

**Prüfung vom 12. Mai 2017**  
**Teil 1: 30 Minuten**

Note 5,875

Di

Name, Vorname:

Wächter Simon

**Allgemeine Hinweise:**

- 1) Diese Prüfung besteht aus zwei Teilen.
- 2) Für diesen ersten Teil der Prüfung sind keine Unterlagen erlaubt.
- 3) Bitte beantworten Sie die Fragen dieses ersten Teiles direkt auf dem Aufgabenblatt.
- 4) Für diesen ersten Teil haben Sie 30 Minuten Zeit, für den darauffolgenden zweiten Teil 60Min.

4

Viel Erfolg!

**Aufgabe 1: Network Programming Basics**

(4 Punkte)

Kreuzen Sie für alle folgenden Aussagen entweder *stimmt* oder *stimmt nicht* an. Aussagen ohne Kreuz, mit zwei Kreuzen oder unklar angekreuzte Felder werden neutral mit 0 Punkten bewertet. Lesen Sie die Aussagen genau durch! (0.5 Punkte pro richtige Antwort, 0.5 Punkte Abzug pro falsche Antwort, min. 0 Punkte)

Aussage	stimmt	stimmt nicht
Die WSDL-Datei eines SOAP Services beschreibt verschiedene Ressourcen mit den auf diesen Ressourcen anwendbaren Operationen (GET, PUT, POST, DELETE etc) und den für jede Operation möglichen HTTP Antwort-Codes. <i>→ Res!</i>		X
HTTP Caching kann für SOAP Methodenaufrufe mit HTTP-Binding nicht verwendet werden.	X	
Für jede auf einem Server eingehende TCP-Verbindung verwendet der Server einen neuen (zufällig gewählten) Port. <i>local Port für neuen Socket nach Accept</i>	X	X
Eine Instanz der Klasse ServerSocket kann auf mehreren Ports auf eingehende Socket-Verbindungen warten, d.h. kann gleichzeitig an mehrere Ports gebunden werden.		X
Falls ein Klient über einen Socket alle Daten an einen Server geschickt hat aber noch auf Antworten warten muss, dann ruft er sinnvollerweise shutdownOutput auf dem Socket auf. <i>socket.getLocalPort</i>	X	
Wenn folgende Anweisung ausgeführt wird: <code>int port = new ServerSocket(1234).accept().getLocalPort();</code> dann ist (wenn die Anweisung nicht mit einer Exception abbricht) in der Variablen port garantiert der Wert 1234 gespeichert.	X	X
Eine Instanz der Klasse DatagramSocket kann gleichzeitig auf mehreren Ports auf eingehende Socketverbindungen warten, d.h. kann gleichzeitig an mehrere Ports gebunden werden.		X
Die Methode disconnect der Klasse DatagramSocket schliesst auf einem Socket eine offene Verbindung zum Server. Sobald ein Socket „disconnected“ ist, kann er weder für das Senden noch für das Empfangen von Datagram-Paketen verwendet werden.	X	X

Welcher Server?





**Aufgabe 2: HTTP****(1+1+1+1 = 4 Punkte)**

Mit dem TCP-Monitor wurde folgender HTTP-Request einer REST Anfrage auf das Doodle-API abgefangen:

PUT /polls/udw26k7f8wwgghg/participants/1874832849 HTTP/1.1

- (1) Host: doodle.com  
 User-Agent: Chrome/58.0.3029.81 Safari/537.36  
 (2) Accept: application/json  
 Content-Type: application/json  
 Referer: www.doodle.com  
 (3) Transfer-Encoding: chunked

40  
 {  
 "participant": {  
 "name": "Paul",  
 "preferences": {  
 (4) 42  
 "option": [  
 "1", "1", "0", "0"  
 ]  
 }  
 }  
 }  
 0

Beschreiben Sie kurz, was die mit (1) bis (4) markierten Zeilen im HTTP-Request bedeuten, d.h. wozu diese Information auf Server-Seite verwendet wird oder was sie konkret beschreiben.

