

@Entity

@NamedQueries {

- @NamedQuery (name = FIND\_ALL, query = "SELECT m FROM Ministro m"),
- @NamedQuery (name = FIND\_BY\_ID, query = "SELECT m FROM Ministro m WHERE m.id = ?1"),
- @NamedQuery (name = FIND\_BY\_COGNOME, query = "SELECT m FROM Ministro m WHERE m.cognome = cognome"),
- @NamedQuery (name = FIND\_BY\_PARTITO, query = "SELECT m FROM Ministro m WHERE m.partito = partito"),
- @NamedQuery (name = FIND\_BY\_MINISTERO, query = "SELECT m FROM Ministro m WHERE m.ministero = ministero"),
- @NamedQuery (name = FIND\_BY\_PORTAFOGLI, query = "SELECT m FROM Ministro m WHERE m.portafogli = true"),
- @NamedQuery (name = FIND\_BY\_SOTTOSEGRETARIO, query = "SELECT m FROM Ministro m WHERE m.sottosegretario > 2"),
- @NamedQuery (name = FIND\_BY\_DIPENDENTI, query = "SELECT m FROM Ministro m WHERE m.dipendenti > dipendenti")

})

public class Ministro implements Serializable {

```

    public static final String FIND_ALL = "Ministro.findAll";
    public static final String FIND_BY_ID = "Ministro.findById";
    public static final String FIND_BY_COGNOME = "Ministro.findByCognome";
    public static final String FIND_BY_PARTITO = "Ministro.findByPartito";
    public static final String FIND_BY_MINISTERO = "Ministro.findByMinistero";
    public static final String FIND_BY_PORTAFOGLI = "Ministro.findByPortafogli";
    public static final String FIND_BY_SOTTOSEGRETARIO = "Ministro.findBySottosegretario";
    public static final String FIND_BY_DIPENDENTI = "Ministro.findByDipendenti";
  
```

@Id

@GeneratedValue

int id;

@Most null

```

    private String cognome;
    private String nome;
    private String ministero;
    private int mSottosegretario;
    private int dipendenti;
    private Boolean portafoglio;
  
```

```

    // costruttore pieno
    // costruttore vuoto
    // getters and setters
    // toString
  
```

```
private ent dipendenti;
@Enumerated(EnumType.String);
private Partito partito;
```

PARTITO.JAVA

```
public enum Partito implements Serializable {
```

M5S,  
PD,  
FI,  
Lege,  
LEU,  
IV,  
Tecmer.

3

MINISTRI REMOTE.JAVA

@Remote

public interface MinistroRemote {

```
void addMinistro(ministro m);
void removeMinistro(ministro m);
void updateMinistro(ministro m);
List<ministro> get findAll();
ministro findbyID(int id);
List<ministro> findByName(String c);
List<ministro> findByPartito(Partito p);
ministro ministro findByNameMinistero(String m);
ministro findbyPortafogli(boolean true);
List<ministro> findbySottosegretario(sottosegretario);
List<ministro> findbyDipendenti(int d);
```

MINISTRIETB. JAVA

@Stateless

@LocalBean

public class MinistroETB implements MinistroRemoto {

@Inject

private EntityManager em;

@Override

public void addMinistro(Ministro m) {  
 em.persist(c);

}

@Override

public void removeMinistro(Ministro m) {  
 em.remove(em.merge(c));

}

@Override

public void updateMinistro(Ministro m) {  
 em.merge(c);

}

@Override ministro

public List&lt;Ministro&gt; findALL() {

TypedQuery<Ministro> query = em.createNamedQuery("Ministro.FIND-ALL", class)  
 return query.getResultList();

}

@Override

public Ministro findById(int id) {

TypedQuery<Ministro> query = em.createNamedQuery("Ministro.FIND-BY-ID", class)  
 query.setParameter(1, id)

return query.getSingleResult();

}

@Override

public List&lt;Ministro&gt; findByCognome()

(Strange)

TypedQuery&lt;Ministro&gt; query = em.createNamedQuery("Ministro.FIND-BY-COGNOME", class)

query.setParameter("cognome", c);

return query.getResultList();

