Verteilte Systeme FS 13
Übung 1

Thomas Baumann

4. März 2013

Inhaltsverzeichnis

1 Beschreibung

Ich habe die vorgegeben Klassen teilweise umbennant, damit es keine Probleme mit dem Verständnis der Klassennamen gibt. So beginnen alle Interface mit einem I. Alle Dateien sind auf der nächsten Seite aufgelistet.

Ich habe sowohl den Client, wie auch den Server in Java programmiert. Deshalb habe ich als Kommunikationsart den Object-Stream gewählt.

Für jede Anfrage wird ein einzelnes Objekt erstellt und gesendet. Diese Objekte sind vom Interface *IRequest* abgeleitet, damit kann auf dem Server immer die gleiche Methode verwendet werden, um die Anfrage zu bearbeiten. Damit die Verarbeitung durchgeführt werden kann, ruft der Server die Methode *handleRequest(IBank)* auf dem erhaltenen Objekt auf und übergibt dabei seine Bank. Als Rückgabewert erhält der Server ein Antwortobjekt (Abgeleitet vom Interface *IAnswer*), welches er zurücksenden kann. Damit die Exceptions auch über das Netzwerk gesendet werden können, existieren für diese eigene Klassen.

Der Client liest das erhaltene Objekt vom *ObjectInputStream* und ruft die *getData()* Methode auf, diese liefert im Normalfall ein *Object* zurück oder wenn ein Fehler auftritt, wird eine Exception geworfen. Dieses Objekt wird in das erwartete Resultat gecastet und an das GUI weitergeleitet. Sollte nicht das korrekte Objekt angekommen sein, so wird eine *IOException* geworfen.

2 Dateistruktur

- *src*
  - *bank*
    - *dummy*
      - Driver.java
    - *gui*
      - *tests*
        - BankTest.java
        - EfficiencyTest.java
        - FunctionalityTest.java
      - BankGUI.java
    - *local*
      - Driver.java
    - *socket*
      - *answer*
        - CloseAccountAnswer.java
        - CreateAccountAnswer.java
        - GetAccountAnswer.java
        - GetAccountNumbersAnswer.java
        - GetBalanceAnswer.java
        - GetOwnerAnswer.java
        - IAnswer.java
        - IllegalArgumentExceptionAnswer.java
        - InactiveExceptionAnswer.java
        - IOExceptionAnswer.java
        - IsActiveAnswer.java
        - OverdrawExceptionAnswer.java
        - VoidAnswer.java
      - *request*
        - CloseAccountRequest.java
        - CreateAccountRequest.java
        - DepositRequest.java
        - GetAccountNumbersRequest.java
        - GetAccountRequest.java
        - GetBalanceRequest.java
        - GetOwnerRequest.java
        - IRequest.java
        - IsActiveRequest.java
        - TransferRequest.java
        - WithdrawRequest.java
      - ClientDriver.java
      - ServerDriver.java
    - Bank.java
    - IAccount.java
    - IBank.java
    - IBankDriver.java
    - InactiveException.java
    - IServerDriver.java
    - OverdrawException.java
    - StartClient.java
    - StartServer.java

3 Code: Aufgabe A

Listing 1: Local Driver

```java
package bank.local;

import bank.IBank;
import bank.IBankDriver;

/**
 * This class provides an local implementation of the IBankDriver which means the bank is
 * coupled to the GUI
 *
 * @see IBankDriver
 * @author Thomas Baumann
 * @version 1.0
 */
public class Driver implements IBankDriver {
    private IBank bank;

    @Override
    public void connect(String[] args) {
        this.bank = new Bank();
        System.out.println("connected...");
    }

    @Override
    public void disconnect() {
        this.bank = null;
        System.out.println("disconnected...");
    }

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

}
```