- (1) Beschreibt den aufgerufenen Host / DNS im Client und ermöglicht so Virtual Hosts auf dem Server, da DNS dem Server nur Request IP gibt, nicht aber die TLD / Host selber
- (2) Gibt die MIME Types an, die der Client unterstützt und erwartet in der Response.
- (3) Gibt die "Formatierung" / Aufbau des Bodies an. Hier wird der Content in chunks ~~ge~~ zerlegt und mit einem Hash prepended, 0 schliesst ab. Dies ermöglicht eine Art Stream-basiertes Schreiben, nicht wie bei Content-length.
- (4) Das 42 stellt die checksumme / hash des zweiten Chunks dar. In unserem Fall wird der REST JSON Request halb zerlegt.
- nein, ist Länge des next chunk.



### Aufgabe 3: HTTP & REST Optimierungen

(2+2 = 4 Punkte)

In dieser Aufgabe betrachten wir zwei weitere HTTP-Request-Header. Beschreiben Sie die Bedeutung dieser beiden Header und geben Sie für beide ein kurzes Anwendungsbeispiel an, d.h. es soll klar werden, wozu diese Felder verwendet werden können und wie dieses Ziel erreicht wird. Das Anwendungsbeispiel können Sie z.B. aus dem Kontext der REST-Bank-Übung wählen. Geben Sie auch an, welche Statuscodes als Antwort vorgesehen sind.

a) If-None-Match

Beispiel: Ich lese via GET Daten

Der Server kann in seiner Response mitteilen, dass es einen sogenannten ETag oder alternative ein Cache Control Date gibt. Diese Werte kann ich nehmen und beim nächsten Request zurücksenden. Für If-None-Match desselben Types/URL setze ich den ETag. Der Server prüft dann, ob der Hash des Responseobjektes dem ETag entspricht → Die Response wäre dann gleich und ich erhalte einen 304 Not Modified und kann den bei mir gecachten Datensatz weiterverwenden. Allenfalls gibt es neue Daten im Response body via 200 OK ✓

b) If-Match

2 → Ohm? Gegenteil von oben?  
Sinn ist also: Mögliches Caching.



**Aufgabe 4: REST****(3+1 = 4 Punkte)**

In dieser Aufgabe soll eine REST-Schnittstelle für den Ticketshop der SBB entworfen werden. Die Funktionen, die der Server dabei anbieten soll, sind in folgenden Java-Interfaces beschrieben:

```
interface TicketShop {
    Ticket createTicket(String from, String to);
}

interface Ticket {
    enum Type {ONEWAY, RETURN}
    enum Class {FIRST, SECOND}
    public void setTicketType(Type type);
    public void setClass(Class c);
    public void setHalfPrice(boolean halfPrice);
    public double getPrice();
    public void pay(String creditcardNr, String date, String cvv);
    public void cancel();
}
```

4  
Müsste setHalfPrice  
heissen 😊  
hat in  
normalen

- Über das Interface TicketShop können neue Tickets erzeugt werden, als Parameter werden Ausgangs- und Zielort angegeben.
- Über das Interface Ticket kann eingestellt werden, ob man ein Einfach- oder Retoureticket kaufen will (setTicketType), in welcher Klasse man fahren will (setClass) und ob man ein Halbpreisabo besitzt oder nicht (setHalfPrice). Die Methode getPrice gibt den für die jeweiligen Einstellungen gültigen Preis an. Mit der Methode pay kann das Ticket bezahlt werden (der Betrag wird dabei von der Kreditkarte abgebucht) und mit der Methode cancel kann der Kaufprozess abgebrochen werden. Nach Aufruf von pay oder cancel werfen die Operationen getPrice, setHalfPrice, setClass und setTicketType eine Exception.

In dieser Aufgabe soll der Server jedoch mit REST modelliert werden (im Sinne der REST Philosophie).

- a) Geben Sie an, welche Ressourcen (d.h. welche Pfade) Sie für diese Applikation bereitstellen, welche HTTP-Operationen Sie auf diesen Ressourcen unterstützen und welche Semantik diese Operationen im Kontext der Ticket-Applikation haben (also was für Funktionen damit ausgelöst werden können) sowie welche HTTP Resultat-Codes sie zurückliefern können (allenfalls Rückseite verwenden).

