

Enterprise Computing

Kontrollfragen v1.1

**Haltner Manfred & Ferrari Fabio, Abteilung I
HS 07 / 08**

1. GRUNDLAGEN	4
1.1. Qualitätsanforderungen an Unternehmenssoftware	4
1.2. Wartbarkeit	4
1.3. Strategische Anforderungen	4
2. SESSION BEANS	5
2.1. String aus Deployment Deskriptor	5
2.2. Float aus Deployment Deskriptor	5
2.3. EJB2.1 Remote Interface	5
2.4. EJB2.1 Home Interface	6
2.5. EJB2.1 SessionBean	6
2.6. Komplette SessionBean Implementierung	7
2.6.1. Eigenschaften	7
2.7. Deployment Descriptor - <session> Tag	8
2.8. EJB2.1 Lifecycle Methode - ejbPassivate(), ejbPostCreate() usw.	8
2.9. EJB2.1 Passivieren für Stateful Session Beans	8
2.10. Referenz verfügbar machen	9
2.11. Referenz von einer Session Bean zu einer anderen Session Bean	9
2.12. Referenz von einer Session Bean zu einer anderen Entity Bean	9
2.13. EJB2.1 ejbRemove()	10
2.14. SessionBean.ejbRemove Aufruf	10
2.15. Kommunikation eines Clients mit der eigentlichen Bean Instanz	10
2.16. EJB2.1 SessionBean Identität	11
3. ENTITY BEANS	12
3.1. CMP	12
3.1.1. EJB2.1 Container Managed Persistent Beans	12
3.1.2. Zusätzliches um EJB2.1 Entity Bean auf Datenbank abzubilden	12
3.1.3. EJB2.1 Container Managed Transactions.....	13
3.1.4. Erweiterungen in EJB3.0.....	13
3.1.5. Minimale Entity Bean Implementierung.....	14
3.1.6. Mapping zwischen Bean Klasse und Datenbankschema.....	15
3.1.7. Finder Methoden um beliebige Entitäten aus DB zu erfragen.....	15
3.2. BMP	15
3.2.1. EJB2.1/3.0 Bean-managed Transactions BMT.....	15
3.2.2. Bean-managed Transactions BMT	16
3.2.3. BMP vs. CMP.....	17
3.2.4. DB-Zugriff.....	17
3.2.5. SQL Statements Generierung	17
4. MESSAGE DRIVEN BEANS	18
5. TRANSACTION	19
5.1. Fundamentale Eigenschaften von Transaktionen (ACID)	19
5.2. Session Bean Methodenaufruf im transaktionalen Kontext Aufruf verbieten Fehler!	
Textmarke nicht definiert.	
5.3. Datenbankverbindung mittel JPA	19
5.4. ORM/JPA	20
5.5. JTA basierte Verbindung zur Verfügung stellen	20
5.6. Datenbanktabelle auf Klassenhierarchie abbilden	21
5.7. Transaktionsattribute	21
5.8. Sprachelemente von JPA und EJB-QL	22
6. JMS	23
6.1. Die 7 fundamentale Interface der JMS API	23
6.2. Messaging Paradigmen	23
6.3. JMS: Typische API Sequenz Code Fragment	23
7. EJB3	24
7.1. Remote zugreifbare EJB3.0 Session Bean	24
7.2. Einer EJB3.0 Session Bean, Informationen über Ablaufkontext mitgeben	24
7.3. EJB3.0 Annotationen	24

7.3.1. Pendant bei EJB2.1	25
7.4. EJB3 JPA	25
8. SECURITY	26
8.1. EJB2.1 und J2EE Security Rollen	26
9. FREI FORMULIERTE FRAGEN.....	27
10. JOLLER PRÜFUNGEN	31

1. GRUNDLAGEN

1.1. Qualitätsanforderungen an Unternehmenssoftware

Nennen Sie 5 unterschiedliche Qualitätsanforderungen an Unternehmenssoftware und beschreiben Sie jede einzelne mit wenigen prägnanten Sätzen. (5 Punkte, 5 Minuten)

Im Skript gibt es 4 grobe Unterscheidungen, die wie folgt lauten:

- **FUNKTIONALE QUALITÄT (KORREKTHEIT):**

Software muss ihre Aufgabe exakt erfüllen und zwar so wie sie durch Anforderungen und Spezifikationen definiert ist (gilt für jede Software)

- **QUASI-FUNKTIONALE QUALITÄTEN (KONFIGURIERBARKEIT / KOMPATIBILITÄT)**

Kompatibilität ist ein Mass der Leichtigkeit, mit der Softwareprodukte mit anderen verbunden werden können

- **LAUFZEITQUALITÄT (PERFORMANZ / STABILITÄT / VERFÜGBARKEIT)**

Fähigkeit von Softwaresystemen, auch unter aussergewöhnlichen Bedingungen zu funktionieren.

- **ENTWICKLUNGSQUALITÄTEN (ÄNDERBARKEIT, WARTBARKEIT)**

Änderbarkeit: Leichtigkeit, mit der Software geändert werden kann.

Wartbarkeit: Änderbarkeit aufgrund von Fehlern

Alternativ Lösung!

- **AVAILABILITY**

Die Applikation sollte, wann immer sie benutzt werden möchte, verfügbar sein

Möglichst wenig Ausfallzeiten

- **STABILITY**

Die Applikation sollte stabil, ohne Abstürze laufen. D.h. Fehler müssen abgefangen und darauf reagiert werden

- **CLARITY**

Die Software sollte intuitiv und möglichst selbsterklärend sein

- **COMPATIBILITY**

Die Software sollte mit anderen Softwareprodukten kompatibel sein (wo es verlangt ist)

- **CONFIGURABILITY**

Die Software sollte nach den eigenen Bedürfnissen und Geschmäcker konfigurierbar sein

1.2. Wartbarkeit

Mit „Wartbarkeit“ (Maintainability) beschreibt man eine wichtige Kategorie von Qualitätsmerkmalen von Software Applikationen. Nennen Sie 5 unterschiedliche Aspekte der Wartbarkeit und beschreiben Sie jede einzelne mit jeweils maximal drei prägnanten Sätzen. (5 Punkte, 5 Minuten)

Der Student hat bei der Prüfung obige Punkte aufgezählt und fast die volle Punktzahl erhalten.

1.3. Strategische Anforderungen

Nennen Sie 5 unterschiedliche strategische Anforderungen an Unternehmenssoftware und beschreiben Sie jede einzelne mit wenigen prägnanten Sätzen. (5 Punkte, 5 Minuten)

- **LANGLEBIGKEIT DER TECHNOLOGIE (KRITISCHE MASSE, STANDARDS)**

- **VERFÜGBARKEIT DER RESSOURCEN (DOKUMENTATION, PERSONAL, SCHULUNG, AUCH**

- **WISSENSCHAFTLICHE FUNDAMENTE "2PHASEN-COMMIT", "2ND LEVEL-CACHE")**

- **VERFÜGBARKEIT VON TOOLS/KOMPONENTEN/Frameworks**

- **INTEGRATIONSPOTENTIAL (SOA)**

2. SESSION BEANS

2.1. String aus Deployment Deskriptor

Wie ermöglichen Sie einer EJB2.1 Session Bean einen speziellen String APPLICATION_NAME aus einem Deployment Deskriptor in Erfahrung zu bringen? Bitte Java Code und XML Deskriptor exakt mit Filenamen angeben. Kein Freitext! (3.5 Punkte, 6 Minuten)

1) XML Deskriptor Fragment „ejb-jar.xml“

```
<session>
...
<env-entry>
  <env-entry-name>APPLICATION_NAME</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>meineApplikation</env-entry-value>
...
</env-entry>
...
</session>
```

2) Per JNDI lookup herausfinden

```
Context initial = new InitialContext();
Context environment = (Context)initial.lookup("java:comp/env");
String applName = (String)environment.lookup("APPLICATION_NAME");
```

2.2. Float aus Deployment Deskriptor

EJB2.1: Eine SessionBean soll Files bis zu einer bestimmten Grösse einlesen und verarbeiten (1 000 000 byte). Diese maximale Grösse soll im Deployment Deskriptor definiert werden. Beschreiben Sie eine mögliche Lösung.

- 1) Java Code Fragment
 - 2) XML Deskriptor Fragment.
- (3 Punkte, 5 Minuten)

1) Per JNDI lookup herausfinden

```
InitialContext ctx = new InitialContext();
Long max = (Long) ctx.lookup("java:comp/env/ejb/MAX_BYTES");
```

2) XML Deskriptor Fragment

```
<env-entry>
  <env-entry-name>MAX_BYTES</env-entry-name>
  <env-entry-type>java.lang.Long</env-entry-type>
  <env-entry-value>1000000</env-entry-value>
</env-entry>
```

2.3. EJB2.1 Remote Interface

EJB2.1: Welche Aufgabe hat das EJB Remote Interface für Session Beans und welche speziellen Java-Eigenschaften (implements/extends) hat dieses Interface, bzw. dessen Methoden? (1.5 Punkte, 2 Minuten)

Über das Remote-Interface kommuniziert der Client mit der entfernten Enterprise JavaBean, daher leitet sich auch der Name Remote her. Dieses Interface deklariert dabei alle Methoden, die der Client auf dem Objekt aufrufen kann. Beispielsweise eine Methode, die einen kurzen Text zurückliefert:

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface MySessionBean extends EJBObject {
    public String getText() throws RemoteException;
}
```

2.4. EJB2.1 Home Interface

EJB2.1: Welche Aufgabe hat das EJB Home Interface für Session Beans und welche speziellen Java-Eigenschaften (implements/extends) hat dieses Interface, welche Methode muss es mindestens definieren (vollständige Signatur) (2 Punkte, 2 Minuten)

Dieses Interface dient dazu, die Bean zu erzeugen. Hierbei fordert der Client beim Server ein gewünschtes Objekt an. Der Server erzeugt daraufhin die EJB und stellt sie dem Client zur Verfügung. Da der EJB-Container dabei auf dem gleichen Server wie die erzeugte EJB läuft, wird dieses Interface als Home-Interface bezeichnet.

