

Prüfung vom 30. April 2018

Teil 1: 30 Minuten

Name, Vorname:

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.

O

Viel Erfolg!

(4 Punkte)

Aufgabe 1: Network Programming Basics

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
Mit einem asynchronen Programmiermodell kann ein synchroner Methodenaufruf emuliert werden.	x	
Mit einem synchronen Programmiermodell kann ein asynchrones Programmiermodell emuliert werden.	x	x
Nehmen wir an, dass ein Programm P1 folgende Anweisung ausgeführt hat: <code>ServerSocket ss = new ServerSocket(2222);</code> Wird nun auf demselben Host aus einem Programm P2 (während Programm P1 noch läuft) die Anweisung <code>DatagramSocket ds = new DatagramSocket(2222);</code> ausgeführt, dann wird eine <code>java.net.BindException</code> geworfen.		x
Bevor ein Socket mit der Methode <code>close()</code> geschlossen wird muss auf allen über diesen Socket mit <code>getOutputStream()</code> erzeugten Streams die Methode <code>flush()</code> aufgerufen werden, sonst können Daten verloren gehen.	x	x
Die HTTP Methode PATCH (die bei REST verwendet wird um Ressourcen zu aktualisieren) ist per Definition idempotent.	x	x
Eine REST-Schnittstelle kann auf dem Server auch mit Hilfe eines Servlets implementiert werden.	x	
Wenn ein RMI Objekt auf einem öffentlichen Server bereitgestellt wird, dann kann dieses Objekt auch von einem Klienten genutzt werden, der in einem lokalen Netzwerk (hinter NAT) z.B. mit IP 192.168.0.13 läuft (vorausgesetzt natürlich dass auf dem Server der entsprechende Port frei ist).	x	
Java-Klassen, auf die mit RMI zugegriffen werden kann, müssen von der Klasse <code>UnicastRemoteObject</code> abgeleitet sein.	x	x

Aufgabe 2: HTTP

(1+1+1+1 = 4 Punkte)

Gegeben ist folgender HTTP-Request mit dem ein bestimmtes Konto abgefragt wird:

```
POST /bank/accounts HTTP/1.1
Host: ubs.com
User-Agent: curl/7.48.0
Accept: application/json
(1) Accept-Encoding: gzip
(2) Content-Type: application/x-www-form-urlencoded
Content-Length: 10
(3) If-Modified-Since: Wed, 18 Apr 2018 07:28:00 GMT
```

ID=2273455

- a) Was bedeutet die mit (1) markierte Zeile im HTTP-Request?

0

Sobald eine Antwort in Form von Json eintrifft, wird sie in gzip verschlüsselt. Nein, kein autoris...
Client unterstützt gzip, muss aber nicht.
L von wer?
sAo n KEINE Verantwortung.

- b) Wann wird der auf Zeile (2) im HTTP-Request angegebene Typ verwendet? Und welche Daten werden in diesem Fall im Body an den Server übertragen?

1/2

Wird beim Request verwendet. Es wird also ein Request mit diesem Typ gemacht. Im Body sind üblicherweise Formulardaten enthalten.
Request kommt von einem Formular.

1/2

- c) Beschreiben Sie kurz den Effekt des mit (3) markierten Headers im HTTP-Request.

0

Es wird ein "lastmodified" mitgeliefert. Falls es auf dem Server übereinstimmt wird das POST ausgeführt.
nein, da es i - mit - modified

- d) Welche erfolgreiche Antworten (abgesehen von Fehlern wie 404 Not Found) sind auf diese Konto-Abfrage (auf Grund des mit (3) markierten Headers) grundsätzlich möglich?