Listing 2: Bank Implementation

```java
package bank.local;

import java.io.IOException;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import bank.IAccount;
import bank.IBank;
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 1.0
 */
public class Bank implements IBank {

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

    @Override
    public Set<String> getAccountNumbers() {
```

```
29          Set<String> set = new HashSet<>();
30          for (Account a : this.accounts.values()) {
31              if (a.isActive()) {
32                  set.add(a.getNumber());
33              }
34          }
35          return set;
36      }
37
38      @Override
39      public String createAccount(String owner) {
40          Account ac = new Account(owner);
41          this.accounts.put(ac.getNumber(), ac);
42          return ac.getNumber();
43      }
44
45      @Override
46      public boolean closeAccount(String number) {
47          Account a = this.accounts.get(number);
48          if (a != null && a.isActive() && a.getBalance() == 0.0) {
49              a.active = false;
50              return true;
51          }
52          return false;
53      }
54
55      @Override
56      public IAccount getAccount(String number) {
57          return this.accounts.get(number);
58      }
59
60      @Override
61      public void transfer(IAccount from, IAccount to, double amount) throws IOException,
62              InactiveException, OverdrawException {
63          from.withdraw(amount);
64          try {
65              to.deposit(amount);
66          } catch (Exception e) {
67              from.deposit(amount);
68              throw e;
69          }
70      }
71
72      static class Account implements IAccount {
73          private static int accountNumbers;
74          private String number;
75          private String owner;
76          private double balance;
77          private boolean active = true;
78
79          Account(String owner) {
80              this.owner = owner;
81              this.number = Integer.toString(++Account.accountNumbers);
82
83          }
84
85          @Override
86          public double getBalance() {
87              return this.balance;
88          }
89
90          @Override
91          public String getOwner() {
92              return this.owner;
93          }
94
95          @Override
96          public String getNumber() {
97              return this.number;
98          }
99
100         @Override
```

```
101         public boolean isActive() {
102             return this.active;
103         }
104 
105         @Override
106         public void deposit(double amount) throws InactiveException {
107             if (amount < 0) {
108                 throw new IllegalArgumentException("Amount can not be less then 0.");
109             }
110             if (!this.isActive()) {
111                 throw new InactiveException("Account is inactive.");
112             }
113             this.balance += amount;
114         }
115 
116         @Override
117         public void withdraw(double amount) throws IllegalArgumentException,
118                 InactiveException, OverdrawException {
119             if (amount < 0) {
120                 throw new IllegalArgumentException("Amount can not be less then 0.");
121             }
122             if (!this.isActive()) {
123                 throw new InactiveException("Account is inactive.");
124             }
125             if (amount > this.getBalance()) {
126                 throw new OverdrawException("The account has to less amount.");
127             }
128             this.balance -= amount;
129         }
130 
131     }
132 }
```

4 Code: Aufgabe B

4.1 Client

Listing 3: Client Socket Driver

```java
package bank.socket;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

import bank.InactiveException;
import bank.OverdrawException;
import bank.StartClient;
import bank.communication.AbstractClientDriver;
import bank.communication.answer.IAnswer;
import bank.communication.request.IRequest;

/**
 * This class provides an implementation of the AbstractClientDriver with sockets.
 *
 * @see AbstractClientDriver
 * @author Thomas Baumann
 * @version 1.1
 */
public final class ClientDriver extends AbstractClientDriver {

    private Socket socket;
    private ObjectOutputStream oout;
    private ObjectInputStream oin;

    @Override
    public void connect(String[] args) throws IOException {
        if (args.length < 2) {
            System.out.println("Usage: java " + StartClient.class.getName() + " "
                    + ClientDriver.class.getName() + " <port>");
            System.exit(1);
        }
        int port = 0;
        try {
            port = Integer.parseInt(args[1]);
        } catch (NumberFormatException e) {
            System.out.println("Port must be a number");
            System.exit(1);
        }
        this.socket = new Socket(args[0], port);
        this.socket.setTcpNoDelay(true);
        this.oout = new ObjectOutputStream(this.socket.getOutputStream());
        this.oin = new ObjectInputStream(this.socket.getInputStream());
        this.bank = new SocketBank();
        System.out.println("connected...");
    }

    @Override
    public void disconnect() throws IOException {
        try {
            // send null to close connection
            this.handleMessage(null);
        } catch (ClassNotFoundException | IllegalArgumentException | ClassCastException
                | OverdrawException | InactiveException e) {
            throw new IOException(e);
        }
        this.bank = null;
        this.socket.close();
        System.out.println("disconnected...");
    }

    @SuppressWarnings("unchecked")
    @Override
```

```
66      protected final <T> T handleMessage(IRequest r) throws ClassNotFoundException,
67              IOException, IllegalArgumentException, OverdrawException, InactiveException,
68              ClassCastException {
69          this.oout.writeObject(r);
70          this.oout.flush();
71          Object o = this.oin.readObject();
72          return ((IAnswer<T>) o).getData();
73      }
74
75  }
```