}

@Override

public List&lt;Ministro&gt; findByPartito(Partito p) {

class

TypedQuery&lt;Ministro&gt; query = em.createNamedQuery("Ministro.FIND-BY-PARTITO", class)

query.setParameter("partito", p);

return query.getResultList();

}

@Override

public Ministeri findByMinistro (String m) {

    TypedQuery<Ministro> query = em.createNamedQuery("Ministro.FIND\_BY\_MINISTRO")

        .createQuery(m);

        query.setParameter("ministro", m);

        return query.getSingleResult();

}

@Override

public Ministeri findByPortafogli (boolean true) {

    TypedQuery<Ministro> query = em.createNamedQuery("Ministro.FIND\_BY\_PORTAFOLI")

        .createQuery(true);

        query.setParameter("portafogli", true);

        return query.getSingleResult();

}

@Override

public List<Ministro> findBySottosegretario () {

    TypedQuery<Ministro> query = em.createNamedQuery("Ministro.FIND\_BY\_SOTTOSEGRETAIO")

        .createQuery();

        return query.getResultList();

}

int d

@Override

public List<Ministro> findByDipendenti () {

    TypedQuery<Ministro> query = em.createNamedQuery("Ministro.FIND\_BY\_DIPENDENTI")

        .createQuery();

        query.setParameter("dipendenti", d);

        return query.getResultList();

)

ALFONSO ZAPPIA

(5)

## DATABASE PRODUCER.JAVA

```
public class DatabaseProducer {  
    @Produces  
    @PersistenceContext(unitName = "EnamePU")  
    private EntityManager em;  
}
```

## DATABASE POPULATOR.JAVA

```
@Singleton  
@Startup  
@DataSourceDefinition {  
    dsName = "org.apache.derby.jdbc.EmbeddedDataSource"  
    name = "EnameDS"  
    user = "user"  
    password = "password"  
    databaseName = "EnameDB"  
    properties = "connectionAttributes=true"  
}
```

```
public class DatabasePopulator {
```

```
    @Inject  
    private MinistroEJB ejb;
```

```
    private Ministro m1, m2, m3;  
    @PostConstruct
```

```
    private void populatedB() {
```

```
        m1 = new Ministro("Di maio", "Luigi", Partito.M5S, "Enteri", 3, true, 1200, 9);
```

```
        m2 = new Ministro("Lamorgese", "Luciana", Partito.Tecnici, "Untermi", 7, true, 8200, 4);
```

```
        m3 = new Ministro("Brunetta", "Renato", Partito.PD, "amministratore", 1, false, 0, 6);
```

```
        ejb.addMinistro(m1);
```

```
        ejb.addMinistro(m2);
```

```
        ejb.addMinistro(m3);
```

```
}
```

```
    @PreDestroy
```

```
    private void clearDB() {
```

```
        ejb.removeMinistro(m1);
```

```
        ejb.removeMinistro(m2);
```

```
        ejb.removeMinistro(m3);
```

MINISTRI STANDARD CLIENT - JAVA

\* nome EJB module  
+ nome pacchetto

```

public class MinistroStandardClient {
    @Inject
    private static MinistroRemote ejb;
    public static void main(String[] args) {
        Context context = new InitialContext();
        ejb = context.lookup("java:global/MinistroEJB/ministro");
        ejb = context.lookup("java:global/GovernoEJB/ministroEJB/ministro");
        MinistroRemote m;
        System.out.println("\n");
        System.out.println("Adesso stampo tutti i ministri con più di due sottosegretari");
        for (Ministro m : ejb.findAllBySottosegretario()) {
            System.out.println(m);
        }
        System.out.println("\n");
        System.out.println("Adesso ti stampo tutti i ministri del PD e del M5S");
        for (Ministro m : ejb.findAllByPartito(Partito.M5S)) {
            System.out.println(m);
        }
        System.out.println("\n");
        for (Ministro m : ejb.findAllByPartito(Partito.PD)) {
            System.out.println(m);
        }
        System.out.println("\n");
        System.out.println("Vedi un ministro che deve avere almeno quattro dipendenti");
        Scanner mnt = new Scanner(System.in);
        int dipendente;
        dipendente = mnt.nextInt();
        for (Ministro m : ejb.findAllByDipendente(dipendente)) {
            System.out.println(m);
        }
    }
}