POST ✓ /ticket 202: create Ticket → 110/TicketObj.  
GET ✓ /ticket/info/{id} 200: getPrice.  
PUT ✓ /ticket/info/{id} 200: setTicketType, setClass, setHalfPrice  
DELETE ✓ /ticket/info/{id} 204: cancel  
POST ✓ /ticket/info/{id} 200: pay(...) ⇒ Alle IDs können auch 409 oder 408 zurück geben  
alternativ eine 3. Ressource

- b) Was bedeutet es wenn eine Operation idempotent ist, und welche der in a) definierten Operationen sind idempotent?

Die Operation ist n Mal wiederholbar und führt immer zum gleichen Resultat (in der Ressource z.B.)  
⇒  $f(f(x)) = f(x)$  ✓  
⇒ HTTP Verben wie GET, PUT oder DELETE  
⇒ GET liefert immer reproduzierbares Resultat und kann deshalb gecached werden

## Prüfung vom 12. Mai 2017

### Teil 2: 60 Minuten

Name, Vorname:

Wächter Simon.

#### Allgemeine Hinweise:

- 1) Bitte starten Sie jede Aufgabe auf einem neuen Blatt. Schreiben Sie auf jedes Blatt Ihren Namen.
- 2) Pro Aufgabe darf höchstens ein gültiger Lösungsversuch abgegeben werden. Ungültige Lösungsversuche müssen klar durchgestrichen sein!
- 3) Lesen Sie eine Aufgabe genau durch bevor Sie sie zu lösen beginnen.
- 4) Insgesamt haben Sie für diesen Teil 60 Minuten Zeit.

Viel Erfolg!

13 / 13



**Aufgabe 5: HTTP-URLConnection / Servlets****(6+6+2= 14 Punkte)**

Wenn man mit dem Browser eine Seite aufruft und diese als Antwort ein

HTTP/1.1 301 Moved Permanently

oder

HTTP/1.1 302 FOUND

zurückgibt (allgemein: Eine Antwort mit dem Status-Code 3XX), so ruft der Browser automatisch die im Location-Header angegebene Seite auf. Die Klasse `URLConnection` folgt auch automatisch den Redirects (bzw. man kann die Klasse entsprechend konfigurieren), aber für diese Aufgabe nehmen wir an, dass die Klasse `URLConnection` keine Redirects auflöst.

Aufgaben:

- a) Schreiben Sie eine Methode `followRedirects` welche für HTTP-Requests die Redirects auflöst, bis der HTTP-Status-Code nicht mehr 3XX ist oder bis `maxFollows` Redirects ausgeführt worden sind. Beim Aufruf sind alle Request-Parameter auf der übergebenen `connection` gesetzt.

```
URLConnection followRedirects(URLConnection connection, int maxFollows)
    throws Exception {
    // ...
}
```

Beachten Sie, dass für jede neue Anfrage an den Server eine neue Instanz der Klasse `URLConnection` erzeugt werden muss. Dabei soll

- die HTTP-Methode
- die Request-Header-Felder

von der ursprünglichen Anfrage übernommen werden. Wir nehmen jedoch an, dass keine Daten im Rumpf geschrieben worden sind.

Bemerkung:

- Das API der Klasse `URLConnection` (Auszug) ist auf der folgenden Seite angegeben.

- b) Schreiben Sie ein einfaches Servlet, welches Anfragen unter der URL

`http://<host>:8080/redirect/<n>`

entgegen nimmt und als Antwort ein Redirect (301) auf die URL

`http://<host>:8080/redirect/<n-1>`

zurückliefert (falls  $n > 1$ ) ist bzw. auf die URL

`http://<host>:8080/echo/`

falls  $n=1$  ist (das Echo-Servlet müssen Sie nicht programmieren).

Das Servlet soll dabei alle Anfragen beantworten (GET, PUT, POST, DELETE etc). Auf Fehlerbehandlungen können Sie verzichten.