```
import java.rmi.RemoteException;
import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;

public interface MySessionBeanHome extends EJBHome {
    public MySessionBean create() throws CreateException, RemoteException;
}
```

2.5. EJB2.1 SessionBean

EJB2.1: Welche Java Eigenschaften (implements/extends) muss die EJB Implementierungsklasse einer SessionBean haben, und welche Methoden muss sie mindestens implementieren? (3.5 Punkte, 5 Minuten)

Die EJB-Implementierung bildet den Kern der EJB. Sie nimmt die gewünschte Geschäftslogik auf und enthält die Methoden des Remote- und des Home-Interfaces, die erforderliche interne Repräsentation der Daten, sowie Methoden, über die der EJB-Container die Beans verwaltet.

```
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.CreateException;

public class MySessionBeanImpl implements SessionBean {

    //Methoden des Remote-Interface (Geschäftslogik)
    public String getText(){
        return „HelloWorld“;
    }

    //Methoden des Home-Interface
    public void ejbCreate() throws CreateException {...}

    //Methoden des Interface SessionBean, dienen der Verwaltung durch den Container
    public void ejbActive() { ... }
    public void ejbPassivate() { ... }
    public void ejbRemove() { ... }
    /* Ermöglicht Kommunikation mit EJB-Container */
    public void setSessionContext(SessionContext aContext) { ... }
```

2.6. Komplette SessionBean Implementierung

Schreiben Sie eine komplette EJB 2.1 Session Bean Implementierung namens „StockQuoteService“ (Java und Deskriptor), die für entfernte Zugriffe die Methode

```
double getStockQuote ( StockSymbol symbol )
```

zur Verfügung stellt. Lassen Sie alle Methodenimplementierungen leer.
(10 Punkte, 10 Minuten)

```
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.CreateException;

public class StockQuoteService implements SessionBean{
    private SessionContext ctx;

    // Methode des Remote-Interface
    public double getStockQuote( StockSymbol symbol ){
        ...
        return value;
    }

    // Methode des Home-Interface
    public void ejbCreate() throws CreateException{
        value = 0;
    }
    public void ejbCreate( double value ){
        this.value = value;
    }

    // Methoden des Interface SessionBean
    public void ejbActivate() { ... }
    public void ejbPassivate() { ... }
    public void ejbRemove() { ... }
    public void setSessionContext { ... }
}

//ejb-jar.xml
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>StockQuoteService</ejb-name>
      <home>StockQuoteServiceHome</home>
      <remote>StockQuoteServiceRemote</remote>
      <ejb-class>[namespace]StockQuoteService</ejb-class>
      <session-type>Stateless</session-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

2.6.1. Eigenschaften

Welche Eigenschaften muss die Klasse StockSymbol haben?
(0.5 Punkte, 30 Sekunden)

Sie muss serialisierbar sein.

2.7. Deployment Deskriptor - <session> Tag

EJB2.1: Woraus muss ein <session> Tag innerhalb des Deployment Deskriptors einer Stateless Session Bean mindestens bestehen, wenn Zugriffe von Remote und Lokal vorgesehen ist? (4 Punkte, 6 Minuten)

```
<session>
  <ejb-name>HelloWorld</ejb-name>

  <home>ch.hsr.ita.ejb.HelloWorldHome</home>
  <remote>ch.hsr.ita.ejb.HelloWorld</remote>

  <local>ch.hsr.ita.ejb.HelloWorldLocal</local>
  <local-home> ch.hsr.ita.ejb.HelloWorldLocalHome</local-home>

  <ejb-class>ch.hsr.ita.ejb.HelloWorldImpl</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Container</transaction-type>
</session>
```

2.8. EJB2.1 Lifecycle Methode - ejbPassivate(), ejbPostCreate() usw.

Erläutern Sie die Anwendung und Bedeutung der EJB2.1 Lifecycle Methode ejbPassivate() für Stateful Session Beans. (1.5 Punkte, 3 Minuten)

▪ EJBPASSIVATE()

Wenn zu viele Beans instanziiert werden, schreibt der Container die Beans auf einen temporären Speicher (persistierung). Zuvor wird die Methode ejbPassivate() aufgerufen, in welcher Sockets & Ressourcen freigegeben werden sollten.

▪ EJBACTIVATE()

Alle benötigten Ressourcen sollten hier wieder beschafft werden (Sockets Connections...)

▪ EJBREMOVE()

Wird aufgerufen vom EJB-Container, nachdem der Client die remove-Methode aufruft & Instanzen für GC freigibt.

▪ EJBCREATE()

Wenn ein Client eine create()-Methode des EJBHome- bzw. EJBLocalHome-Objektes aufruft, wird die äquivalente ejbCreate()-Methode einer Instanz der Bean-Klasse aufgerufen.

▪ EJBPOSTCREATE()

Nach dem erzeugen der Instanz und dem setzen des SessionContext ruft der Container die Methode auf. → keine Parameter, wird vom Container aufgerufen.

2.9. EJB2.1 Passivieren für Stateful Session Beans

EJB2.1: Erläutern Sie den Vorgang des Passivierens für Stateful Session Beans: Warum braucht der Container einen solchen Mechanismus, und wie kann ein Entwickler seine Implementierung auf ein solches Ereignis vorbereiten? (4 Punkte, 6 Minuten)

Passivierung tritt ein, wenn der Container eine neue Instanz braucht, aber die Größe des Pools damit überschritten würde. Dann wird die am längsten nicht mehr gebrauchte Instanz des Pools in persistenten Speicher geschrieben.

Dazu ruft der Container ggf. die Lifecycle-Callback Methoden (z.B. ejbPassivate(), ejbActivate()) auf, um die EJB von seinen kommenden Aktivitäten zu unterrichten. Ferner müssen die Beans serialisierbar sein.

2.10. Referenz verfügbar machen

EJB2.1: Was genau müssen Sie wo deklarieren, um innerhalb einer SessionBean eine Referenz auf eine andere im Komponentenspezifischen Namensraum unter „java:comp/env“ verfügbar zu machen, so dass sie mittels `InitialContext.lookup(...)` gefunden wird? (7 Punkte, 5 Minuten)

In `ejb-jar.xml` bei der Session Bean:

```
<session>
  <ejb-name>ThisBean</ejb-name>
  <ejb-local-ref id="whatever">
    <ejb-ref-name>ejb/OtherBean</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <localhome>OtherBeanLocalHome</localhome>
    <local>OtherBeanHome</local>
    <ejb-link>Other</ejb-link>
  </ejb-local-ref>
</session>
```

2.11. Referenz von einer Session Bean zu einer anderen Session Bean

Wie gelangen Sie von einer SessionBean Instanz an eine Referenz auf eine andere lokale SessionBean „UserMgmt“ in EJB2.1? Java Code für JNDI API und XML Deskriptor Fragmente, kein Freitext. Verwenden Sie selbstkommentierende Klassennamen mit Suffixen („Local“ oder „Home“) zur Erläuterung. (5. Punkte, 10 Minuten)

```
// MyServletBean.java:
try{
  InitialContext ictx = new InitialContext();
  Object thing = ictx.lookup("UserMgmt");
  UserMgmtLocalHome umlh = (UserManagementLocalHome )PortableObject.narrow(
    Thing, UserMgmtLocalHome.class);
  userMgmt = umlh.create();
  userMgmt.doSth();
}catch(NamingException e){
  Throw new RuntimeException(e);
}catch(FinderException e){
  Throw new RuntimeException(e);
}

// JBoss.xml:
<session>
  <ejb-name>UserMgmt</..
  <jndi-name>userMgmt</..
  <local-jndi-name>UserMgmtLocal</..
</session>

// ejb-jar.xml
<session id="Session_UserMgmt">
  <display-name>UserMgmt</..
  <ejb-name>UserMgmt</..
  <home>UserMgmtHome</..
  <local-home>UserMgmtLocalHome</..
  <ejb-class>UserMgmt</..
```

2.12. Referenz von einer Session Bean zu einer anderen Entity Bean

Wie gibt eine Session Bean eine Referenz auf sich selbst an eine andere Entity Bean korrekt weiter? (1 Punkt, 2 Minuten)

Über den `SessionContext`

`sessionContext.getEJBObject()` in der Regel mit `Typecast` auf dem Typ des Objekts

2.13.EJB2.1 ejbRemove()

EJB2.1: Was müssen Sie in Ihrem EJB Client Code tun, damit die Methode `ejbRemove` Ihrer Bean Implementation aufgerufen wird? (1 Punkt, 1 Minute)

Grundsätzlich wird `ejbRemove` vom Applicationcontainer aufgerufen, wenn das zugehörige Sessionobjekt zerstört (d. h. der Garbage Collection übergeben) wird. Dies ist immer dann der Fall, wenn entweder ein Client explizit die `remove`-Methode aufgerufen hat oder die Haltbarkeitszeit der Session abgelaufen ist. Das EJB muss also vom EJP Container aufgerufen werden können.