4.2 Server

Listing 4: Start des Servers

```
1   package bank;
2
3   import java.io.IOException;
4
5   /**
6    * Class StartServer is used to start the Server side of the bank application. As a
7    * runtime parameter the name of the class which implements the <code>IServerDriver</code>
8    * interface has to be specified. This class is then loaded and used as the server. This
9    * class needs a public constructor.
10   *
11   * <pre>
12   * Usage: java bank.StartServer &lt;classname&gt;
13   * </pre>
14   *
15   * E.g. start the application with one of the following commands. The additional runtime
16   * arguments are passed to the start method of the IServerDriver implementation.
17   *
18   * <pre>
19   * java bank.StartServer bank.socket.ServerDriver
20   * </pre>
21   *
22   * @author Thomas Baumann
23   * @version 1.0
24   */
25  public final class StartServer {
26
27      /** Utility class which is only used to start the application */
28      private StartServer() {}
29
30      public static void main(String args[]) {
31          if (args.length < 1) {
32              System.out.println("Usage: java " + StartServer.class.getName() + " <class>");
33              System.exit(1);
34          }
35
36          IServerDriver server = null;
37          try {
38              Class<?> c = Class.forName(args[0]);
39              server = (IServerDriver) c.newInstance();
40          } catch (ClassNotFoundException e) {
41              System.out.println("class " + args[0] + " coult not be found");
42              System.exit(1);
43          } catch (InstantiationException e) {
44              System.out.println("class " + args[0] + " could not be instantiated");
45              System.out.println("probably it has no public default constructor!");
46              System.exit(1);
47          } catch (IllegalAccessException e) {
48              System.out.println("class " + args[0] + " could not be instantiated");
49              System.out.println("probably it is not declared public!");
50              System.exit(1);
51          }
52
53          String[] serverArgs = new String[args.length - 1];
54          System.arraycopy(args, 1, serverArgs, 0, args.length - 1);
55
56          try {
```

```
57              server.start(serverArgs);
58          } catch (IOException e) {
59              System.out.println("Problem while starting the server:");
60              e.printStackTrace();
61              System.exit(1);
62          }
63      }
64
65  }
```

Listing 5: Interface für Server Driver

```
1  package bank;
2
3  import java.io.IOException;
4
5  /**
6   * The ServerDriver interface is used to access a particular bank server. The main program
7   * calls start to start the server.
8   *
9   * @author Thomas Baumann
10  * @version 1.0
11  */
12 public interface IServerDriver {
13
14      /**
15       * Starts an implementation of a bank server. Parameters which designate e.g. the port
16       * of the server and possibly other arguments may be passed.
17       *
18       * @param args array of implementation specific arguments
19       * @throws IOException if a communication problem occurs
20       */
21      public void start(String[] args) throws IOException;
22
23 }
```