- c) Wir wollen das Redirect-Servlet als Web-Applikation auf Tomcat bereitstellen.

- Geben Sie die Struktur der Webapplikation an (als Verzeichnisstruktur). Es ist Ihnen überlassen, ob sie die Metainformationen über Annotationen oder mit einem Konfigurationsfile definieren.
- Ergänzen Sie ihr Servlet aus Aufgabe b) mit den nötigen Annotationen oder geben Sie das Konfigurationsfile an.

## Method Summary

boolean	<u>getDoInput()</u> Returns the value of this <code>URLConnection</code> 's <code>doInput</code> flag.
boolean	<u>getDoOutput()</u> Returns the value of this <code>URLConnection</code> 's <code>doOutput</code> flag.
void	<u>setDoInput(boolean doinput)</u> Sets the value of the <code>doInput</code> field for this <code>URLConnection</code> to the specified value.
void	<u>setDoOutput(boolean dooutput)</u> Sets the value of the <code>doOutput</code> field for this <code>URLConnection</code> to the specified value.
<code>Map&lt;String, List&lt;String&gt;&gt;</code>	<u>getRequestProperties()</u> Returns an unmodifiable <code>Map</code> of general request properties for this connection. The <code>Map</code> keys are <code>Strings</code> that represent the request-header field names. Each <code>Map</code> value is a unmodifiable <code>List</code> of <code>Strings</code> that represents the corresponding field values.
<u>String</u>	<u>getRequestProperty(String key)</u> Returns the value of the named general request property for this connection.
void	<u>setRequestProperty(String key, String value)</u> Sets the general request property.
void	<u>addRequestProperty(String key, String value)</u> Adds a general request property specified by a key-value pair.
<u>String</u>	<u>getHeaderField(String name)</u> Returns the value of the named response header field.
<u>String</u>	<u>getRequestMethod()</u> Get the request method.
void	<u>setRequestMethod(String method)</u> Set the method for the URL request, one of: GET POST HEAD OPTIONS PUT DELETE TRACE are legal, subject to protocol restrictions.
<u>InputStream</u>	<u>getInputStream()</u> Returns an input stream that reads from this open connection.
<u>OutputStream</u>	<u>getOutputStream()</u> Returns an output stream that writes to this connection.
int	<u>getResponseCode()</u> Gets the status code from an HTTP response message.
<u>String</u>	<u>getResponseMessage()</u> Gets the HTTP response message, if any, returned along with the response code from a server.
<u>URL</u>	<u>getURL()</u> Returns the value of this <code>URLConnection</code> 's <code>URL</code> field.
abstract void	<u>connect()</u> Opens a communications link to the resource referenced by this <code>URL</code> , if such a connection has not already been established.

$$6 + 1/2 + 1/2 = 13$$



# Aufgabe 5

```
a) HttpURLConnection followRedirects(...) {  
    int current attempts = 0 int current attempts = maxFollows;  
    if (maxFollows == 0) {  
        if (current attempts == maxFollows) {  
            return null; oder throw new Exception();  
            ↳ Etwas unklar ✓  
        }  
        current attempts --;  
        HttpURLConnection urlc = new HttpURLConnection (  
            (asb (HttpUrl (connection).  
            connection.getUri().open  
                Connection());  
        1d urlc.setRequestMethod(connection.getRequest  
            for (Key <String, String> key Method);  
        1e urlc.setRequestProperty(key.getKey(), key.  
            RequestProperties());  
            get Value());  
        }  
        1f int value = urlc.getResponseCode();  
        if (value ≥ 300 & & value ≤ 399) {  
            1b String newLocation = urlc.getHeaderField(  
                "Location");  
            1c URL url = new URL(newLocation);  
            1a return followRedirects(urlc, current attempts);  
        } else {  
            return urlc;  
        }  
    }  
}
```



b) @ WebServlet ("/redirect")  
public class MyServlet extends HttpServlet {

*Path wie ~~et~~ etw.*

@ override

1/2 public doService (...) {

// Unbekannt: Server IP, halt ein setzen  
RequestUri

String uri = request.getRequestUri();  
String[] params = uri.split("/");  
int value = Integer.parseInt(params[1]);

if (value > 1) {

value --;

response.setHeader("Location", "http://<  
IP of server > /redirect" + value);

