Il Ricambio Auto

traccia

Scrivere un insieme di Enterprise JavaBeans e client che rappresentino un archivio di parti di ricambio per automobili, persistente sul DB, che contengano le informazioni su: ID (int), nome del pezzo, categoria (Motore, Freni, Filtri, Carrozzeria, Accensione, Sospensioni, Impianto Elettrico), prezzo, disponibilità, pezzi venduti.

a) Tramite JPA si deve gestire l'archivio persistente dei prodotti su DB (EsameDB) dove la chiave primaria è la id. [13 punti]

1. Devono essere previste delle query per id, per categoria ed una query che restituisce tutti i pezzi in magazzino.
2. Deve essere previsto un bean singleton che inizializzi l'archivio
3. Scrivere un client basato su invocazione di un bean stateless che stampi:

a. Tutti i pezzi disponibili in archivio
b. Tutti i pezzi di una data categoria data in input
c. Tutti i pezzi per i quali bisogna procedere con un ordine (disponibilità <10)

b) Scrivere un Interceptor che provvede a mantenere un computo totale del numero di volte un metodo è stato chiamato, stampandolo a video (Esempio: «Chiamata n. 5: metodo xxxx») [5 punti]

c) Scrivere un client basato su messaggi che invia un messaggio per aggiornare la quantità di pezzi venduti per un dato pezzo di ricambio. Il messaggio contiene il numero di nuove vendite effettuate per un dato pezzo (e la sua id), e si deve prevedere l'aggiornamento del valore delle vendite (cioè va sommato al precedente il valore passato nel messaggio). Quando viene cambiato tale valore, tramite un evento viene stampato sulla console del server un messaggio di avviso. [5 punti]

d) rendendo i metodi del bean invocabili come web service, scrivere un client basato su invocazione di web service che modifichi il prezzo di un pezzo (chiesto all'utente).

Il Ricambio Auto

Analisi traccia passo passo

Scrivere un insieme di Enterprise JavaBeans e client che rappresentino un archivio di parti di ricambio per automobili, persistente sul DB, che contengano le informazioni su: ID (int), nome del pezzo, categoria (Motore, Freni, Filtri, Carrozzeria, Accensione, Sospensioni, Impianto Elettrico), prezzo, disponibilità, pezzi venduti.

Scrivere un insieme di Enterprise JavaBeans e client che rappresentino un archivio di parti di **ricambio per automobili**, persistente sul DB, che contengano le informazioni su: **ID (int), nome del pezzo, categoria (Motore, Freni, Filtri, Carrozzeria, Accensione, Sospensioni, Impianto Elettrico), prezzo, disponibilità, pezzi venduti**.

```
@Entity
public class RicambioAuto implements Serializable{

    private int id;
    private String categoria;
    private String nome;
    private int disponibilita;
    private int pezziVenduti;
    private float prezzo;

   // constructors (empty and with all parameters), getters and setters
}
```

Scrivere un insieme di Enterprise JavaBeans e client che rappresentino un archivio di parti di ricambio per automobili, persistente sul DB, che contengano le informazioni su: ID (int), nome del pezzo, categoria (Motore, Freni, Filtri, Carrozzeria, Accensione, Sospensioni, Impianto Elettrico), prezzo, disponibilità, pezzi venduti.

a) Tramite JPA si deve gestire l'archivio persistente dei prodotti su DB (EsameDB) dove la chiave primaria è la id. [13 punti]

Scrivere un insieme di Enterprise JavaBeans e client che rappresentino un archivio di parti di ricambio per automobili, persistente sul DB, che contengano le informazioni su: ID (int), nome del pezzo, categoria (Motore, Freni, Filtri, Carrozzeria, Accensione, Sospensioni, Impianto Elettrico), prezzo, disponibilità, pezzi venduti.

a) Tramite JPA si deve gestire l'archivio persistente dei prodotti su DB (EsameDB) dove la **chiave primaria è la id**. [13 punti]

```
@Entity
public class RicambioAuto implements Serializable{
   @Id
   private int id;
   private String categoria;
   private String nome;
   private int disponibilita;
   private int pezziVenduti;
   private float prezzo;

  // constructors (empty and with all parameters), getters and setters
}
```