Listing 6: Server Socket Driver

```
1  package bank.socket;
2
3  import java.io.IOException;
4  import java.io.ObjectInputStream;
5  import java.io.ObjectOutputStream;
6  import java.net.ServerSocket;
7  import java.net.Socket;
8
9  import bank.IBank;
10 import bank.IServerDriver;
11 import bank.StartClient;
12 import bank.communication.answer.IAnswer;
13 import bank.communication.request.IRequest;
14 import bank.local.Bank;
15
16 /**
17  * This class provides an implementation of the IServerDriver interface which creates a
18  * ServerSocket.
19  *
20  * @see IServerDriver
21  * @author Thomas Baumann
22  * @version 1.1
23  */
24 public class ServerDriver implements IServerDriver {
25
26      private IBank bank = new Bank();
27
28      @Override
29      public void start(String[] args) throws IOException {
30          if (args.length < 1) {
31              System.out.println("Usage: java " + StartClient.class.getName() + " "
```

```
32                      + ServerDriver.class.getName() + " <portnumber>");
33              System.exit(1);
34          }
35          int port = 0;
36          try {
37              port = Integer.parseInt(args[0]);
38          } catch (NumberFormatException e) {
39              System.out.println("Port must be a number");
40              System.exit(1);
41          }
42
43          ServerSocket server = new ServerSocket(port);
44          try {
45              while (true) {
46                  try {
47                      Socket s = server.accept();
48                      s.setTcpNoDelay(true);
49                      Thread t = new Thread(new BankServerHandler(s));
50                      t.start();
51                  } catch (IOException e) {
52                      System.out.println("Problem while connection to a client");
53                      e.printStackTrace();
54                  }
55
56              }
57          } finally {
58              server.close();
59          }
60      }
61
62      private class BankServerHandler implements Runnable {
63          private Socket socket;
64          private ObjectInputStream oin;
65          private ObjectOutputStream oout;
66
67          public BankServerHandler(Socket s) throws IOException {
68              this.socket = s;
69              this.oout = new ObjectOutputStream(this.socket.getOutputStream());
70              this.oin = new ObjectInputStream(this.socket.getInputStream());
71          }
72
73          @Override
74          public void run() {
75              try {
76                  boolean connected = true;
77                  while (connected) {
78                      try {
79                          Object o = this.oin.readObject();
80                          if (o != null) {
81                              IAnswer<?> answer = ((IRequest) o)
82                                      .handleRequest(ServerDriver.this.bank);
83                              this.oout.writeObject(answer);
84                              this.oout.flush();
85                          } else {
86                              connected = false;
87                          }
88                      } catch (ClassNotFoundException e) {
89                          System.out.println("invalid object arrived");
90                          e.printStackTrace();
91                      }
92                  }
93                  this.socket.close();
94              } catch (IOException e) {
95                  System.out.println("problem with the connection occured");
96                  e.printStackTrace();
97              }
98          }
99
100     }
101
102 }
```

4.3 Kommunikation Objekte

4.3.1 Interface

Listing 7: Interface für Request Objekte

```
1  package bank.communication.request;
2
3  import java.io.Serializable;
4
5  import bank.IBank;
6  import bank.communication.answer.IAnswer;
7
8  /**
9   * This interface must be used as request object for the socket communication from the
10  * client to the server.
11  *
12  * @author Thomas Baumann
13  * @version 1.1
14  */
15 public interface IRequest extends Serializable {
16
17      /**
18       * Handles the request with the specified bank.
19       *
20       * @param b Bank to hande the request
21       * @return answer object to send back
22       */
23      public IAnswer<?> handleRequest(IBank b);
24
25 }
```

Listing 8: Interface für Answer Objekte

```
1  package bank.communication.answer;
2
3  import java.io.IOException;
4  import java.io.Serializable;
5
6  import bank.InactiveException;
7  import bank.OverdrawException;
8
9  /**
10  * This interface must be used as answer object for the object communication from the
11  * server to the client.
12  *
13  * @author Thomas Baumann
14  * @version 1.1
15  * @param <T>
16  */
17 public interface IAnswer<T> extends Serializable {
18      /**
19       * Returns an object or throws an exception.
20       *
21       * @return Returns the answer
22       * @throws IllegalArgumentException When answer is an IllegalArgumentException
23       * @throws IOException When an IO problem occurs
24       * @throws OverdrawException When answer is an OverdrawException
25       * @throws InactiveException When answer is an InactiveException
26       */
27      public T getData() throws IllegalArgumentException, IOException, OverdrawException,
28              InactiveException;
29
30 }
```

4.3.2 Anfrage Objekte

Listing 9: Anfrage Objekt um Konto zu eröffnen

```java
package bank.communication.request;

import java.io.IOException;

import bank.IBank;
import bank.communication.answer.Answer;
import bank.communication.answer.IAnswer;
import bank.communication.answer.IOExceptionAnswer;

/**
 * This class provides a create account request.
 *
 * @see IRequest
 * @author Thomas Baumann
 * @version 1.1
 */
public class CreateAccountRequest implements IRequest {

    private String owner;

    public CreateAccountRequest(String owner) {
        this.owner = owner;
    }

    @Override
    public IAnswer<?> handleRequest(IBank b) {
        try {
            String s = b.createAccount(this.owner);
            return new Answer<String>(s);
        } catch (IOException e) {
            return new IOExceptionAnswer(e);
        }
    }

}
```