2.14.SessionBean.ejbRemove Aufruf

Wann rufen Sie in Ihrem EJB Client Code die Methode `SessionBean.ejbRemove()` auf? (1 Punkt, 1 Minute)

Garnicht! Der Container verwendet diese Methode, wenn er die `SessionBean`-Instanz nicht mehr benötigt.

2.15.Kommunikation eines Clients mit der eigentlichen Bean Instanz

EJB2.1: Erläutern Sie in einigen prägnanten Sätzen die Schritte, mit denen die Kommunikation eines Clients mit der eigentlichen, von Ihnen implementierten Bean Instanz stattfinden. Beginnen Sie mit dem JNDI Lookup. (9 Punkte, 8 Minuten)

Ich schreibe den Code nur auf, um ein Beispiel für das Testen einer EJB zu haben.

```
// Konfiguration des Service Providers (JBoss)
Hashtable env = new Hashtable();

// Setzen der JBoss JNDI-Factory
env.put(Context.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");

// URL des Namensdienstes (JBoss-Standard-Port: 1099)
env.put(Context.PROVIDER_URL, "localhost:1099");

// Konfiguration der URL Package Präfixe
env.put(Context.URL_PKG_PREFIXES, "org.jboss.naming:org.jnp.interfaces");

// Erzeugen des Initialen Kontextes
InitialContext ctx = new InitialContext(env);

// Ermitteln der RMI-Referenz über den JNDI-Name der EJB
Object objRef = ctx.lookup("ejb/HelloWorld");

// Umwandeln der RMI-Referenz in ein Objekt (Cast)
HelloWorldHome helloWorldHome = (HelloWorldHome) PortableRemoteObject.
narrow(objRef, HelloWorldHome.class);

// Freiegeben der JNDI-Ressourcen
ctx.close();

// Erzeugen des EJB-Objekts über das Home-Interface
HelloWorld bean = helloWorldHome.create("Fabio");
System.out.println(bean.getGreeting());
```

Zunächst muss der Zugang zum Namensdienst des JBoss-Servers konfiguriert werden. Dies geschieht mittels dessen `NamingContextFactory` und setzen der URL, unter der dieser erreichbar ist. Um die vom JBoss-Namensdienst verwendeten Package-Präfixe auflösen zu können, muss auch die Konstante `URL_PKG_PREFIXES` gesetzt werden. Das Standardwert `com.sun.jndi.url` bleibt dabei ebenfalls erhalten und wird vom Server automatisch hinten angefügt.

Nachdem der Namensdienst konfiguriert und der `InitialContext` erstellt ist, muss eine Referenz auf das gewünschte Objekt extrahiert werden (**lookup**). Dazu wird der im JBoss Deployment Descriptor definierte JNDI-Namen verwendet. Da die EJB-Objekte jedoch via RMI und damit über das RMI-IIOP-Protokoll übertragen werden, wobei sie zunächst serialisiert und anschliessend wiederhergestellt werden, kann das zurückgegebene Objekt nicht direkt in das Home-Interface umgewandelt werden. Stattdessen muss die RMI-Cast Funktion **narrow()** verwendet werden. Danach hat man Zugriff auf das Home-Interface der EJB, über dessen `create()`-Methode die Bean erzeugt werden kann. Nachdem die `create()`-Methode aufgerufen wurde, überprüft der Server, ob bereits eine entsprechende Bean existiert. Siehe Foliensatz 3.37!

2.16.EJB2.1 SessionBean Identität

EJB2.1: Wie bringen Sie in einer SessionBean die Identität des eingelogten Users in Erfahrung?
(0.5 Punkte, 1 Minute)

→ ctx wird vom Container gesetzt `setSessionContext()`, `setEntityContext()`

```
Principal principal = ctx.getCaller.Principal();  
String identity = principal.getName();
```

3. ENTITY BEANS

3.1. CMP

3.1.1. EJB2.1 Container Managed Persistent Beans

EJB2.1 CMP: Schreiben Sie eine minimale Entity Bean Implementierung, die die Datenbanktabelle CUSTOMERTYPE mit den Spalten

*ID CHAR(255) und
DESCRIPTION CHAR(255)*

abbildet. Implementieren Sie nur den lokalen Zugriff. Geben Sie die minimale Java Klassenimplementierung und die nötigen Interfaces an (Kompletter Java Code!). Hinweis NUR CODE, kein Freitext.
(7.5 Punkte, 12 Minuten)

Ich setzte zuerst eine Vaterklasse auf, die für alle EJB-Klassen dienen, die als CMP aufgesetzt werden.

```
public class CMPMasterBean implements EntityBean {
    protected EntityContext entityContext;
    protected EJBEnv env;
    public CMPMasterBean() {
        // Initalisieren der EJBEnv
        env = EJBEnv.getInstance();
    }
    public void setEntityContext(EntityContext ec) throws
        EJBException, RemoteException {
        this.entityContext = ec;
    }
    public void unsetEntityContext() throws EJBException, RemoteException {
        this.entityContext = null;
    }
    public void ejbRemove() throws RemoveException, EJBException, RemoteException {}
    public void ejbActivate() throws EJBException, RemoteException {}
    public void ejbPassivate() throws EJBException, RemoteException {}
    public void ejbLoad() throws EJBException, RemoteException {}
    public void ejbStore() throws EJBException, RemoteException {}
}
public interface CMPMasterLocal extends javax.ejb.EJBLocalObject {
    //Hier könnten Methoden stehen, die für alle CMPs angeboten werden müssen
}
public interface CMPMasterLocalHome extends javax.ejb.EJBLocalHome {
    //Hier könnten Methoden stehen, die für alle CMPs angeboten werden müssen
}
public abstract class CustomerTypeBean extends CMPMasterBean {
    public Character ejbCreate(Character id) throws CreateException {
        this.setID(id);
        return null;
    }
    public void ejbPostCreate(Character id) {}
    // GETTER
    public abstract Character getID();
    public abstract Character getDescription();

    // SETTER
    public abstract void setID(Characater id);
    public abstract void setDescription(Character description);
}
public interface CustomerTypeLocal extends CMPMasterLocal {
    public Character getId();
    public Character getDescription();
    public void setId();
    public void setDescription();
}
public interface CustomerTypeLocalHome extends CMPMasterLocalHome {
    public CustomerTypeLocal create(Character id) throws CreateException;
    public CustomerTypeLocal findByPrimaryKey(Character pk) throws FinderException;
}
```

3.1.2. Zusätzliches um EJB2.1 Entity Bean auf Datenbank abzubilden

Welche beiden Artefakte (Files) braucht es zusätzlich, um eine EJB2.1 Entity Bean auf eine Datenbank abzubilden? In kurzen Worten: Was steht in diesen Files beschrieben? (2.5 Punkte, 3 Minuten)

Es braucht noch die beiden Files `ejb-jar.xml` und `jboss.xml`. Genau wie bei einer lokalen Session Bean wird der Zusammenhang zwischen den Interfaces und der EJB-Klasse spezifiziert. (z.b. `ejb-name`, `local-home`, `local`, `ejb-class`, `persistence-type`, `prim-key-class`, `reentrant` und `cmp-field`. Im `jboss.xml` wird innerhalb des `entity-Element`s ein lokaler JNDI-Name für die CMP „CustomerType“ vergeben.

3.1.3. EJB2.1 Container Managed Transactions

EJB2.1 Container Managed Transactions: Was genau muss geschehen, damit ein EJB-Container nach einer transaktionalen Methode einer SessionBean einen Rollback durchführt?
(2 Punkte, 3 Minuten)

Es muss eine `RuntimeException` geworfen werden.
Es muss die Methode `setRollbackOnly` aufgerufen worden sein.

3.1.4. Erweiterungen in EJB3.0

Welche erweiterten Möglichkeiten bietet EJB3.0 (bezogen auf die obige Frage)
(2.5 Punkte, 4 Minuten)

Entweder über Annotations, oder über Deployment Descriptor.