Scrivere un insieme di Enterprise JavaBeans e client che rappresentino un archivio di parti di ricambio per automobili, persistente sul DB, che contengano le informazioni su: ID (int), nome del pezzo, categoria (Motore, Freni, Filtri, Carrozzeria, Accensione, Sospensioni, Impianto Elettrico), prezzo, disponibilità, pezzi venduti.

a) Tramite JPA si deve gestire l'archivio persistente dei prodotti su DB (EsameDB) dove la chiave primaria è la id. [13 punti]

1. Devono essere previste delle **query** per **id**, per **categoria** ed una query che restituisce **tutti** i pezzi in magazzino.

```
import static ricambi.RicambioAuto.TROVA_TUTTI;
import static ricambi.RicambioAuto.TROVA_PER_CATEGORIA;
import static ricambi.RicambioAuto.TROVA_PER_ID;

@Entity
@NamedQueries({
   @NamedQuery(name = TROVA_TUTTI, query = "SELECT r FROM RicambioAuto r"),
   @NamedQuery(name = TROVA_PER_CATEGORIA, query = "SELECT r FROM RicambioAuto r WHERE r.categoria = :categoria"),
   @NamedQuery(name = TROVA_PER_ID, query = "SELECT r FROM RicambioAuto r WHERE r.id = ?1"),
})
public class RicambioAuto implements Serializable{
   public static final String TROVA_TUTTI = "RicambioAuto.trovaTutti";
   public static final String TROVA_PER_CATEGORIA = "RicambioAuto.trovaPerCategoria";
   public static final String TROVA_PER_ID = "RicambioAuto.trovaPerId";

   …
}
```

```
@Stateless
@LocalBean
public class RicambioAutoEJB implements RicambioAutoEJBRemote { … }

@Remote
public interface RicambioAutoEJBRemote {
   void aggiungiRicambioAuto(RicambioAuto a);
   void rimuoviRicambioAuto(RicambioAuto a);
   RicambioAuto aggiornaRicambioAuto(RicambioAuto a);

   List<RicambioAuto> cercaTuttiRicambioAuto();
   RicambioAuto cercaPerId(Integer id);
   List<RicambioAuto> cercaPerCategoria(String categoria);
}
```

+ **beans.xml** (con “annotated” sostituito da “all”)

Scrivere un insieme di Enterprise JavaBeans e client che rappresentino un archivio di parti di ricambio per automobili, persistente sul DB, che contengano le informazioni su: ID (int), nome del pezzo, categoria (Motore, Freni, Filtri, Carrozzeria, Accensione, Sospensioni, Impianto Elettrico), prezzo, disponibilità, pezzi venduti.

a) Tramite JPA si deve gestire l'archivio persistente dei prodotti su DB (**EsameDB**) dove la chiave primaria è la id. [13 punti]

1. Devono essere previste delle query per id, per categoria ed una query che restituisce tutti i pezzi in magazzino.
2. Deve essere previsto un **bean singleton che inizializzi l'archivio**

```
@Singleton
@Startup
@DataSourceDefinition(
  name = "java:global/jdbc/EsameDS", databaseName = "EsameDB", //other settings
)
public class DatabasePopulator {
   @Inject
   private RicambioAutoEJB ricambioAutoEjb;

   @PostConstruct
   private void populateDB(){…}
   @PreDestroy
   private void clearDB(){…}
}
```

+ DatabaseProducer
+ persistence.xml

ID	Nome	Categoria	Prezzo	Disponibilità	Vendite
1	Sensore Temperatura	Motore	18,22	100	1000
2	FAP	Filtri	188,47	9	1
3	Dischi Freni	Freni	18,15	2000	5000