Listing 10: Anfrage Objekt um Konto zu schliessen

```java
package bank.communication.request;

import java.io.IOException;

import bank.IBank;
import bank.communication.answer.Answer;
import bank.communication.answer.IAnswer;
import bank.communication.answer.IOExceptionAnswer;

/**
 * This class provides a close account request.
 *
 * @see IRequest
 * @author Thomas Baumann
 * @version 1.1
 */
public class CloseAccountRequest implements IRequest {

    private String number;

    public CloseAccountRequest(String number) {
        this.number = number;
    }

    @Override
    public IAnswer<?> handleRequest(IBank b) {
        try {
            boolean ans = b.closeAccount(this.number);
            return new Answer<Boolean>(ans);
        } catch (IOException e) {
            return new IOExceptionAnswer(e);
        }
```

```
33      }
34
35  }
```

Listing 11: Anfrage Objekt um Konto abzufragen

```
1  package bank.communication.request;
2
3  import java.io.IOException;
4
5  import bank.IBank;
6  import bank.communication.answer.Answer;
7  import bank.communication.answer.IAnswer;
8  import bank.communication.answer.IOExceptionAnswer;
9
10 /**
11  * This class provides a get account request.
12  *
13  * @see IRequest
14  * @author Thomas Baumann
15  * @version 1.1
16  */
17 public class GetAccountRequest implements IRequest {
18
19     private String number;
20
21     public GetAccountRequest(String number) {
22         this.number = number;
23     }
24
25     @Override
26     public IAnswer<?> handleRequest(IBank b) {
27         try {
28             return new Answer<Boolean>(b.getAccount(this.number) != null);
29         } catch (IOException e) {
30             return new IOExceptionAnswer(e);
31         }
32     }
33
34 }
```

Listing 12: Anfrage Objekt um Kontonummer abzufragen

```
1  package bank.communication.request;
2
3  import java.io.IOException;
4  import java.util.Set;
5
6  import bank.IBank;
7  import bank.communication.answer.Answer;
8  import bank.communication.answer.IAnswer;
9  import bank.communication.answer.IOExceptionAnswer;
10
11 /**
12  * This class provides a get account numbers request.
13  *
14  * @see IRequest
15  * @author Thomas Baumann
16  * @version 1.1
17  */
18 public class GetAccountNumbersRequest implements IRequest {
19
20     @Override
21     public IAnswer<?> handleRequest(IBank b) {
22         try {
23             Set<String> s = b.getAccountNumbers();
24             return new Answer<Set<String>>(s);
25         } catch (IOException e) {
26             return new IOExceptionAnswer(e);
```

```
27          }
28
29      }
30
31  }
```

Listing 13: Anfrage Objekt um Geld zu transferieren

```
1  package bank.communication.request;
2
3  import java.io.IOException;
4
5  import bank.IAccount;
6  import bank.IBank;
7  import bank.InactiveException;
8  import bank.OverdrawException;
9  import bank.communication.answer.Answer;
10 import bank.communication.answer.IAnswer;
11 import bank.communication.answer.IOExceptionAnswer;
12 import bank.communication.answer.IllegalArgumentExceptionAnswer;
13 import bank.communication.answer.InactiveExceptionAnswer;
14 import bank.communication.answer.OverdrawExceptionAnswer;
15
16 /**
17  * This class provides a transfer request.
18  *
19  * @see IRequest
20  * @author Thomas Baumann
21  * @version 1.1
22  */
23 public class TransferRequest implements IRequest {
24
25     private String numberFrom;
26     private String numberTo;
27     private Double amount;
28
29     public TransferRequest(String numberFrom, String numberTo, Double amount) {
30         this.numberFrom = numberFrom;
31         this.numberTo = numberTo;
32         this.amount = amount;
33     }
34
35     @Override
36     public IAnswer<?> handleRequest(IBank b) {
37         try {
38             IAccount f = b.getAccount(this.numberFrom);
39             IAccount t = b.getAccount(this.numberTo);
40             b.transfer(f, t, this.amount);
41             return new Answer<Object>(null);
42         } catch (IllegalArgumentException e) {
43             return new IllegalArgumentExceptionAnswer(e);
44         } catch (IOException e) {
45             return new IOExceptionAnswer(e);
46         } catch (OverdrawException e) {
47             return new OverdrawExceptionAnswer(e);
48         } catch (InactiveException e) {
49             return new InactiveExceptionAnswer(e);
50         }
51     }
52
53 }
```