▪ ANNOTATIONS

Eigene Exceptionklassen können markiert werden ob beim werfen einer Exception ein Rollback durchgeführt werden soll.

```
@ApplicationException (rollback=true)
Public class MyException extends Exception{
}
```

▪ ÜBER EJB-JAR.XML - FILE

```
<ejb-jar>
...
  <assembly-descriptor>
    <application-exception>
      <exception-class>
        Java.sql.SQLException
      </exception-class>
      <rollback>true</rollback>
    </application-exception>
  </assembly-descriptor>
</ejb-jar>
```

3.1.5. Minimale Entity Bean Implementierung

Schreiben Sie eine minimale Entity Bean Implementierung, die die Datenbanktabelle CURRENCY mit den Spalten ID (BigInteger) und CURRENCY_CODE VARCHAR(3) abbildet. Implementieren Sie nur den lokalen Zugriff. Geben Sie die minimale Java Klassenimplementierung und die nötigen Interfaces an. (Kompletter Java Code!). Hinweis: NUR CODE, kein Freitext!
(7.5 Punkte, 12 Minuten)

```
// ejb-jar.xml:
<?xml version="1.0" encoding="UTF-8"?>
<display-name>MyEntityProject</display-name>
<enterprise-beans>
  <entity>
    <ejb-name>Currency</ejb-name>
    <local-home>MyEntityLocalHome</local-home>
    <local>MyEntityLocal</local>
    <ejb-class>MyEntityBean</ejb-class>
    <persistence-type>Container</persistence-type>
    <prim-key-class>java.lang.Integer</prim-key-class>
    <reentrant>true</reentrant>
    <cmp-version>2.x</cmp-version>
    <abstract-schema-name>CURRENCY</abstract-schema-name>
    <cmp-field>
      <field-name>ID</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>CURRENCY_CODE</field-name>
    </cmp-field>
    <primkey-field>ID</primkey-field>
  </entity>
</enterprise-beans>
</ejb-jar>

// MyEntityLocal.java
import javax.ejb.EJBLocalObject;

public interface MyEntityLocal extends EJBLocalObject {
  public Integer getID();
  public void setID(Integer id);

  public String getCURRENCY_CODE();
  public void setCURRENCY_CODE(String code);
}

// MyEntityLocalHome.java
import javax.ejb.CreateException;
import javax.ejb.EJBLocalHome;
import javax.ejb.FinderException;

public interface MyEntityLocalHome extends EJBLocalHome {
  MyEntityLocal create(Integer id, String currency) throws CreateException;
  // mind. eine Finder-Methode muss vorhanden sein!!!
  MyEntityLocal findByPrimaryKey(Integer id) throws FinderException;
}

// MyEntityBean.java
import java.rmi.RemoteException;

import javax.ejb.CreateException;
import javax.ejb.EJBException;
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import javax.ejb.RemoveException;

public abstract class MyEntityBean implements EntityBean {

  public abstract Integer getID();
  public abstract void setID(Integer id);

  public abstract String getCURRENCY_CODE();
  public abstract void setCURRENCY_CODE(String code);

  public MyEntityBean() {}

  public Integer ejbCreate(Integer id, String code) throws CreateException {
    setID(id);
    setCURRENCY_CODE(code);
  }
}
```

```

        return id;
    }

    public void ejbPostCreate(Integer id, String code) throws CreateException {}
    public void ejbActivate() throws EJBException, RemoteException {}
    public void ejbLoad() throws EJBException, RemoteException {}
    public void ejbPassivate() throws EJBException, RemoteException {}
    public void ejbRemove() throws RemoveException, EJBException, RemoteException {}
    public void ejbStore() throws EJBException, RemoteException {}
    public void setEntityContext(EntityContext arg0) throws
    EJBException, RemoteException {}
    public void unsetEntityContext() throws EJBException, RemoteException {}
}

```

3.1.6. Mapping zwischen Bean Klasse und Datenbankschema

Wo ist das eigentliche Mapping zwischen Bean Klasse und Datenbankschema definiert (bezogen auf die obige Frage)?
(0.5 Punkte, 1 Minute)

Im `cmp-field` – Tag des `ejb-jar.xml` wird festgelegt, wie die Spalten in der Datenbank heissen. In der `implementations`-Klasse (`MyEntityBean.java`) wird das Mapping anhand der `getter`- und `setter`-Methoden vervollständigt (Datentypen). Der Rest wird vom Container übernommen.

3.1.7. Finder Methoden um beliebige Entitäten aus DB zu erfragen

Beschreiben Sie detailliert die sprachlichen Mittel (EJB2.1 `EntityBean` API und `EJB-QL`), mit denen es Ihnen möglich ist, praktisch beliebige Entitäten aus der Datenbank zu erfragen: Hinweis: Finder Methoden. Geben Sie Java Code Sequenzen und XML Fragmente anhand eines einfachen Beispiels ihrer Wahl an.
(5 Punkte, 8 Minuten)

- `findByPrimaryKey()`
- die Finder-Methode `findByPrimaryKey(PkClass key)`; muss immer im `Home` Interface beschrieben sein
- Signatur: `findXXX(...)` im `home`-Interface deklariert. Im `ejb-jar.xml` werden die `EJB-Queries` zu dieser Methode definiert. Der Container implementiert anhand dieser Daten die `finder`-Methode.

```

//ejb-jar.xml:
<query>
  <query-method>
    <method-name>findByName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
</ejb-ql>
<![CDATA[SELEC OBJECT(a) FROM LegalPerson21 as a where a.name=?1]]>
</ejb-ql>
</query>

```

```

Public interface LegalPersonLocalHome extends EJBLocalHome{
    Collection findByName(String name) throws FinderException;
    LegalPersonLocal findByPrimaryKey(Long pk) throws FinderException;
}

```

3.2. BMP

3.2.1. EJB2.1/3.0 Bean-managed Transactions BMT

EJB2.1/3.0: Schreiben und kommentieren Sie eine EJB Methode, die mit den Mitteln von BMT irgendeine andere Methode „doSave()“ so aufruft, wie es das Transaktionsattribute „Required“ beschreibt. (Berücksichtigen sie nur aktive (Status.Active) Transaktionen.
(5 Punkte, 8 Minuten)

```

/**
 * @ejb.interface-method
 */
public void blabla() throws EJBException {
    // Transaktion beende?
    if(!transaktionBeendet){
        throw new EJBException(„Transaktion noch nicht abgeschlossen.“);
    }
}

```

```

    } else {
        try{
            //Beginnen mit der Transaktion
            UserTransaction ut = sessionContext.getUserTransaction();
            ut.begin();
            //Festlegen der Haltbarkeit der Transaktion in Sekunden
            ut.setTimeout(50);

            //Den Datensatz anlegen
            dbAccessor.save();
            //Flat setzen
            transaktionBeendet = false;
        } catch(Exception e){
            //Transaktion abbrechen, Fehler melden
            UserTransaction ut = sessionContext.getUserTransaction();
            ut.rollback();
            error = true;
            throw new EJBException (e.toString());
        }
    }
}

```

3.2.2. Bean-managed Transactions BMT

EJB3.0: Welche programmiersprachlichen Mittel benötigen Sie, um Transaktionen selbst zu managen (Bean-managed Transactions BMT)? (3 Punkte, 4 Minuten)

- **NUTZUNG DES USERTRANSACTION**

Interface auf welches entweder über einen JNDI lookup, oder über den session-Context zugegriffen wird

- **NATIVE SQL**

```

@stateless
@TransactionManagement(TransactionManagerType.BEAN)
public class HypotheticalBean implements HypotheticalLocal{
}

```

```

UserTransaction tr = (UserTransaction) ctx.lookup („java:comp/env/userTransaction“);
..tr.begin()
..tr.commit()
..tr.rollback()

```

3.2.3. BMP vs. CMP

Was unterscheidet eine Bean-managed persistente (BMP) Enterprise Bean von einer Containermanaged persistenten (CMP) Bean?

(eine gültige Antwort)

1. eine BMP Bean muss die `ejbLoad()` und `ejbCreate()` Methoden implementieren
2. eine BMP Bean kann die Persistenz mithilfe eines beliebigen Datastores, beispielsweise eines legacy Systems, implementieren.
3. eine BMP Bean muss ihre Persistenz selber (mithilfe eines Datastores) implementieren
4. alle obigen

4)

3.2.4. DB-Zugriff

Mit welcher Technik wird in BMP's auf die Datenbank zugegriffen?

JDBC

3.2.5. SQL Statements Generierung

Im Falle der Bean Managed Persistence (BMP) werden die SQL-Statements durch (eine gültige Antwort)

1. den Container
2. die EJB Bean-Klasse
3. den EJB Server
4. JDBC

generiert / zur Verfügung gestellt.

2)

4. MESSAGE DRIVEN BEANS

Welche der folgenden Aussagen treffen für asynchrone Architekturen zu?
(mehrere gültige Antworten)

1. bessere Ausnutzung der Kommunikations-Bandbreite
2. gestattet eine bessere Ausnutzung der verarbeitenden Komponenten
3. Antworten an den Absender erfolgen sofort
4. Skalierbarkeit

1), 2) und 4)

Welche der folgenden Aussagen trifft auf die Publish / Subscribe Architektur (Pattern) zu?

(Mehrere gültige Antworten)

1. basiert auf dem Einsatz von URLs, über die wird Publisher gefunden
2. adressiert Subjekt-basiert
3. verwendet ortsunabhängige Publisher
4. kommuniziert synchron zwischen Publisher und Subscriber

2) und 3)

Welche Methode wird bei Eintreffen einer Message in einem Message-Bean aufgerufen?

onMessage(Message)

Nennen Sie den Hauptunterschied zwischen einem Topic und einer Queue.

Aus einem Topic können mehrere Subscriber Messages empfangen. Bei einer Queue gibt es nur einen Empfänger.