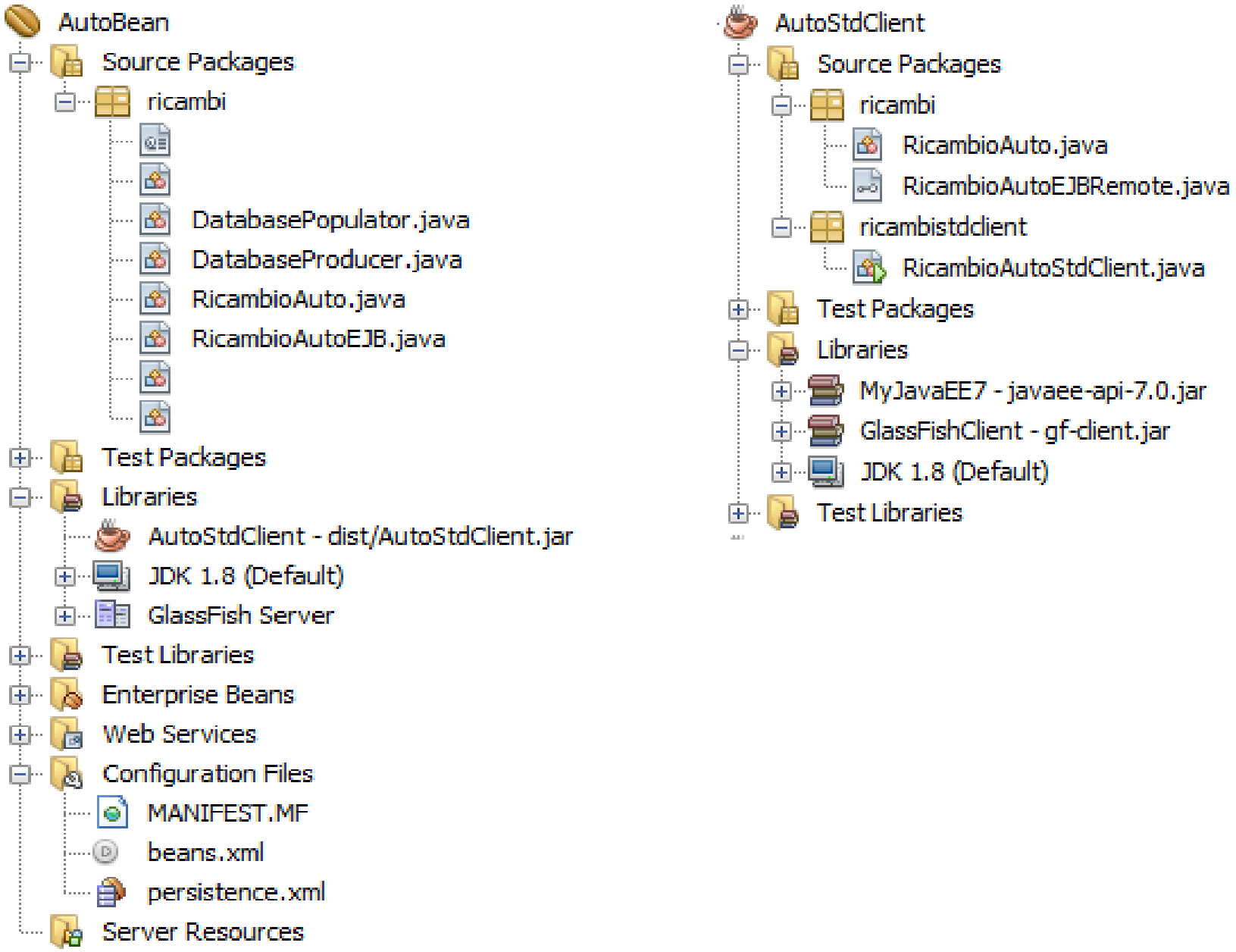
Scrivere un insieme di Enterprise JavaBeans e client che rappresentino un archivio di parti di ricambio per automobili, persistente sul DB, che contengano le informazioni su: ID (int), nome del pezzo, categoria (Motore, Freni, Filtri, Carrozzeria, Accensione, Sospensioni, Impianto Elettrico), prezzo, disponibilità, pezzi venduti.

a) Tramite JPA si deve gestire l'archivio persistente dei prodotti su DB (EsameDB) dove la chiave primaria è la id. [13 punti]

1. Devono essere previste delle query per id, per categoria ed una query che restituisce tutti i pezzi in magazzino.
2. Deve essere previsto un bean singleton che inizializzi l'archivio
3. Scrivere un **client** basato su invocazione di un **bean stateless** che stampi:

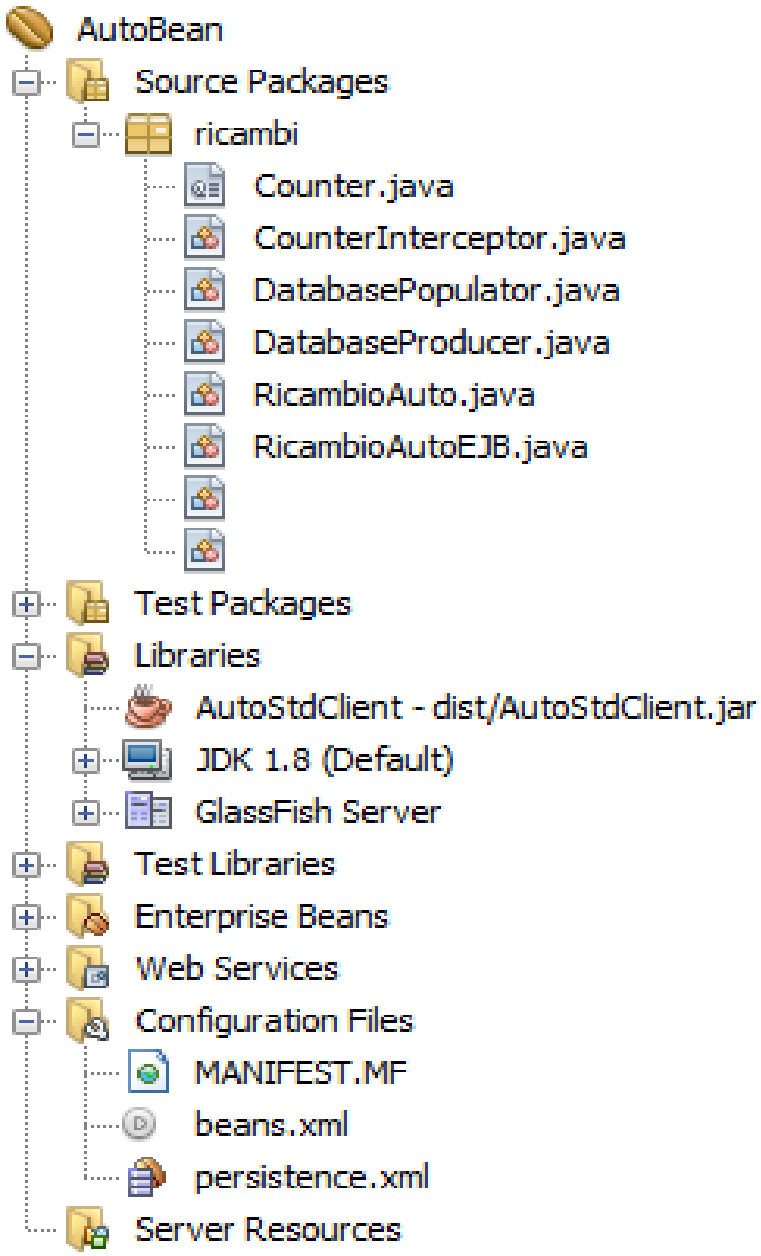
a. **Tutti i pezzi** disponibili in archivio
b. Tutti i **pezzi di una data categoria** data in input
c. Tutti i **pezzi** per i quali bisogna procedere con un ordine (**disponibilità <10**)

```
public class RicambioAutoStdClient {

   private static RicambioAutoEJBRemote ricambiEJB;

   public static void main(String[] args) throws NamingException {
      Context ctx = new InitialContext();

      ricambiEJB = (RicambioAutoEJBRemote)
            ctx.lookup("java:global/AutoBean/RicambioAutoEJB!ricambi.RicambioAutoEJBRemote");

      // query invocations
}
```


AutoBean
Source Packages
ricambi
DatabasePopulator.java
DatabaseProducer.java
RicambioAuto.java
RicambioAutoEJB.java
Test Packages
Libraries
AutoStdClient - dist/AutoStdClient.jar
JDK 1.8 (Default)
GlassFish Server
Test Libraries
Enterprise Beans
Web Services
Configuration Files
MANIFEST.MF
beans.xml
persistence.xml
Server Resources
AutoStdClient
Source Packages
ricambi
RicambioAuto.java
RicambioAutoEJBRemote.java
ricambistdclient
RicambioAutoStdClient.java
Test Packages
Libraries
MyJavaEE7 - javaee-api-7.0.jar
GlassFishClient - gf-client.jar
JDK 1.8 (Default)
Test Libraries