Listing 14: Anfrage Objekt um Kontobesitzer abzufragen

```
1  package bank.communication.request;
2
3  import java.io.IOException;
4
5  import bank.IBank;
```

```java
import bank.communication.answer.Answer;
import bank.communication.answer.IAnswer;
import bank.communication.answer.IOExceptionAnswer;

/**
 * This class provides a get owner request for an account.
 *
 * @see IRequest
 * @author Thomas Baumann
 * @version 1.1
 */
public class GetOwnerRequest implements IRequest {

    private String number;

    public GetOwnerRequest(String number) {
        this.number = number;
    }

    @Override
    public IAnswer<?> handleRequest(IBank b) {
        try {
            String owner = b.getAccount(this.number).getOwner();
            return new Answer<String>(owner);
        } catch (IOException e) {
            return new IOExceptionAnswer(e);
        }
    }

}
```

Listing 15: Anfrage Objekt um Kontostand abzufragen

```java
package bank.communication.request;

import java.io.IOException;

import bank.IBank;
import bank.communication.answer.Answer;
import bank.communication.answer.IAnswer;
import bank.communication.answer.IOExceptionAnswer;

/**
 * This class provides a get balance request for an account.
 *
 * @see IRequest
 * @author Thomas Baumann
 * @version 1.1
 */
public class GetBalanceRequest implements IRequest {

    private String number;

    public GetBalanceRequest(String number) {
        this.number = number;
    }

    @Override
    public IAnswer<?> handleRequest(IBank b) {
        try {
            Double balance = b.getAccount(this.number).getBalance();
            return new Answer<Double>(balance);
        } catch (IOException e) {
            return new IOExceptionAnswer(e);
        }
    }

}
```

Listing 16: Anfrage Objekt für Aktiv/Inaktiv Zustand

```java
package bank.communication.request;

import java.io.IOException;

import bank.IBank;
import bank.communication.answer.Answer;
import bank.communication.answer.IAnswer;
import bank.communication.answer.IOExceptionAnswer;

/**
 * This class provides an is active request for an account.
 *
 * @see IRequest
 * @author Thomas Baumann
 * @version 1.1
 */
public class IsActiveRequest implements IRequest {

    private String number;

    public IsActiveRequest(String number) {
        this.number = number;
    }

    @Override
    public IAnswer<?> handleRequest(IBank b) {
        try {
            boolean ans = b.getAccount(this.number).isActive();
            return new Answer<Boolean>(ans);
        } catch (IOException e) {
            return new IOExceptionAnswer(e);
        }
    }

}
```

Listing 17: Anfrage Objekt um Geld abzuheben

```java
package bank.communication.request;

import java.io.IOException;

import bank.IBank;
import bank.InactiveException;
import bank.communication.answer.Answer;
import bank.communication.answer.IAnswer;
import bank.communication.answer.IOExceptionAnswer;
import bank.communication.answer.IllegalArgumentExceptionAnswer;
import bank.communication.answer.InactiveExceptionAnswer;

/**
 * This class provides a deposit request for an account.
 *
 * @see IRequest
 * @author Thomas Baumann
 * @version 1.1
 */
public class DepositRequest implements IRequest {

    private String number;
    private Double amount;

    public DepositRequest(String number, Double amount) {
        this.number = number;
        this.amount = amount;
    }

    @Override
```

```
31      public IAnswer<?> handleRequest(IBank b) {
32          try {
33              b.getAccount(this.number).deposit(this.amount);
34              return new Answer<Object>(null);
35          } catch (IOException e) {
36              return new IOExceptionAnswer(e);
37          } catch (IllegalArgumentException e) {
38              return new IllegalArgumentExceptionAnswer(e);
39          } catch (InactiveException e) {
40              return new InactiveExceptionAnswer(e);
41          }
42
43      }
44
45  }
```