0

202 Accepted

301

201 OK

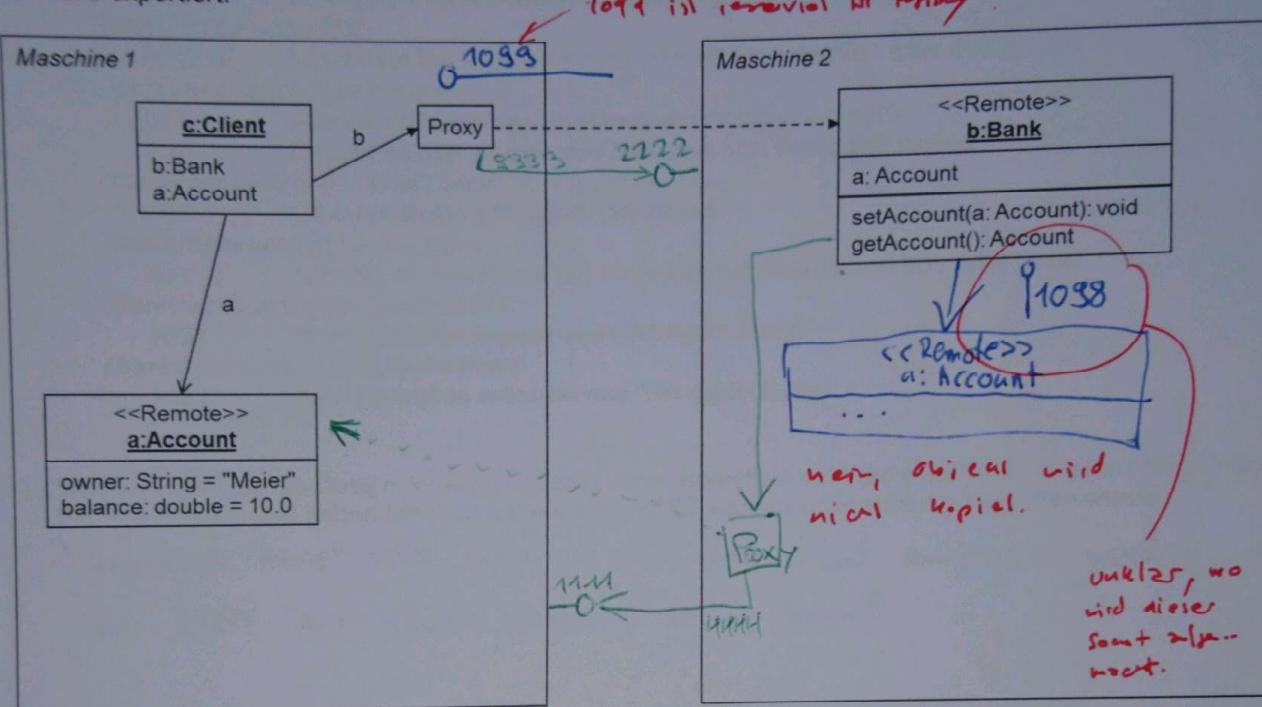
304 Not Modified

1/2

Aufgabe 3: RMI

(1+1+2 = 4 Punkte)

Gegeben ist die unten abgebildete Objektstruktur. Auf Maschine 2 läuft ein Server-Prozess, der das *exportierte* Remote-Objekte b vom Typ Bank enthält. Diese Bank kann genau ein Konto verwalten, welches mit den Methoden setAccount und getAccount gesetzt und gelesen werden kann, aber vorerst ist diese Referenz noch null. Im Client-Prozess auf Maschine 1 befindet sich das Objekt c vom Typ Client, welches eine Referenz auf das Server-Objekt b und auf eine Referenz a auf ein lokales Objekt vom Typ Account hat. Das Objekt a ist ebenfalls exportiert.



Fragen:

- a) Der Client c referenziert in seinem Prozess ein Proxy-Objekte. Welche Informationen sind in einem solchen Proxy Objekt enthalten?

Port, Host, Class
✓ ✓ Object ID

1/L

- b) Client c führt folgende Anweisung aus:

c.b.setAccount(c.a);

d.h. er übergibt die lokale Referenz c.a der Bank.

Zeichnen Sie in das obige Diagramm die dadurch auf Maschine 1 und 2 entstehende Objektstruktur (inkl. allfälliger Proxy-Objekte). Die Feldnamen müssen Sie beim Zeichnen neuer Objekte nicht angeben.