5. TRANSACTION

5.1. Fundamentale Eigenschaften von Transaktionen (ACID)

Bennen Sie die vier fundamentalen Eigenschaften von Transaktionen (ACID) und beschreiben Sie jede einzelne mit wenigen prägnanten Sätzen. (6 Punkte, 10 Minuten)

- **ATOMIZITÄT**

Eine Transaktion wird entweder ganz oder gar nicht ausgeführt. Transaktionen sind also unteilbar.

- **CONSISTENZ**

Nach Ausführung der Transaktion ist der Datenbestand nach wie vor in einer widerspruchsfreien Form.

- **ISOLATION**

Durch das Prinzip der Isolation wird verhindert, dass sich in Ausführung befindliche Transaktionen gegenseitig beeinflussen: Realisiert wird dies beispielsweise durch spezielle Sperrprotokolle oder Zeitstempelverfahren.

- **DAUERHAFTIGKEIT**

Die Wirkung einer erfolgreich abgeschlossenen Transaktion bleibt dauerhaft in der Datenbank erhalten, insbesondere auch nach Systemabstürzen.

5.2. Session Bean Methodenaufruf im transaktionalen Kontext Aufruf verbieten

Welche drei Möglichkeiten haben Sie um zu deklarieren, dass Ihre Session Bean Methode auf keinen Fall innerhalb einem existierenden transaktionalen Kontext aufgerufen werden soll? Erläutern Sie jede einzelne Option (3 Punkte, 6 Minuten).

- **REQUIREDNEW**

Die Methode erfordert eine aktive Transaktion, Falls keine mit dem Thread assoziiert ist, wird eine neue begonnen und bei Verlassen der Methode wieder beendet. Wenn eine existiert, dann nimmt die Methode automatisch an dieser Teil und beendet sie nicht.

- **NEVER**

Wenn eine Transaktion mit dem Thread ssoziiert ist, dann wird eine Ausnahme ausgelöst, Wenn nicht, wird die Methode ohne Transaktionskontext aufgerufen.

- **NOTSUPPORTED**

Wenn eine Transaktion mit dem Thread assoziiert ist, dann wird diese suspendiert und nach Verlassen wieder aufgenommen.

5.3. Datenbankverbindung mittel JPA

Beschreiben Sie, wie mit Hilfe von JPA innerhalb einer Serverumgebung eine Verbindung auf eine spezielle Datenbank für eine SessionBean zur Verfügung gestellt wird. (3 Punkte, 5 Minuten)

Ein Persistenzkontext ist eine Menge von Entitätsklassen (Persistence Unit) assoziiert mit einer Datenquelle (DataSource)

```
<persistence>
  <persistence-unit name="greenbill">
    <jta-data-source>java:/MySQLXaGreenBillDs</jta-data-source>
    <jar-file>../domain.jar</jar-file> <!-- optional -->
    <class>com.greenbill.dmn.Money</class> <!-- optional -->
    <exclude-unlisted-classes/> <!-- optional -->
    <properties> <!-- optional -->
      <property name="...">...</property>
    </properties>
  </persistence-unit>
</persistence>
```

5.4. ORM/JPA

ORM/JPA: Stellen Sie eine beidseitig navigierbare vielfach-zu-vielfach (m-to-n) Beziehung zwischen zwei Klassen „Customer“ und „Account“ mittels JPA Annotationen auf den Klassen dar. Beide Klassen besitzen genau jeweils die Attribute „name“ und „id“. Verwenden Sie dazu auf der Datenbank eine eigene Linktabelle.

Weitere Vorgaben:

- Verwenden Sie Listen mit Java5 Generics wo angebracht
- Verwenden Sie Attribute-Annotationen (keine getter/setter)
- Auch wo JPA Defaultwerte annimmt, sollen explizite Angabane gemacht werden (Spaltennamen, etc)
- Die Primärschlüssel beider Tabellen sollen automatisch generiert werden (gemäß default der verwendeten Datenbank)
- Wenn eine „User“ Instanz auf die Datenbank geschrieben wird, sollen automatisch auch die Rollen mit abgespeichert werden.
- Beim Lesen von „Customer“ Instanzen sollen automatisch die dazugehörigen „Account“ Instanzen mitgelesen werden. (13 Punkte, 15 Minuten)

```
@Entity
@Table(name = „customer“)
public class Customer implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = „id“, nullable = false, unique = true)
    private int id;
    @Column(name = „name“, nullable = false)
    private String name;
    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = „customer_accounts“,
        joinColumns = {@JoinColumn(name = „customer“)},
        inverseJoinColumns = {@JoinColumn(name = „account“)})
    private List<Account> accounts;
}
```

```
@Entity
@Table(name = „account“)
public class Account implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = „id“, nullable = false, unique = true)
    private int id;
    @Column(name = „name“, nullable = false)
    private String name;
}
```

5.5. JTA basierte Verbindung zur Verfügung stellen

Wie stellt ein JavaEE Application Server einer SessionBean innerhalb der JPA Spezifikation eine JTA-basierte Verbindung auf eine spezielle Datenbank zur Verfügung (6 Punkte, 5 Minuten)

Der Transaktionstyp bestimmt, ob die Persistence Unit durch das Transaktionsmanagement eines Applikationsservers verwaltet werden soll, der Wert des Attributs wird dann auf JTA gestellt und ist die Standardannahme in einer JavaEE-Anwendung, oder ob die Anwendung das Transaktionsmanagement selbst übernimmt. Letzteres wird durch den Attributwert RESOURCE_LOCAL ausgedrückt und ist in der JavaSE-Welt Standard.

```
<persistence>
  <persistence-unit name=„greenbill“ transaction-type=„JTA“>
    <jta-data-source>java:/MySqlXaGreenBillDs</jta-data-source>
    ...
  </persistence-unit>
</persistence>
```

5.6. Datenbanktabelle auf Klassenhierarchie abbilden

Bilden Sie die im folgenden beschriebene Datenbanktabelle auf eine Klassenhierarchie mit den 3 Klassen „Fahrzeug“, „PersonenKraftWagen“, und „LastKraftWagen“ ab. Verwenden Sie dazu nur die unbedingt nötigen Annotationen aus der Java Persistence API. Reiner Java Code. Kein Freitext. Hinweis: Verwenden Sie keine Setter und Getter Methoden.

Tabelle: FAHRZEUGE

Spalten:

ID	BIGINTEGER
KATEGORIE	VARCHAR(32)
BAUJAHR	integer
NUTZLAST	integer
PASSAGIERE	integer

Zu implementierende Klassen: Fahrzeug, LastKraftWagen und PersonenKraftWagen

Es gelten folgende Beziehungen:

- * Ein **LastKraftWagen** ist ein **Fahrzeug** und hat eine Eigenschaft **NUTZLAST**
- * Ein **PersonenKraftWagen** ist ein **Fahrzeug** mit der Eigenschaft **PASSAGIERE**
- * Jedes **Fahrzeug** hat die Eigenschaft **BAUJAHR**
- * Die Spalte **KATEGORIE** wird verwendet um **LastKraftWagen** von **Personenkraftwagen** zu unterscheiden. Die Werte sind „LKW“ und „PKW“.
- * Die ID Werte sollen automatisch erzeugt werden.

(4.5 Punkte, 10 Minuten)

```
@Entity
@Table(name="FAHRZEUGE")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="KATEGORIE", discriminatorType=DiscriminatorType.STRING)
public class Fahrzeug implements Serializable{
    @Id @GeneratedValue
    private long id;
    @column(name="BAUJAHR")
    private int baujahr
}

@Entity
@DiscriminatorValue("LKW")
public class PersonenKraftWagen extends Fahrzeug{
    @Column(name="PASSAGIERE")
    private int passagiere;
}

@Entity
@DiscriminatorValue("PKW")
public class LastKraftWagen extends Fahrzeug{
    @Column(name="NUTZLAST")
    private int nutzlast;
}
```

5.7. Transaktionsattribute

Mit welchen Transaktionsattributen lassen Sie zu, dass ihre Session Bean Methode ohne existierenden transaktionalen Kontext aufgerufen wird? Erläutern Sie jede einzelne Option. (6 Punkte, 6 Minuten)

- **NOTSUPPORTED**
Bean läuft ohne Transaktionskontext
- **SUPPORTS**
Methodenaufrufe nutzen den Transaktionskontext, sobald er existiert, erzeugen jedoch sonst keinen
- **NEVER**
Es darf kein Transaktionskontext existieren, ansonsten Ausnahme
- **REQUIRED**
- **REQUIRES NEW**

Container erzeugen neue Transaktion vor jedem Methodenaufruf des Beans, die vor dem Return committed wird.

5.8. Sprachelemente von JPA und EJB-QL

Welche sprachlichen Elemente stellen JPA und EJB-QL zur Verfügung, um Daten erst bei wirklichem Bedarf von der Datenbank zu laden? Hinweis: Das EJB-QL Feature überschreibt das JPA Feature um den Bedarf explizit anzumelden.
(2 Punkte, 3 Minuten)

fetchType=LAZY ist Standard für Referenzen. Damit werden die Daten für das referenzierte Objekt erst mit dem ersten expliziten Zugriff aus der Datenbank geladen.