Listing 18: Anfrage Objekt um Geld einzuzahlen

```
1  package bank.communication.request;
2
3  import java.io.IOException;
4
5  import bank.IBank;
6  import bank.InactiveException;
7  import bank.OverdrawException;
8  import bank.communication.answer.Answer;
9  import bank.communication.answer.IAnswer;
10 import bank.communication.answer.IOExceptionAnswer;
11 import bank.communication.answer.IllegalArgumentExceptionAnswer;
12 import bank.communication.answer.InactiveExceptionAnswer;
13 import bank.communication.answer.OverdrawExceptionAnswer;
14
15 /**
16  * This class provides a withdraw request for an account.
17  *
18  * @see IRequest
19  * @author Thomas Baumann
20  * @version 1.1
21  */
22 public class WithdrawRequest implements IRequest {
23
24      private String number;
25      private Double amount;
26
27      public WithdrawRequest(String number, Double amount) {
28          this.number = number;
29          this.amount = amount;
30      }
31
32      @Override
33      public IAnswer<?> handleRequest(IBank b) {
34          try {
35              b.getAccount(this.number).withdraw(this.amount);
36              return new Answer<Object>(null);
37          } catch (IOException e) {
38              return new IOExceptionAnswer(e);
39          } catch (IllegalArgumentException e) {
40              return new IllegalArgumentExceptionAnswer(e);
41          } catch (OverdrawException e) {
42              return new OverdrawExceptionAnswer(e);
43          } catch (InactiveException e) {
44              return new InactiveExceptionAnswer(e);
45          }
46      }
47
48  }
```

4.3.3 Antwort Objekte

Listing 19: Allgemeines Antwort Objekt

```java
package bank.communication.answer;

/**
 * This class provides an answer object. It includes a variable to save some value.
 *
 * @see IAnswer
 * @author Thomas Baumann
 * @version 1.1
 */
public class Answer<T> implements IAnswer<T> {
    private T value;

    public Answer(T value) {
        this.value = value;
    }

    @Override
    public T getData() {
        return this.value;
    }

}
```

Listing 20: Antwort Objekt für IllegalArgumentException

```java
package bank.communication.answer;

/**
 * This class provides a IllegalArgumentException answer. It includes an exception of the
 * type illegal argument. When the getData method will be called the exception will be
 * thrown.
 *
 * @see IAnswer
 * @author Thomas Baumann
 * @version 1.1
 */
public class IllegalArgumentExceptionAnswer implements IAnswer<Object> {
    private IllegalArgumentException e;

    public IllegalArgumentExceptionAnswer(IllegalArgumentException e) {
        this.e = e;
    }

    @Override
    public Object getData() throws IllegalArgumentException {
        throw this.e;
    }

}
```

Listing 21: Antwort Objekt für InactiveException

```java
package bank.communication.answer;

import bank.InactiveException;

/**
 * This class provides a InactiveException answer. It includes an exception of the type
 * inactive. When the getData method will be called the exception will be thrown.
 *
 * @see IAnswer
 * @author Thomas Baumann
 * @version 1.1
 */
public class InactiveExceptionAnswer implements IAnswer<Object> {
    private InactiveException e;

```

```
16      public InactiveExceptionAnswer(InactiveException e) {
17          this.e = e;
18      }
19
20      @Override
21      public Object getData() throws InactiveException {
22          throw this.e;
23      }
24
25  }
```

Listing 22: Antwort Objekt für IOException

```
1  package bank.communication.answer;
2
3  import java.io.IOException;
4
5  /**
6   * This class provides a IOException answer. It includes an exception of the type IO. When
7   * the getData method will be called the exception will be thrown.
8   *
9   * @see IAnswer
10  * @author Thomas Baumann
11  * @version 1.1
12  */
13 public class IOExceptionAnswer implements IAnswer<Object> {
14      private IOException e;
15
16      public IOExceptionAnswer(IOException e) {
17          this.e = e;
18      }
19
20      @Override
21      public Object getData() throws IOException {
22          throw this.e;
23      }
24
25 }
```

Listing 23: Antwort Objekt für OverdrawException

```
1  package bank.communication.answer;
2
3  import bank.OverdrawException;
4
5  /**
6   * This class provides a OverdrawException answer. It includes an exception of the type
7   * Overdraw. When the getData method will be called the exception will be thrown.
8   *
9   * @see IAnswer
10  * @author Thomas Baumann
11  * @version 1.1
12  */
13 public class OverdrawExceptionAnswer implements IAnswer<Object> {
14      private OverdrawException e;
15
16      public OverdrawExceptionAnswer(OverdrawException e) {
17          this.e = e;
18      }
19
20      @Override
21      public Object getData() throws OverdrawException {
22          throw this.e;
23      }
24
25 }
```