O

- c) Zeichnen Sie in obiges Diagramm auch ein, welche ServerSockets geöffnet sind (mit der Notation ○—, die Ports können Sie frei wählen) und geben Sie an, welche Socket-Verbindungen aktiv sind (bzw. regelmäßig aktiv werden), nachdem die in Aufgabe b) beschriebene Anweisung ausgeführt worden ist (geben Sie bei den Socket-Verbindungen an *beiden* Enden jeweils die Port-Nummer an).

1/L / 1

Aufgabe 4: REST

Für die Bank sei folgende REST-Schnittstelle definiert worden:

```
/bank/accounts
    GET      gibt die Kontonummern aller Kontis zurück
1) { /bank/accounts/{name}
    POST     erzeugt ein neues Konto mit den angegebenen Namen, Resultat 201 oder 400
/bank/accounts/{id}
    DELETE   Löscht ein bestimmtes Konto, gibt als Resultat 200 oder 406 zurück
→ /bank/accounts/{id}/owner
    GET      Gibt den Namen des Besitzers von Konto {id} zurück
    PUT      Erlaubt es den Namen des Besitzers von Konto {id} zu ändern
/bank/accounts/{id}/balance
    GET      Gibt den Saldo von Konto {id} zurück
/bank/accounts/{id}/active
    GET      Gibt an, ob das Konto {id} noch aktiv ist (Resultat ist true oder false)
/bank/accounts/{id}/deposit/
    POST    führt Operation deposit aus (Betrag im Body)
/bank/accounts/{id}/withdraw/
    POST    führt Operation withdraw aus (Betrag im Body)
```

- a) Beurteilen Sie dieses Design. Welche Definitionen entsprechen nicht der REST-Philosophie? Begründen Sie Ihre Antwort und geben Sie kurz an, wie eine REST-konforme Umsetzung aussehen würde.

1) Argument "name" sollte nicht so über den Pfad mitgegeben werden, sondern mit POST auf /bank/accounts?name={name}?

oder in Body

1

- Name → sollte in Body sein
- zu viele Pfade, die zusammengeführt werden könnten

- b) Was bedeutet es wenn eine Operation idempotent ist, und welche der gegebenen REST-Schnittstelle definierten Operationen sind idempotent?

Wenn bei der Operation immer dasselbe Resultat entsteht, unabhängig wie oft man es ausführt.

1

2

Aufgabe 5: HTTP-URLConnection / Servlets

(6+4 = 10 Punkte)

Gegeben ist folgendes Servlet welches Währungsanfragen beantworten kann:

```
@WebServlet("/convert")
public class Converter extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException {
        String amount = request.getParameter("amount");
        String from = request.getParameter("from");
        String to = request.getParameter("to");
        String res = computeResult(amount, from, to);

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.print("<html><body><h1>Currency Converter</h1>");
        out.printf(String.format("%s %s = %s %s", amount, from, res, to));
        out.println("</body></html>");
    }

    static String computeResult(String amount, String from, String to) { ... }
}
```

Dieses Servlet wird auf einem Servlet-Container als Web-Archiv unter dem Namen exam.war bereitgestellt (das war-File wird im webapps-Verzeichnis abgelegt). Der Servlet-Container nimmt HTTP-Anfragen auf dem Host **bank.fhnw.ch** unter Port **8080** entgegen.

Aufgaben:

- Schreiben Sie einen Java-Klienten, der mit Hilfe der Klasse HttpURLConnection auf dem Server **bank.fhnw.ch** abfragt, welches der aktuelle Wert von 100 CHF in EUR ist und die HTML-Antwort auf der Konsole ausgibt. Die Anfrageparameter müssen im Body des Requests übergeben werden.

Bemerkungen:

- Es genügt, wenn Sie den Rumpf der main-Methode angeben:

```
public static void main(String[] args) throws Exception {
    ...
}
```
 - Das API der Klasse HttpURLConnection (Auszug) ist auf Seite 5 angegeben.
 - Die Klasse HttpURLConnection nimmt gewisse Einstellungen automatisch vor. In dieser Aufgabe sollen Sie jedoch trotzdem explizit folgende Daten setzen:
 - o HTTP-Methode
 - o Content-Type
 - Die Daten im Body sollen mit **chunked-encoding** an den server geschickt werden.
- Geben Sie an, wie der HTTP Request (mit HTTP-Version 1.1) aussieht der bei dieser Anfrage an den Server geschickt wird. Die Header-Felder User-Agent, Accept und Connection können Sie dabei weglassen.