Hinweis: Bei bereits geschlossener Session erfolgt eine LazyInitializationException

Annotation:

```
@OneToMany(fetch = FetchType.LAZY)
@OneToMany(fetch = FetchType.EAGER)
```

ejb-jar.xml:

```
<hibernate-mapping ...>
  <class name="..." table="..." lazy="true">
    ...
  </class>
</hibernate-mapping>
```

6. JMS

6.1. Die 7 fundamentale Interface der JMS API

Nennen Sie die 7 fundamentalen Interfaces der JMS API, und erläutern Sie deren Funktion. Tipp: Wenn Sie einen JMS Client explizit ausprogrammieren, verwenden Sie diese Klassen automatisch. Nennen Sie bitte nur die Interfaces, die für beide Message Paradigmen gelten.
(3.5 Punkte, 6 Minuten)

- **JAVAX.JMS.CONNECTIONFACTORY**

Sie dient dazu, Verbindungen zum Nachrichten-Service aufzubauen.

- **JAVAX.JMS.CONNECTION**

Eine Connection ist vergleichbar mit einer geöffneten Datenbankverbindung und ermöglicht einem die Kommunikation mit anderen Teilnehmern.

- **JAVAX.JMS.SESSION**

Eine Session bildet eine konkrete Verbindung mit einem oder mehreren Nachrichten-Kanälen. Quasi eine Session zwischen Customer und Producer

- **JAVAX.JMS.DESTINATION**

Eine Destination ist ein Zielpunkt, an den Nachrichten weitergeleitet werden.

- **JAVAX.JMS.MESSAGE**

Die Message ist der Kern-Bestandteil. Sie dient dem Austausch von Nachrichten und Objekten und kann sogar zum Versand von XML-Dokumenten verwendet werden.

- **MESSAGEPRODUCER**

Nachrichten senden

- **MESSAGECUSTOMER**

Nachrichten empfangen

6.2. Messaging Paradigmen

Nennen Sie die beiden Messaging Paradigmen und nennen Sie dabei auch den Hauptunterschied.
(2 Punkte, 4 Minuten)

- **POINT-TO-POINT-MODELL**

Beim Point-to-Point-Modell wird die Nachricht genau an einen Empfänger versandt. (Queue)

- **PUBLISH-AND-SUBSCRIBE-SYSTEM**

Hier wird die Nachricht an eine Gruppe interessierter Konsumenten geschickt. Da sich jede dieser Gruppen in der Regel mit genau einem Thema beschäftigt, werden diese Topic genannt.

6.3. JMS: Typische API Sequenz Code Fragment

JMS: Schreiben Sie ein typisches API Sequenz Code Fragment, indem eine Text Message zu einem Topic publiziert wird. Berücksichtigen Sie dabei, dass zwei der dazu benötigten JMS-typischen Klasseninstanzen aus dem JNDI Namensraum erlangt werden müssen. Hier geht es darum, dass Sie die API abfolge verstehen. Kein Freitext (4 Punkte, 8 Minuten)

```
InitialContext ctx = new InitialContext();
ConnectionFactory factory = (ConnectionFactory) ctx.lookup("ConnectionFactory");
javax.jms.Topic topic = (javax.jms.Topic) ctx.lookup("topic/testTopic");
Connection connect = factory.createConnection();
Session session = connect.createSession(false, Session.AUTO_ACKNOWLEDGE);
MessageProducer sender = session.createProducer(topic);
TextMessage msg = session.createTextMessage();
msg.setText("eigentliche Nachricht");
msg.setStringProperty("MessageFormat", "Message");
sender.send(msg);
connect.close();
```

7. EJB3

7.1. Remote zugreifbare EJB3.0 Session Bean

Welche minimalen Mittel müssen Sie einsetzen, um eine remote zugreifbare EJB3.0 Session Bean zu entwickeln? Beantworten Sie die Frage durch Beispiel Code. Kein Freitext.

(1.5 Punkte, 3 Minuten)

```

import javax.ejb.*
@Stateless
public class MySessionBean implements MySessionRemote{
    public MyThing doSomething(MyParam mp){}
}

@Remote
public interface MySessionRemote{
    public MyThing doSomething(MyParam mp);
}

```

7.2. Einer EJB3.0 Session Bean, Informationen über Ablaufkontext mitgeben

Was ist der einfachste Weg, einer EJB3.0 Session Bean Informationen über Ihren Ablaufkontext mitzugeben? Codezeile und kurze Erläuterung, wie der Container herausfindet, was genau er zu tun hat.

(1.5 Punkte, 4 Minuten)

Innerhalb einer Umgebung, in welcher die Session Bean ausgeführt wird, können beliebige Ressourcen definiert werden, auf die die Bean Zugriff hat.

```
@Resource SessionContext ctx;
```

Dependency Injection

Über die @Resource Annotation wird der Container angewiesen, eine Instanz vom angegebenen Typ durch ein lookup über den angegebenen Namen oder MappedName zu bestimmen und der Methode als Parameter mitzugeben. Der Container stellt die entsprechende Resource per Injektion zur Verfügung.

7.3. EJB3.0 Annotationen

EJB3.0 Wie erreichen Sie mittels Annotationen, dass a) der Container die Transaktionalität Ihrer SessionBean managed, dass b) per Default alle Methoden Ihre Bean im Prinzip Transaktionen unterstützen aber nicht unbedingt erfordern, und dass c) eine spezielle Methode im Gegensatz zu den anderen immer in einer eigenen Transaktion läuft, ohne mögliche existierende Transaktion zu gefährden? (6 Punkte, 6 Minuten)

a)

```
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
```

```
@TransactionManagement(TransactionManagerType.CONTAINER)
```

b)

```
@TransactionAttribute(TransactionAttributeType.SUPPORTS)
```

c)

```
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
```

7.3.1. Pendant bei EJB2.1

EJB2.1 Wie erreichen Sie das gleiche mittels Deklaration im Deployment Descriptor. Nennen Sie relevante XML code Fragmente, kein Freitext.
(4 Punkte, 6 Minuten)

```
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>ArtikelVerwaltungBean</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
  <container-transaction>
    <method>
      <ejb-name>ArtikelVerwaltungBean</ejb-name>
      <method-name>getArtikel</method-name>
      <method-params>
        <method-param>int</method-param>
      </method-params>
    </method>
    <trans-attribute>NotSupported</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

7.4. EJB3 JPA

EJB3: Nennen und beschreiben Sie kurz die unterschiedlichen Möglichkeiten, die JPA bietet, um Vererbung von Klassen auf Datenbnaktabellen darzustellen
(6 Punkte, 6 Minuten).

Mit Hilfe von Annotation

▪ **SINGLE_TABLE**

Eine Tabelle mit allen Attributen aller beteiligten Klassen. Ist sehr performant, aber Attribute können null sein

▪ **TABLE_PER_CLASS**

Eine Tabelle für jede konkrete Klasse. Ist jedoch unperformant, aber dafür kann not-null speziell definiert werden.

▪ **JOINED**

Hinreichend performant, bildet gemäss dritter Normalform ab

8. SECURITY

8.1. EJB2.1 und J2EE Security Rollen

EJB2.1 Wie genau werden in SessionBeans mit der J2EE Security Rollen verwendet, um Zugriff nur für User in bestimmten Rollen zuzulassen? Geben Sie beispielhaft die XML Syntax für die Berechtigungsbeschreibung im Deployment Deskriptor an, und beschreiben Sie die Funktionalität in kurzen klaren Sätzen. Hinweis: Berücksichtigen Sie den Mechanismus der Abstraktion unterschiedlicher Rollenbezeichnungen.
(4.5 Punkte, 10 Minuten)

Container-Hersteller geben praktisch immer ein Backend Authentisierungsinterface vor

Man kann eigene Implementierungen schreiben

Container-spezifische Konfiguration in: <jboss-home>\server\default\conf\login-config.xml

Programmtechnisch stehen über das EJBContext Interface, von dem SessionContext und EntityContext abgeleitet sind, die Methoden zur Verfügung:

boolean EJBContext.isCallerInRole(String role)

Principal EJBContext.getCallerPrincipal()

```
//Dieser Teil gehört nicht mehr direkt zur Aufgabe
<security-constraint>
  <web-resource-collection>
    <web-resource-name>PrivateAccountData</web-resource-name>
    <url-pattern>/private/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>PRIVATE_CUSTOMER</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

9. FREI FORMULIERTE FRAGEN

Was besagt das Transaktions-Management und worin besteht der Vorteil von Enterprise JavaBeans Transaktionen?

Wenn auf die Eingabe eines Benutzers verschiedene Dinge parallel geschehen müssen und entweder alle oder keines ausgeführt werden soll, wird ein Transaktions-Management-System benötigt. Obwohl viele Datenbanken inzwischen Transaktionen unterstützen, besteht der Vorteil von Enterprise JavaBeans darin, dass deren Transaktionen nicht allein auf den Datenaustausch mit Datenbanken beschränkt sind, sondern auch Nachrichten an entfernte Systeme einer verteilten Applikation gesendet werden können (beispielsweise über den Java Message Service JMS).

Erkläre den Unterschied von JavaBeans und Enterprise JavaBeans.

Während JavaBeans vor allem dazu eingesetzt werden, um Daten zu logischen Einheiten zusammenzufassen, sind Enterprise JavaBeans dazu gedacht Geschäftslogik aufzunehmen und vor dem Anwender zu verbergen (Information Hiding).