} else {

response.setHeader("Location", "http://  
IP of server > /echo");  
response.setStatus(301);

}

c) Gar nicht. Da heute Verwaltbarkeit / Unterhalt  
und einfaches Deployen wichtiger sind als  
ein "statischer" Applicationserver mit fixer  
Servletversion, würde ich einen Servlet  
Server integrieren, also eine ausführ-  
bare UBER-JAR à la Spring Boot  
erstellen, sonst

weapps  
redirect ✓ 1/2  
echo ✓ 1/2  
1. class / classes  
WEB-INF  
lib  
\*.jar



**Aufgabe 6: Bank REST****(6+10 =16 Punkte)**

In den Übungen haben Sie die Bank mit Hilfe von REST realisiert. Sie haben dabei das BankDriver-Interface realisiert. In dieser Aufgabe wollen wir auf der Basis von REST ein vereinfachtes BankDriver-Interface realisieren. Wir beschränken uns dabei auf das Erzeugen, Abfragen und Löschen von Kontos, aber zusätzlich soll ein Update-Handler registriert werden können. Dieser Handler wird immer dann aufgerufen, wenn auf dem Server (über irgend einen Klienten) ein neues Konto erzeugt oder ein existierendes gelöscht worden ist.

```
public interface BankDriver {
    void connect();

    String createAccount(String name);
    void removeAccount(String id);
    Set<String> getAccounts();

    void registerUpdateHandler(UpdateHandler handler);

    interface UpdateHandler { void accountsChanged(); }
}
```

Ein Klientenprogramm könnte wie folgt aussehen:

```
public class Client1 {

    public static void main(String[] args) throws Exception {
        BankDriver driver = new BankDriverImpl();
        driver.connect();

        driver.registerUpdateHandler(() -> {
            System.out.println("accounts changed: " + driver.getAccounts());
        });

        String id = driver.createAccount("Peter");
        Thread.sleep(2000);
        driver.removeAccount(id);
        Thread.sleep(2000);
    }
}
```

Für die Implementierung des Update-Handlers starten Sie (im Driver auf Klientenseite) einen Thread, der jede Sekunde die Kontis auf dem Server abholt und prüft, ob sich etwas geändert hat ( $\Rightarrow$  Polling). Falls ja, dann soll die Methode `accountsChanged()` aufgerufen werden. Wir nehmen an, dass im Driver höchstens ein Update-Handler registriert wird. Damit Ressourcen geschont werden soll beim Polling nicht die Kontis abgefragt werden, sondern es soll nur geprüft werden, ob sich der ETag der Ressource accounts geändert hat (d.h. arbeiten Sie mit einem Conditional-GET).

Die Implementierung des Servers finden Sie auf der Rückseite. Was noch fehlt ist die Methode für die Abfrage der Kontis (`getAccounts`). Die Kontis werden als Instanz der Klasse `Set<String>` zurückgegeben (kann in „application/json“ serialisiert werden). Auf dieses Set kann mit `accounts.keySet()` zugegriffen werden.

**Aufgaben:**

- Implementieren Sie den Fehlenden Teil des Servers (nur Abfrage der Kontonummern). Bei Bedarf dürfen Sie die existierenden Methoden auch ändern oder ergänzen.
- Implementieren Sie die Klasse `BankDriverImpl` welche das Interface `BankDriver` implementiert (alle Methoden aus dem Interface). Sie verwenden dazu am einfachsten das Jersey Client API.



```
@Path("bank/accounts")
@Singleton
public class BankResource {

    static class Account {
        public Account(String id, String name, double balance) {
            this.id = id;
            this.name = name;
            this.balance = balance;
        }
        public final String id;
        public final String name;
        public double balance; // wird in dieser Aufgabe nicht verwendet
    }

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

    @POST
    public String createAccount(String name) {
        String key = "100-" + id.incrementAndGet();
        accounts.put(key, new Account(key, name, 0));
        return key;
    }

    @DELETE
    @Path("{id}")
    public void deleteAccount(@PathParam("id") String id) {
        accounts.remove(id);
    }

    // ...
}
```