b) Scrivere un **Interceptor** che provvede a mantenere un computo totale del numero di volte un metodo è stato chiamato,
stampandolo a video (Esempio: «Chiamata n. 5: metodo xxxx») [5 punti]

```
@InterceptorBinding
@Retention(RUNTIME)
@Target({METHOD, TYPE})
public @interface Counter {}

@Interceptor
@Counter
public class CounterInterceptor {
   private HashMap<String, Integer> counterMap = new HashMap<String, Integer>();
   @AroundInvoke
   public Object logMethod(InvocationContext ic) throws Exception{…}
}

@Counter
public class RicambioAutoEJB implements RicambioAutoEJBRemote {…}

+

<beans …>
      <interceptors>
            <class>ricambi.CounterInterceptor</class>
       </interceptors>
</beans>
```
In **beans.xml**


AutoBean
Source Packages
ricambi
Counter.java
CounterInterceptor.java
DatabasePopulator.java
DatabaseProducer.java
RicambioAuto.java
RicambioAutoEJB.java
Test Packages
Libraries
AutoStdClient - dist/AutoStdClient.jar
JDK 1.8 (Default)
GlassFish Server
Test Libraries
Enterprise Beans
Web Services
Configuration Files
MANIFEST.MF
beans.xml
persistence.xml
Server Resources

b) Scrivere un Interceptor che provvede a mantenere un computo totale del numero di volte un metodo è stato chiamato, stampandolo a video (Esempio: «Chiamata n. 5: metodo xxxx») [5 punti]

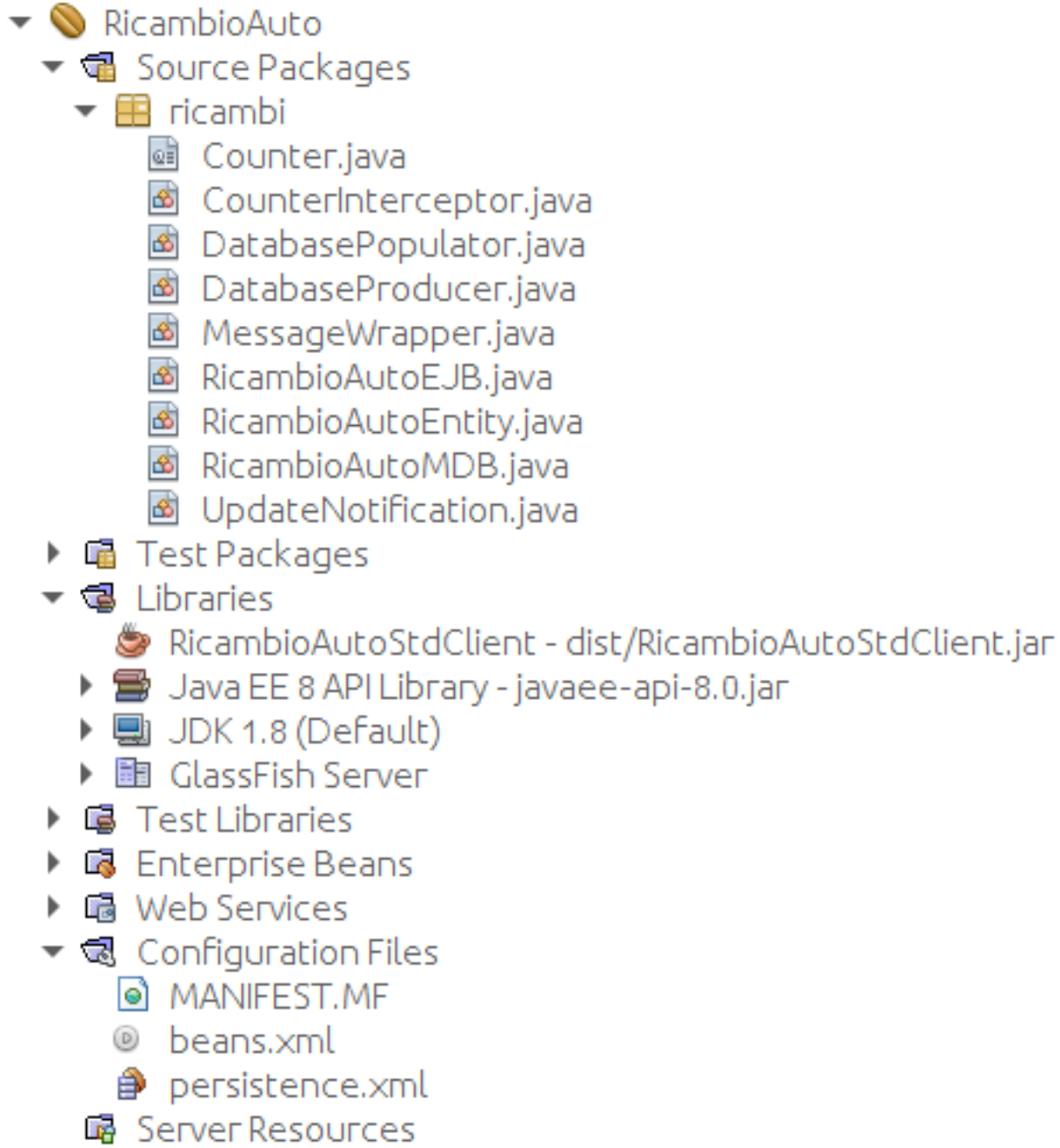
c) Scrivere un **client basato su messaggi** che invia un **messaggio** per aggiornare la **quantità di pezzi venduti** per un dato pezzo di ricambio. Il messaggio contiene il numero di nuove vendite effettuate per un dato pezzo (e la sua **id**), e si deve prevedere **l'aggiornamento del valore delle vendite** (cioè va sommato al precedente il valore passato nel messaggio). Quando viene cambiato tale valore, tramite un **evento** viene stampato sulla console del server un messaggio di avviso. [5 punti]