Aufgabe 6: Bank REST**(6 + 5 = 11 Punkte)**

In folgenden Code-Fragmenten ist eine REST-Ressource (mit Jersey-Annotationen) sowie ein Client gegeben. Die Ressource wird auf dem Server unter der Basis-URL localhost:9999/bank bereitgestellt. Der Client greift regelmässig auf die Ressource /accounts zu und gibt die Liste der Kontis auf der Konsole aus.

Server:

```
@Singleton
@Path("accounts")
public class BankResource {
    private final Bank bank = new BankImpl();

    @GET
    @Produces("application/json")
    public Set<String> getAccountNumbers() {
        return bank.getAccountNumbers();
    }
}
```

Client:

```
public class BankClient {
    public static void main(String[] args) throws Exception {
        WebTarget target = ClientBuilder.newClient()
            .target("http://localhost:9999/bank/accounts");

        Set<String> result = null;
        while(true) {
            Response res = target.request("application/json").get();
            Set<String> result = res.readEntity(Set.class);
            System.out.println(result);
            Thread.sleep(1000);
        }
    }
}
```

Der Client lädt dabei jedoch jede Sekunde alle Kontodaten vom Server auf den Klienten. Korrigieren Sie diese Lösung so, dass die Daten nur noch dann übertragen werden, wenn sie sich auf dem Server geändert haben. Dazu soll ein Conditional-GET auf der Basis von E-Tags realisiert werden.

Aufgaben:

- Passen Sie die Ressource entsprechend an. Als E-Tag können Sie den hashCode des Sets bank.getAccountNumbers verwenden. Es genügt, wenn Sie die geänderte Methode getAccountNumbers() angeben.
- Passen Sie entsprechend auch den Bank-Klienten an, d.h. der Client soll jeweils den E-Tag des letzten Standes beim Request mitschicken und das Result-Set result nur dann ändern wenn sich dieses geändert hat.

Bemerkungen:

- Die Schnittstellen der Typen Request und Response sind auf Seiten 6-8 abgedruckt.

Aufgabe 7: Socket Chat**(9 Punkte)**

In dieser Aufgabe soll ein einfacher Socket-basierter Chat implementiert werden. Der Server stellt einen Dienst unter Port 1234 zur Verfügung. Sobald sich ein Client mit diesem Dienst verbindet, so kann er Textzeilen an den Server schicken. Diese Zeilen werden dann vom Server an alle Klienten zurück geschickt (auch an jenen, der die Zeile geschickt hat). Auf diesen Dienst kann z.B. mit telnet zugegriffen werden.

Aufgabe:

Schreiben Sie den Server, der auf Port 1234 diesen Chat-Dienst aufmacht, Zeilen entgegen nimmt und diese laufend alle Klienten weiterleitet.

Bemerkungen:

- Exceptions müssen Sie nicht behandeln.
- Falls der Aufruf von `readLine()` auf der Instanz der Klasse `BufferedReader` als Resultat `null` zurückgibt, dann muss dieser Client nicht mehr berücksichtigt werden (dieser Client hat sich dann abgemeldet und soll aus allfälligen Datenstrukturen entfernt werden und nicht mehr mit Meldungen bedient werden).
- Um aus den `InputStream` und `OutputStream` der Sockets Reader und Writer erzeugen zu können stehen in der Klasse `Server` die Methoden `toReader` und `toWriter` zur Verfügung.

```
public class Server {  
  
    public static void main(String[] args) throws Exception {  
        // ihr code  
    }  
  
    static BufferedReader toReader(InputStream is) {  
        return new BufferedReader(new InputStreamReader(is));  
    }  
  
    static PrintWriter toWriter(OutputStream os) {  
        return new PrintWriter(os, true);  
    }  
}
```

