Workflow Esame PD (esempio Oggetti Smarriti)

Ordine delle classi allo scritto:

Entità, Interfacce, DB Producer, DB Populator, EJB, Client EJB
JMS: MDB (messageWrapper, notification e poi bean effettivo), Client
Web Services: Client

Creazione dei progetti su NetBeans:

4 progetti: server, client con main semplice e main per i messaggi, server con web services e client con web services.

OggettiEJB:

- package database per populator e producer;
- package jms per bean, notification e wrapper;
- package server con entity, EJB ed interceptor;
- Properties -> Libraries -> Add JAR/Folder -> javaee-api-7.0.jar
- La dipendenza da OggettiClient viene inserita in automatico quando si crea l'ejb e si spunta la casella interfaccia remota
- in persistence.xml bisogna mettere come nome 'EsamePU', come data source 'jdbc/EsameDS' e come generation strategy 'drop and create'
- nel file beans.xml bisogna cambiare il valore di 'bean-discovery-mode' da "annotated" a "all"

OggettiClient:

- package meta-inf creato in automatico quando si crea il file persistence.xml (stessa struttura di sopra);
- package jms in cui si copia il Wrapper e si scrive il main per i messaggi;
- package oggetticlient (stesso nome del progetto, senza maiuscole, creato in automatico) in cui inserire il main semplice;
- package server in cui copiare l'entity e scrivere l'interfaccia remota dell'EJB;
- Properties -> Libraries -> aggiungi in Classpath le librerie GlassFishClient e Java EE 7 API Library (è vitale che glassfish sia il primo)

OggettiWebService:

- package database copiato da OggettiEJB
- package server copiato da OggettiEJB. ***In OggettoEJB bisogna aggiungere come prima annotazione @WebService***
- Anche qui aggiungere javaee-api-7.0.jar, mentre la dipendenza dal client semplice viene sempre in automatico
- beans e persistence come sopra

OggettiWebClient:

- Importante: da creare dopo aver completato, buildato e deployato il WebService
- dopo la creazione del progetto, tasto destro sul nome e clicca su new -> 'Web Service Client'; inserire il riferimento all'EJB presente nel progetto OggettiWebService e premere finish. Ora si può scrivere senza problemi il main.

OggettiEJB: **public class Oggetto implements Serializable,** l'entity class. Esternamente si annota:

- @Entity
- @Table(name = "Oggetto")
- @NamedQueries({
- @NamedQuery(name = FIND_ALL, query = "SELECT o FROM Oggetto o"),
- ... [nessuna virgola dopo l'ultima query]
- }).

Prima cosa da scrivere:

- private static final long serialVersionUID = 1L;
- public static final String FIND_ALL = "Oggetto.findAll";
- ...

Sopra il parametro id bisogna annotare:

- @Id
- @GeneratedValue(strategy = GenerationType.AUTO)

Nomi degli altri parametri ed ordine da prendere dalla traccia e dalla tabella con i dati di test. Queste cose possono essere lasciate come commento, dato che poi saranno generate: 3 costruttori (vuoto, tutti i parametri tranne id, tutti i parametri), getter e setter per tutti i parametri, toString(), hashCode() ed equals(Object obj).

OggettiClient: **public interface OggettoEJBRemote,** interfaccia remota che dichiara i metodi *create|modify|deleteOggetto(Oggetto o)* e poi tutte le query, ad esempio *List<Oggetto> findByCategoria(String categoria)*. Viene annotata esternamente con:

- @Remote

OggettiEJB: **public class DatabaseProducer,** la classe che produce il DB. Presenta soltanto le annotazioni:

- @Produces
- @PersistenceContext(unitName = "EsamePU")

sopra ad un *private EntityManager em.*

OggettiEJB: **public class DatabasePopulator,** la classe che popola il DB e poi lo ripulisce. Viene annotato esternamente con:

- @Singleton
- @LocalBean
- @Startup
- @DataSourceDefinition(
- className = "org.apache.derby.jdbc.EmbeddedDataSource40",
- name = "java:global/jdbc/EsameDS",
- user = "APP",

- password = "APP",
- properties = {"connectionAttributes=;create=true"}
-)

Mentre all'interno della classe abbiamo:

- @Inject, sopra a *private CircoloEJB ejb.*
- @PostConstruct, sopra al metodo *public void populateDB()*.
- @PreDestroy, sopra al metodo *public void clearDB().*

OggettiEJB: **public class Interceptor,** l'eventuale classe interceptor. Dichiara soltano una *private static hashmap<String, Integer>* per tracciare il nome del metodo ed il numero di invocazioni. Annotatazioni:

- @javax.interceptor.Interceptor, sopra la classe.
- @AroundInvoke, sopra al metodo *public Object interceptor(InvocationContext ic) throws Exception*.

OggettiEJB: **public class OggettoEJB implements OggettoEJBRemote,** la classe che implementa l'interfaccia remota. Viene annotata con:

- @Stateless
- @LocalBean
- @Interceptors(Interceptor.class), se necessario.
- @Inject, sopra al private EntityManager em.
- @Override, sopra ciascun metodo.

OggettiClient: **public class OggettiClient,** la prima main class. Non usa annotazioni, ma c'è il seguente lookup:

- OggettoEJBRemote ejb = (OggettoEJBRemote) ctx.lookup("java:global/OggettiEJB/OggettoEJB!server.OggettoEJBRemote");

È importante che il main abbia un *throws NamingException.*

OggettiEJB: **public class MessageWrapper implements Serializable,** il wrapper per passare in modo comodo id e parametro da cambiare di un certo oggetto. Non ha annotazioni, ma richiede (commentabile) un costruttore vuoto, un costruttore con tutti i parametri, getters e setters. Implementa Serializable, quindi la prima riga:

- private static final long serialVersionUID = 1L;

OggettiEJB: **public class ModificaStatoNotification,** la classe che lancia l'evento di modifica tramite Observer. Presenta un'unico metodo *public void notify(@Observes Oggetto o)* e non ha alcuna annotazione.

OggettiEJB: **public class OggettoMDB implements MessageListener,** il bean jms. Esternamente si può annotare in due modi:
Cherry scrive soltanto

- *@MessageDriven (mappedName = "jms/javaee7/Topic")*

Dovrebbe essere simile alla soluzione di Pasku, che invece scrive:

- @MessageDriven(activationConfig ={
- @ActivationConfigProperty(propertyName = "clientId", propertyValue = "jms/javaee7/Topic"),
- @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue = "jms/javaee7/Topic"),
- @ActivationConfigProperty(propertyName = "subscriptionDurability", propertyValue = "Durable"),
- @ActivationConfigProperty(propertyName = "subscriptionName", propertyValue = "jms/javaee7/Topic"),
- @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
- })

La differenza credo sia soltanto su subscriptionDurability, che Cherry lascia con il default "nonDurable". Non ho voglia di capire se posso metterle insieme oppure no.
All'interno della classe si usano le seguenti annotazioni:

- @Inject, sopra a *private CircoloEJB ejb;*
- @Inject, sopra a *Event<Oggetto> event;*

OggettiClient: **public class OggettiJMSClient,** la seconda classe main che manda il messaggio con le informazioni da cambiare. Non usa annotazioni, ma ci sono i seguenti lookup:

- Destination topic = (Destination) ctx.lookup("jms/javaee7/Topic");
- ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/javaee7/ConnectionFactory");

È importante che il main abbia un *throws NamingException.*

OggettiWebClient: **public class OggettiWebClient,** main che usa i servizi. Nessuna annotazione e nessun lookup, i file wsdl sono creati in automatico da NetBeans.