Der grösste Unterschied bezieht sich jedoch auf den Ort der Ausführung, denn während gewöhnliche JavaBeans in derselben Virtual Machine laufen wie die Applikation, die sie nutzen, werden Enterprise JavaBeans von einem J2EE Application Server bereitgestellt und von der lokalen Client-Anwendung aus der VM des Servers aufgerufen.

Was ist der Unterschied zwischen einem Servlet-Container und einem EJB-Container?

Ein Webserver (z.B. Tomcat) beinhaltet einen Servlet-Container, der die Laufzeitumgebung für Servlets und JSPs bildet. Genau wie diese benötigen auch die Enterprise Java Beans einen Platz, an dem sie erzeugt, mit Aufgaben betraut und schliesslich wieder entfernt werden. Diese Laufzeitumgebung ist der so genannte EJB-Container. Tomcat hat keinen EJB-Container → wir benötigen einen vollwertigen J2EE Application Server wie z.B. JBoss!

Beschreibe die unterschiedlichen Typen von EJBs.

▪ ENTITY-BEANS

Entity-Beans bilden die Schnittstelle zur Datenbank oder anderen persistenten Speichermedien. Sie repräsentieren dabei ein Objekt in der Geschäftslogik und werden verwendet, um Datensätze zu suchen oder zu erzeugen.

▪ SESSION-BEANS

Session-Beans werden dazu verwendet, um Geschäftsprozesse zu implementieren. Wenn Entity-Beans beispielsweise ein Konto repräsentieren, dann können Session-Beans für eine Überweisung zwischen Konten verwendet werden.

▪ MESSAGE-DRIVEN-BEANS

Ermöglichen eine asynchrone Kommunikation von EJB-Komponenten über den JMS. MDB sind im Grunde erweiterte Session-Beans, bei denen die beiden Partner nicht mehr warten müssen, bis der jeweils andere fertig ist, sondern stattdessen parallel weiterarbeiten können. Dazu hinterlassen sie einfach eine Nachricht über den gewünschten Service und holen sich die Antwort zu einem späteren Zeitpunkt ab.

Was gibt es für Restriktionen bei der Implementierung von EJBs?

Damit EJBs sicher von einem EJB-Container verwaltet werden können, müssen sie sich einigen Restriktionen unterwerfen. So darf eine Enterprise JavaBean unter keinen Umständen einer der folgenden Dinge tun:

- **MIT OBJEKTEN AUS DEM PACKAGE JAVA.IO AUF DATEIEN ZUGREIFEN**
- **EINEN SERVERSOCKET ERSTELLEN ODER MANIPULIEREN**
- **EIGENE THREADS ERSTELLEN ODER STARTEN**
- **NATIVE BIBLIOTHEKEN LADEN**
- **ÜBER KLASSEN DES AWT- ODER SWING-PAKETS DIREKT MIT DEM ANWENDER IN VERBINDUNG TRETEN**

Diese Einschränkungen sind nötig, da eine vom EJB-Container verwaltete EJB keine Annahme darüber machen kann, wann und in welchem Kontext sie erstellt wird.

Wie heissen die 3 Teile woraus eine EJB besteht?

- **HOME-INTERFACE**
- **REMOTE-INTERFACE**
- **SOWIE IMPLEMENTIERUNG**

Warum muss sich sowohl eine `ejb-jar.xml` und eine `jboss.xml` definieren?

Im `ejb-jar.xml` werden alle Komponenten zu einer EJB konfiguriert. Über die Konfigurationsdatei `jboss.xml` wird dem JBoss mitgeteilt, in welchem Kontext und unter welchem Namen die EJB später erreichbar sein soll. Dazu muss man den im Deployment Descriptor `ejb-jar.xml` definierten symbolischen Namen der EJB verwenden.

```
<?xml version="1.0" encoding="UTF-8" ?>
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>HelloWorld</ejb-name>
      <jndi-name>ejb/HelloWorld</jndi-name>
    </session>
  </enterprise-beans>
</jboss>
```

Wozu dient das Java-Archive (JAR) und das Enterprise Archive (EAR)?

Die EJB-Klassen können mit dem Deployment Deskriptor und der JBoss Konfigurationsdatei zu einem Java-Archiv (JAR) zusammengefasst werden, um diese leichter verwalten zu können.

Die Applikation kann nun noch zu einem so genannten Enterprise Archive (EAR) zusammengefasst werden, welches dazu gedacht ist, eine vollständige Applikation aufzunehmen, um diese leicht von einem Application Server auf einen anderen übertragen zu können. Es muss einfach noch ein Application Deployment Descriptor (`application.xml`) definiert werden, über den der JBoss Application Server die Anwendung und ihre Module später einbinden kann.

Schreiben Sie eine EJB 2.1 SessionBean mit den folgenden remote Methoden:

```
setCustomerId(long id);
CustomerBillingInfo getBillingInfo();
void setBillingInfo(CustomerBillingInfo info);
```

und der lokalen Methode

```
setLoggingPrefs(Map prefs);
```

Die Methode muss in einer eigenen Transaktion ablaufen.
Die Methode darf nur mit der Rolle „CUSTOMER_ADMIN“ aufgerufen werden.
Lassen Sie die Methoden leer.

```
public class MySessionBean implements SessionBean {
    // Methoden des Remote-Interfaces
    public void setCustomerId(long id) throws RemoteException {}

    public CustomerBillingInfo getBillingInfo() throws RemoteException {}

    public void setBillingInfo(CustomerBillingInfo info) throws RemoteException {}

    // Methoden des Local-Interfaces
    public void setLoggingPrefs(Map prefs) {}

    // Methoden des Home-Interfaces
    public void ejbCreate() throws CreateException {}
}

// Remote-Interface
public interface MySessionBeanRemote extends EJBObject {

    public void setCustomerId(long id) throws RemoteException;
    public CustomerBillingInfo getBillingInfo() throws RemoteException;
    public void setBillingInfo(CustomerBillingInfo info) throws RemoteException;
}

// Local-Interface
public interface MySessionBeanLocal extends EJBLocalObject {

    public void setLoggingPrefs(Map prefs);
}

// Home-Interface
public interface MySessionBeanHome extends EJBHome {

    public MySessionBeanRemote create() throws CreateException;
}

// LocalHome-Interface
public interface MySessionBeanLocalHome extends EJBLocalHome {

    public MySessionBeanLocal create() throws CreateException;
}
```

```
// ejb-jar.xml
<ejb-jar>
  <enterprise-beans>
    <session>
      <display-name>Session Bean with Remote Interface</display-name>
      <ejb-name>MySessionBeanRemote</ejb-name>
      <home>MySessionBeanHome</home>
      <remote>MySessionBeanRemote</remote>

      <local-home>MySessionBeanLocalHome</home>
      <local>MySessionBeanLocal</local>

      <ejb-class>MySessionBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>

  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>MySessionBeanLocal</ejb-name>
        <method-intf>Local</method-intf>
        <method-name>setLoggingPrefs</method-name>
      </method>
      <trans-attribute>RequiresNew</trans-attribute>
    </container-transaction>

    <method-permission>
      <role-name>CUSTOMER_ADMIN</role-name>
      <method>
        <ejb-name>MySessionBeanLocal</ejb-name>
        <method-name>setLoggingPrefs</method-name>
        <method-params>Map</method-params>
      </method>
    </method-permission>
  </assembly-descriptor>
</ejb-jar>
```

10. JOLLER PRÜFUNGEN

Ist es möglich SOAP Messages über Telnet zu versenden? Wenn ja: wie? Wenn nein: warum nicht?

Ja, es ist möglich!

- 1) Header eintippen, mit allen Leerzeilen
- 2) SOAP Emvelop
- 3) SOAP Header
- 4) SOAP Body

1) Erläutern Sie den Unterschied zwischen einer SOAP Message und einem SOAP RPC. 2) Wo erkennen Sie, dass es sich bei einer SOAP Message um einen RPC handelt?

1)

SOAP Message: allgemein, also Document, Wrapped, Messaging oder RPC

SOAP RPC: Spezialfall (in der Regel synchroner Prozeduraufruf)

2)

```
<service name="EchoService2" provider="java:RPC">
```

Warum wird `javax.ejb.EJBObject` von `java.rmi.Remote` abgeleitet und was hat das für Konsequenzen?

Die Kommunikation des Clients zum EJB Container soll transparent sein. Deswegen wird RMI als Basistechnologie eingesetzt. Dies hat die Konsequenz das Methoden des Remote Interface das `java.rmi.RemoteException` werfen müssen.

Warum wird `javax.ejb.EnterpriseBean` aus `javax.io.Serializable` abgeleitet und geschieht die Parameter-Übergabe und Rückgabe „by value“ oder „by reference“?

Der Container verwendet Standard Java Serializable für die Parameter- und Rückgabe-Werte. Die Argumente und Ergebnisse der Methoden des remote und remote Home Interface werden „by value“ übergeben.

Vergleichen Sie Session Beans mit Entity Beans.

Sichten	Session Bean	Entity Bean
Client	Wird für einen einzelnen Client ausgeführt	Kann gleichzeitig von mehreren Clients benutzt werden
Transaktion	Kann Transaktionen beschreiben	Kann Transaktionen beschreiben
Persistenz	Nicht	Immer
Ausfallverhalten	Werden einen Container Crash nicht überleben	Sind persistent gespeichert und überleben auch Container Crashes
Lebensdauer	Lebensdauer eher kurz	Lebensdauer durch Persistenz bestimmt
Concurrency	Mehrere Session Beans können concurrent ausgeführt werden	Mehrere Entity Beans können concurrent ausgeführt werden.