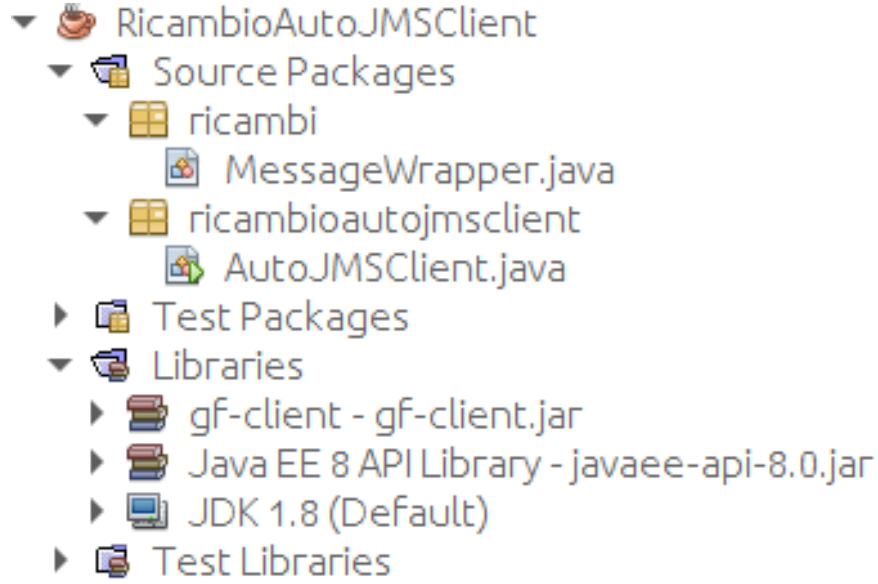
```
public class MessageWrapper implements Serializable{
    private Integer id;
    private Integer vendite;

    // constructor, getters, setters
}
```

```
public class AutoJMSClient {

    public static void main(String[] args) throws NamingException{
        Context cxt  = new InitialContext();
        ConnectionFactory cf = (ConnectionFactory) cxt.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination) cxt.lookup("jms/javaee7/Topic");

        MessageWrapper msgContent = new MessageWrapper(1, 10);

        try(JMSContext jmsCxt = cf.createContext()){
            jmsCxt.createProducer().
                send(topic, msgContent);
        }
    }
}
```

```
@MessageDriven(mappedName = "jms/javaee7/Topic")
public class RicambioAutoMDB implements MessageListener {
   @Inject    private RicambioAutoEJB ricambiejb;
   @Inject    Event<RicambioAutoEntity> event;

   @Override
   public void onMessage(Message msg) {
      try {
         MessageWrapper msgContent = msg.getBody(MessageWrapper.class);

         Integer id = msgContent.getId();
         Integer itemsSold = msgContent.getVendite();

         RicambioAutoEntity a = ricambiejb.cercaPerId(id);
         a.setPezziVenduti(a.getPezziVenduti()+itemsSold);
         a = ricambiejb.aggiornaRicambioAuto(a);

         event.fire(a);
      } catch (JMSException ex) {
         Logger.getLogger(RicambioAutoMDB.class.getName()).log(Level.SEVERE, null, ex);
      }
   }
}
```

```
public class UpdateNotification {
    public void notify(@Observes RicambioAuto ricambio){
        System.out.println(ricambio.getId() +" has been updated. Status: " + ricambio);
    }
}
```