Aufgabe 5

a) public static void main (String[] args) ... {

texam/convert

✓ ↓ -

URL ucl = new URL("http://bank.fhnw.ch:8080/convert");
HttpURLConnection con = (HttpURLConnection) ucl.openConnection(); ✓
con.setRequestMethod("POST"); ✓ "application/x-www-form-urlencoded"
con.setRequestProperty("Content-Type", "text/html"); -
con.setChunkedStreamingMode(5); ✓ ✓
ObjectOutputStream oos = new ObjectOutputStream(con.getOutputStream());
1) oos.writeObject("amount=100?from=CHF,to=EUR"); ✓
ObjectInputStream ois = new ObjectInputStream(con.getInputStream()); ✓
2) System.out.println(ois.readObject()); -

3

es war ein
Java session

set Do output (true) -

- 1) oos.write("from=CHF&to=EUR&amount=100".getBytes());
2) con.setInputStream().transferTo(System.out);

b)

body:
amount=100\r\nfrom=CHF\r\n to=EUR

HTTP

✓

/

Post /exam/convert HTTP/1.1
Host: bank.fhnw.ch:8080
Content-Type: application/x-www-form-urlencoded
Transfer-Encoding: chunked

1a
from=CHF&to=EUR&amount=100
01 Shai,-rde

var this wird

da bin t

oder was

5

Aufgabe 6.

④ Context Request. request

+

a) `Response`

↓

```
public Set<String> getAccountNumbers() {
    EntityTag tag = new EntityTag(bank.getAccountNumbers().getHashCode());
    ResponseBuilder builder = request.evaluatePreconditions(tag);
    if (builder != null) {
        return builder.build();
    }
    builder = Response.ok(bank.getAccountNumbers());
    builder.tag(tag);
    return builder.build();
}
```

5

b)

```
Set<String> result;
EntityTag lasttag; ✓
:
while (true) {
    .header("If-none-Match", lasttag).set()
    Response res = target.request("application/json").get();
    lasttag = res.getEntityTag();
    result = res.readEntity(Set.class);
    if (lasttag == null) {
        if (lasttag != res.getEntityTag()) {
            result = res.readEntity(Set.class);
        }
    }
}
```

wenn der val genau wird dann wird
nur bei ersten val die remove
es sollt.

--- mit
equal ver-
stehen man

Der Request muss etag mitgeben
und da ist Header ---

und by nun ergebn und --

↑ if(res.getStatus() == 200) { ... }

1'2

6'2

Aufgabe 7

```
public static void main(String[] args) {  
    List<PrintWriter> connections = new CopyOnWriteArrayList<>();  
  
    try (ServerSocket server = new ServerSocket(1234)) { ✓  
        while (true) { ✓  
            new Thread(new ClientWorker(server.accept()));  
            } } ↗ PrintWriter wr = toWriter(s.getOutputStream); -  
    }
```

```
private static class ClientWorker implements Runnable {  
    private final Socket client;
```

```
    public ClientWorker(Socket client) {  
        this.client = client;
```

```
    @Override  
    public void run() {  
        try (ObjectInput in = new ObjectInputStream(client.getInputStream());  
             ObjectOutputStream out = new ObjectOutputStream(client.getOutputStream());) {  
            do {
```

initial mit OIS (oo)

```
        } while (!client.isClosed()); ✓ ✓  
    }
```

lrb	---
read	--
read	--
cln	-

3

Aufgabe 7

```
List<PrintWriter> connections = new CopyOnWriteArrayList<>();  
ServerSocket ss = new ServerSocket(1234);  
while (true) {  
    Socket s = ss.accept();  
    PrintWriter wr = new PrintWriter(s.getOutputStream());  
    connections.add(wr);  
    BufferedReader rd = new BufferedReader(s.getInputStream());  
    new Thread(() -> {  
        while (true) {  
            String line = rd.readLine();  
            if (line == null) {  
                connections.remove(wr);  
                wr.close();  
                break;  
            }  
        }  
    }).start();  
}
```