Ichnehme an,  
hier fehlt noch  
@FormParam("name")

nein, das sind die Daten  
die per POST über-  
tragen werden.

```
public class Server {

    public static void main(String[] args) throws IOException {
        final String baseUrl = "http://localhost:9999/";
        final ResourceConfig rc = new ResourceConfig().packages("bank.rest");
        HttpServer httpServer = GrizzlyHttpServerFactory.createHttpServer(
            URI.create(baseUrl), rc);

        System.in.read();
        httpServer.shutdown();
    }
}
```

#### Bemerkungen:

- Wenn auf dem Server eine Methode für GET-Requests implementiert ist, so kann auf dieser Resource auch ein HEAD-Request abgesetzt werden.
- Die Kommunikation soll mit application/json erfolgen.

## Aufgabe 6

a) Annahme: Server/Client haben z.B. Jackson JSON  
Serialisierer im Classpath ✓

<snipped>

```
@GET @Produces("application/json") @Context Request  
public Set<String> getAccounts () {
```

```
    CacheControl cc = new CacheControl();
```

```
    cc.setMaxAge(1000);
```

→ optional, wenn  
nur ETags gewollt sind  
weglassen. 1

```
    ResponseBuilder builder = request.evaluatePreconditions  
        (new EntityTag (
```

```
            Integer.parseInt(bank.getAccounts().  
                hashcode()) );
```

1

```
    if (builder != null) {  
        1 builder.cacheControl(cc);  
        return builder.build();
```

1

```
    }
```

Entity Tag zwischen speichern...

04

```
    builder = Response.ok(response);
```

```
    builder.cacheControl(cc);
```

```
    builder.tag(1);
```

```
    return builder.build();
```

```
}
```

1/2

nicht definiert

6 1/2



b) public class BankDriverImpl implements BankDriver {  
 private WebTarget target;  
 @Override  
 public void connect () {  
 String host = "http://localhost:1234";  
 Client client = ClientBuilder.newClient ();  
 target = client.target (host);  
 }  
 }  
 // Allenfalls via Constructor setzen  
 // da Cell pld bank/errors

private Set<String> numbers;

@Override  
 public String createAccount (String name) {  
 Form form = new Form ();  
 f.param ("name", name);  
 Response r = target.request ().post (Entity.form (form));  
 return r.readEntity (String.class);  
 }  
 // 1/2 // 1/2 // 1/2

@Override  
 public void removeAccount (String id) {  
 Response r = target.path (id).request ().delete ();  
 }  
 // 1/2 // 1/2

@Override  
 public Set<String> getAccounts () {  
 Response r = target.request ().get ();  
 if (r.getStatus () == Status.NOT\_MODIFIED) {  
 return numbers;  
 } else {  
 numbers = r.readEntity (Set<String>);  
 return numbers;  
 }  
 }  
 }  
 // 1/2  
 // wie du bist sich das kein if none - value - Header liefert.

```
@Override
public void registerUpdateHandler (UpdateHandler handler)
{
```

```
    this.handler = handler;
    new Thread () → {
```

```
        Set<String> data = getAccounts();
```

```
        while (true) {
```

```
            if (compareSets (data, numbers)) {
```

```
                this.handler.accountsChanged();
            }
```

```
        }.start();
```

```
    }
```

OK, aber ihr get Accounts  
verändert KEIN read-only-GET.  
Dazu muss der Tag gespeichert  
werden.

== (1)

```
private UpdateHandler handler; 13
```

```
class Util Helper {
    public boolean compareSets (....) {
        // == 0 Match
        // < -1 > 1 Unmatch
        // => über alle Keys iterieren und diese
        // vergleichen
    }
```

gleich 2 Sets macht  
dies.