+ creare Topic e
Connection Factory sulla console di GlassFish

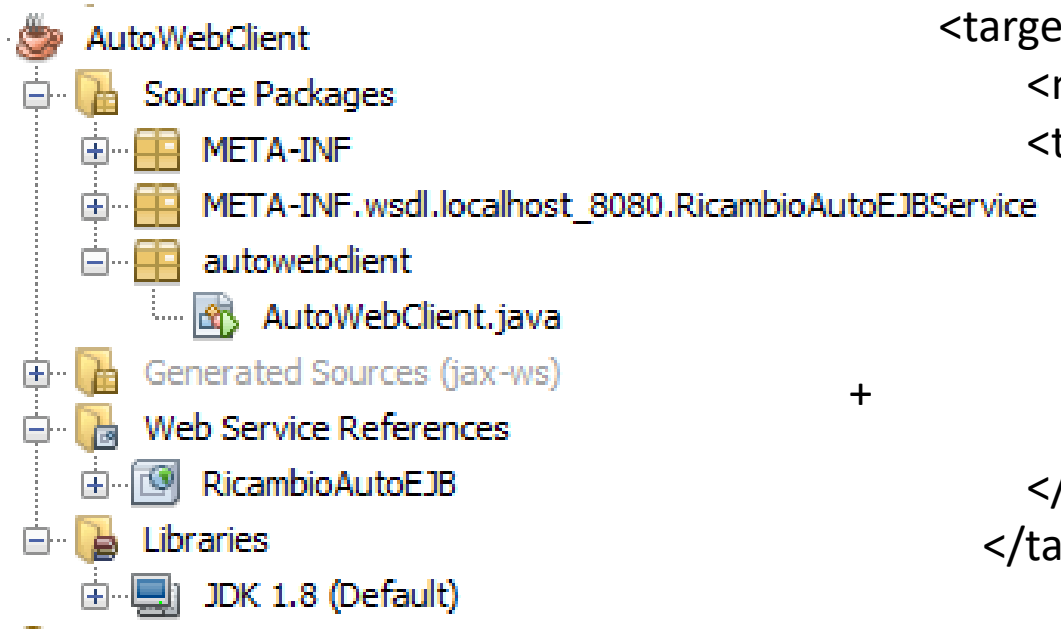
b) Scrivere un Interceptor che provvede a mantenere un computo totale del numero di volte un metodo è stato chiamato, stampandolo a video (Esempio: «Chiamata n. 5: metodo xxxx») [5 punti]

c) Scrivere un client basato su messaggi che invia un messaggio per aggiornare la quantità di pezzi venduti per un dato pezzo di ricambio. Il messaggio contiene il numero di nuove vendite effettuate per un dato pezzo (e la sua id), e si deve prevedere l'aggiornamento del valore delle vendite (cioè va sommato al precedente il valore passato nel messaggio). Quando viene cambiato tale valore, tramite un evento viene stampato sulla console del server un messaggio di avviso. [5 punti]

d) rendendo i metodi del bean invocabili come **web service**, scrivere un **client** basato su invocazione di web service che **modifichi il prezzo di un pezzo** (chiesto all'utente).

@XmlRootElement
public class RicambioAuto implements Serializable{…}

@WebService
public class RicambioAutoEJB implements RicambioAutoEJBRemote {…}



+

```
<target name="wsimport-init" depends="init">
    <mkdir dir="${build.generated.sources.dir}/jax-ws"/>
    <taskdef name="wsimport" classname="com.sun.tools.ws.ant.WsImport">
      <classpath>
        <fileset dir="./extra-lib">
          <include name="**/*.jar"/>
        </fileset>
      </classpath>
    </taskdef>
  </target>
```

In javaws-build.xml

+ librerie javaws in una nuova directory chiamata extra-lib