Womit (Klasse / Interface) kommunizieren Enterprise Beans, um Runtime Context Informationen des EJB Containers zu erhalten?

`javax.ejb.EJBContext` des Containers

Welche Informationen werden beim Passivation einer Session Bean nicht serialisiert / gespeichert?

- 1) DB Connection Infos
- 2) Transiente Variablen

Was geschieht, wenn eine Business Methode ein Bean aufruft, welches seinen Timeout bereits erreicht hat?

Eine `ObjectNotFoundException` wird geworfen

Nennen Sie 4 Unterschiede zwischen BMPs und CMPs.

- 1) Bei CMP werden die Datenbankzugriffe durch den Container gemacht
- 2) Bei CMP ist die Bean Klasse abstrakt
- 3) Bei CMP werden die Felder mit abstrakten getter und setter Methoden definiert.
- 4) Bei CMP gibt es keine ejbFinderMethoden innerhalb der Bean-Klasse

Aus welchen Blöcken besteht eine SOAP Message im Allgemeinen?

Envelope, Header, Body

Wie kann eine SOAP Message an den richtigen Empfänger gelangen, falls kein Header definiert ist?

WSDL definiert die URL des Endpoints

WSDL definiert die URI des Services (am Endpoint)

WSDL definiert die Methoden des Services

WSDL definiert die Parameter des Services

Schreiben / skizzieren Sie eine SOAP message, welche einen Hello World Service mit einem String Parameter aufruft.

```
<?xml version="1.0" encoding="utf-8" ?>
<SOAP-ENV:Envelope xmlns:SOAPENV=
http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:xsd1=http://www.xmlbus.com/PurchaseOrderService-xsd
xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <nsl:postHelloWorld xmlns:nsl="http://.../HelloWorldService">
      <return xsi:type="xsd:String">Hello World</return>
    </nsl:postPurchaseOrderResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Warum wurde der SOAP Header recht ungenau spezifiziert (Header Elemente sind fakultativ, der Inhalt ist nicht spezifiziert)? Nennen Sie einen möglichen Grund.

Der Header muss firmenintern und firmenextern verwendbar sein. Daher muss im Standard jeder Fall berücksichtigt werden.

Wozu kann das Attribut „mustUnderstand“ verwendet werden?

Die Beachtung und Verarbeitung von Header-Element ist optional, ausser das Attribut mustUnderstand ist logisch wahr, also true oder 1. Kann ein SOAP-Knoten ein Header-Element in einem solchen Fall nicht bearbeiten, so muss dieser einen SOAP-Fault erzeugen und an den Sender zurückschicken. In diesem Fall ist ein SOAP Knoten jeder Rechner vom Sender zum Empfänger, der SOAP versteht.

Worum geht es eigentlich bei der EJB Architektur, was definiert diese? (eine gültige Antwort)

1. Transaktionskomponenten
2. Verteilte Objektkomponenten
3. Server-seitige Komponenten
4. alle obigen

4)

Welche der folgenden Aussagen über EJB Komponenten, Containers und Applikations-Servern treffen zu ? (Mehrere gültige Antworten)

1. Komponenten laufen in Containern
2. Container werden durch Applikations-Server zur Verfügung gestellt
3. Container laufen in Komponenten
4. Applikations-Server laufen in Containern

1) und 2)

Welcher Unterschied besteht zwischen einem Enterprise JavaBean Container und einem Enterprise JavaBean Server? (eine gültige Antwort)

1. Container laufen innerhalb eines Servers
2. Server laufen innerhalb eines Containers
3. pro Container kann nur ein Server ausgeführt werden
4. pro Server kann nur ein Container ausgeführt werden
5. Containers und Server haben die selbe Funktion

1)

Welche der folgenden Komponente ist verantwortlich für die Ausführung der EJB Komponenten? (eine gültige Antwort)

1. ein Web- Server
2. ein Applikations-Server
3. ein EJB-Container
4. ein Datenbank-Server

3)

Womit kommunizieren Enterprise Beans, um Runtime Context Informationen des EJB Containers zu erhalten? (eine gültige Antwort)

1. javax.ejb.EJBContext des Containers
2. ein JNDI Context
3. ein javax.ejb.EJBHome Objekt des Containers
4. ein javax.eib.EJBMetaData Objekt des Containers

1)

Mit Hilfe welches Interfaces kreiert oder findet eine Applikation Enterprise Beans? (eine gültige Antwort)

1. java.rmi.Remote
2. javax.ejb.EJBHome
3. javax.ejb.EJBObject
4. javax.ejb.EntityBean

2)

Mit welchem Typ Enterprise Bean werden Business Objekte realisiert? (eine gültige Antwort)

1. javax.ejb.EnterpriseBean
2. javax.rmi.Remote
3. javax.ejb.SessionBean
4. javax.ejb.EntityBean

4)

Mit welchem Typ Enterprise Bean werden Zustandsinformationen von Applikationen festgehalten? (eine gültige Antwort)

1. javax.ejb.EnterpriseBean
2. javax.rmi.Remote
3. javax.ejb.SessionBean
4. javax.ejb.EntityBean

3)

Welches Interface muss eine EJB implementieren damit eine Applikation deren Methoden einsetzen kann? (eine gültige Antwort)

1. javax.ejb.EntityBean
2. javax.ejb.EJBHome
3. javax.ejb.EJBObject
4. javax.rmi.Remote

3)

Welche Interfaces enthalten Methoden, mit deren Hilfe EJBs entfernt werden können?

(Mehrere gültige Antworten)

1. EJBHome
2. EJBContext
3. EJBObject
4. EntityContext
5. EnterpriseBean

1) und 3)

Welche der folgenden Properties wird bei einer EJB Passivation gespeichert?
(Mehrere gültige Antworten)

1. EJBHome Referenz
2. transiente Variablen
3. User Transaktionen
4. NamingContext
5. EJBObject Referenz

1), 3), 4) und 5)

Welche der folgenden sind im EJB Code nicht gestattet? (Mehrere gültige Antworten)

1. Thread Management
2. transiente Variablen
3. Socket Listeners
4. JDBC Calls
5. Transaktions-Management

1) und 3)

Welche Eigenschaften sollte die PrimaryKey Klasse besitzen?

(Mehrere gültige Antworten)

1. sie sollte das java.io.serialize Interface implementieren
2. sie kann aus einem Java Basisdatentyp (primitive data type) bestehen
3. sie muss einen Null-Argument-Konstruktor (Default-Konstruktor) besitzen
4. sie kann aus einer Java Wrapper-Klasse bestehen
5. sie muss die hashCode() Methode überschreiben

1), 3) und 4)

Wie kann eine Bean eine Referenz auf sich an eine andere Bean weitergeben?
(eine gültige Antwort)

1. mittels this
2. mittels des Bean Klassennamen
3. eine Referenz an das Remote Interface der Bean übergeben
4. eine EJBObject Referenz aus dem SessionContext/EntityContext übergeben
5. das geht überhaupt nicht

4)

Wann wird die ejbPostCreate() Methode aufgerufen und welche Parameter hat die Methode?

Nach ejbCreate(), gleiche Parameter wie ejbCreate()

Wann (exakt) im Lebenszyklus einer Container Managed Bean, gilt diese als kreiert? (eine gültige Antwort)

1. direkt bevor die ejbCreate() Methode ausgeführt wird
2. direkt nachdem die ejbCreate() Methode ausgeführt wurde
3. nachdem Daten der CMP Bean an den persistenten Datenspeicher übermittelt wurden
4. während der Ausführung der ejbPostCreate() Methode

2)

Analysieren Sie die folgende Web Service Description und beantworten sie die folgenden Fragen zum Aufbau einer WSDL.

- 1) Aus welchen Blöcken besteht eine WSDL generell?
- 2) Ist aus einer WSDL Beschreibung ein eindeutiger Client konstruierbar?

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="services:StockTicker"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:impl="services:StockTicker"
xmlns:intf="services:StockTicker"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:message name="getStockQuotes"> </wsdl:message>
  <wsdl:message name="getStockResponse">
    <wsdl:part name="getQuoteReturn" type="xsd:string"/>
  </wsdl:message>
  <wsdl:portType name="StockTicker">
    <wsdl:operation name="getQuotes">
      <wsdl:input name="getQuotesRequest" message="impl:getQuotesRequest"/>
      <wsdl:output name="geQuotesResponse" message="impl:getQuotesResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="StockTickerSoapBinding" type="impl:StockTicker">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getQuote">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getQuoteRequest">
        <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="services:StockTicker"/>
      </wsdl:input>
      <wsdl:output name="getQuoteResponse">
        <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="services:StockTicker"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="StockTickerService">
    <wsdl:port name="Zeitansage" binding="impl:StockTickerSoapBinding">
      <wsdlsoap:address
location="http://localhost:8080/axis/services/StockTicker"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

- 1) Typen, Messages, PortTypes, Bindings, Ports, Services
- 2) Nein, denn es fehlen die Informationen über den Methodeninhalt (unbekannt)