```

MINISTRI JMS CLIENT

```

public class ministriJMSClient {
    Context context = new InitialContext();
    ConnectionFactory ejb = context.lookup("jms/Janee7/Topic");
    Destination destination = context.lookup("jms/Janee7/topic");
    MessageWrapper msg = new MessageWrapper();
    try {
        JMSEContext jmse = ejb.createContext();
        jmse.createProducer().send(destination, msg);
    }
}

```

MESSAGE WRAPPER.JAVA

```

public class MessageWrapper implements Serializable {
    int id;
    int dipendenti;
    int recoveryPoint;
    // getters e setters Partito partito;
    // costruttore
    // toString
}

```

MINISTRI MDB.JAVA

@MessageDriven(mappedName = "jms/Janee7/topic")

public class ministriMDB implements MessageListener

@Inject

Event<ministri> updateDipendenti

Event<ministri> recoveryPoint(Event

Event<ministri> moltoFogliato(Event

private ministri ejb

public void onMessage(Message msg)

try {

MessageWrapper msgContent = msg.getBody(MessageWrapper.class);  
int id = msgContent.getId();

Ministri m = ejb.findById(id)

~~int dipendenti = msgContent.getDipendenti()~~

~~Ministri m = ejb.findByDipendenti(dipendenti)~~

~~m.setnDipendenti(dipendenti)~~

~~updateDipendenti(m)~~

## ALFONSO ZAPPIA

(8)

```
int interesse = msgContent.getIntere... getRecoveryFound()
m.setRecoveryFound(intere...  
updateDati.fire()
```

```
} (msgContent.getInterestRecoveryFound >= 7 && PortafoglioEvent {  
recoveryFoundEvent.fire();
```

```
} (msgContent.getInterestRecoveryFound >= 5 && msgContent.getPortafogli = true)  
portafogliEvent.fire()
```

```
} catch (JMSException e)  
System.out.println(e)
```

## UPDATEDATI EVENT.JAVA

```
public class UpdateDatiEvent {
    public void notify(@Observes Ministro m) {
        System.out.println(m.getId() "è stato modificato")
    }
}
```

## RECOVERY FOUND EVENT.JAVA

```
public class RecoveryFoundEvent {
    public void notify(@Observes Ministro m) {
        System.out.println(m.getId(), "Ecco un europeo")
    }
}
```

## PORATAFOGLI EVENT. JAVA

```
public class PortafoglioEvent {
    public void notify(@Observes Ministro m) {
        System.out.println(m.getId(), "E' ormai possibile");
    }
}
```

Normalmente, quando parliamo di comunicazione tra componenti enterprise, ci riferiamo soltanto su comunicazioni sincrone, ossia che una classe me chiama un'altra. Le comunicazioni asincrone invece, avvengono attraverso un API ben definito, chiamato JMS, acronimo di Java Message Service. Per scambiarsi messaggi asincroni, ponetevi mette davanti un nuovo strato software, detto MOM, acronimo di Message-oriented Middleware, in cui vengono prodotti e consumati messaggi. Quando viene inviato un messaggio, c'è chi si occupa di memorizzarlo e inviarlo, ed esso si chiama Producer o anche Broker. Il mittente del messaggio è detto Producer e il percorso in cui viene inviato il messaggio è detto destinazione. Il destinatario è chiamato Consumer. Esistono due modelli principali di messaging:

- Modello Point-to-Point (P2P), in cui la destinazione è una coda in cui si accumulano i messaggi.
- Modello Publish-Subscribe (pub-sub) in cui la destinazione è chiamato topic, in cui un client "pubblica" un topic e altri consumeri possono "iscriversi".

Normalmente, in un normale container EJB possiamo definire dei consumeri di messaggi, chiamati Message-Driven Beans, mentre prima bisogna definire un client di JMS in cui è inizializzata una ConnectionFactory.