource(url + "accounts/" + number);
    ClientResponse response = r.delete(ClientResponse.class);
    switch (response.getClientResponseStatus()) {
      case OK:
      case NO_CONTENT:
        return true;
      case CONFLICT:
        return false;
      default:
        throw new IOException(response.getClientResponseStatus().getReasonPhrase());
    }
  }


  @Override
  public bank.Account getAccount(String number) throws IOException {
    if (number.length() > 0) {
      WebResource r = c.resource(url + "accounts/" + number);
      ClientResponse response = r.get(ClientResponse.class);
      switch (response.getClientResponseStatus()) {
        case OK:
          return new Account(response.getEntity(AccountData.class), r);
        case NOT_FOUND:
          return null;
        default:
          throw new IOException("Error connecting to the server");
      }
    } else {
      return null;
    }
  }

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

    boolean pending = true;
```

```
134
        if (amount < 0) {
          throw new IllegalArgumentException();
        }

139     while (pending) {
          from.update();
          to.update();

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

          if (amount > from.getBalance()) {
            throw new OverdrawException();
149       }

          TransactionData data = new TransactionData();
          data.setFromETag(from.modified);
          data.setFromAmount(from.balance - amount);
154       data.setFromNumber(from.number);
          data.setToEtag(to.modified);
          data.setToAmount(to.balance + amount);
          data.setToNumber(to.number);

159       ClientResponse response = rTransfer.post(ClientResponse.class, data);

          switch(response.getClientResponseStatus()) {
            case NO_CONTENT:
              pending=false;
164           break;
            case CONFLICT:
              pending=true;
              break;
            default:
169           throw new IOException(response.getClientResponseStatus().getReasonPhrase());
          }
        }
      }
174
    static class Account implements bank.Account {
      private final String number;
      private final String owner;
      private volatile double balance;
179   private volatile boolean active = true;
      private volatile String modified;
      private WebResource r;

      public Account(AccountData entity, WebResource r) {
184     this.number = entity.getNumber();
        this.owner = entity.getOwner();
        this.balance = entity.getBalance();
        this.active = entity.isActive();
        this.modified = String.valueOf(entity.getModified());
189     this.r = r;
      }

      public synchronized void update() throws IOException {
        ClientResponse response = r.get(ClientResponse.class);
194     switch (response.getClientResponseStatus()) {
          case OK:
            AccountData acc = response.getEntity(AccountData.class);
            this.active = acc.isActive();
            this.balance = acc.getBalance();
199         this.modified = String.valueOf(acc.getModified());
            break;
          default:
            throw new IOException(response.getClientResponseStatus().getReasonPhrase());
        }
204   }
```

```
    public String getModified() {
      return modified;
    }

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

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

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

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

    @Override
    public synchronized void deposit(double amount) throws InactiveException,
        IllegalArgumentException, IOException {
      boolean commited = false;
      while (!commited) {
        update();

        if (!this.active) {
          throw new InactiveException("Account is closed!");
        }

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

        ClientResponse response = r.header("If-Match", new EntityTag(modified)).put(
            ClientResponse.class,String.valueOf(balance+amount));

        switch(response.getClientResponseStatus()) {
          case NO_CONTENT:
            commited=true;
            break;
          case PRECONDITION_FAILED:
            commited=false;
            break;
          default:
            throw new IOException(response.getClientResponseStatus().getReasonPhrase());
        }
      }
    }

    @Override
    public void withdraw(double amount) throws InactiveException, OverdrawException,
        IllegalArgumentException, IOException {
      boolean commited = false;
      while (!commited) {
        update();

        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!");
            }

279         ClientResponse response = r.header("If-Match", new EntityTag(modified)).put(
                ClientResponse.class,String.valueOf(balance-amount));

            switch(response.getClientResponseStatus()) {
              case NO_CONTENT:
                commited=true;
284             break;
              case PRECONDITION_FAILED:
                commited=false;
                break;
              default:
289             throw new IOException(response.getClientResponseStatus().getReasonPhrase());
            }
          }
        }

294   }

   }
```