

MINISTRI.TAVA

② Entity

③ Named Queries {E

- ④ NamedQuery (name = FIND-ALL, query = "SELECT m FROM ministri m");
- ④ NamedQuery (name = FIND-BY-ID, query = "SELECT m FROM ministri m WHERE m.id = ?1");
- ④ NamedQuery (name = FIND-BY-COGNOME, query = "SELECT m FROM ministri m WHERE m.cognome = cognome");
- ④ NamedQuery (name = FIND-BY-PARTITO, query = "SELECT m FROM ministri m WHERE m.partito = partito");
- ④ NamedQuery (name = FIND-BY-MINISTERO, query = "SELECT m FROM ministri m WHERE m.ministero = ministero");
- ④ NamedQuery (name = FIND-BY-PORTAFOGLI, query = "SELECT m FROM ministri m WHERE m.portafogli = true");
- ④ NamedQuery (name = FIND-BY-SOTTOSEGRETARIO, query = "SELECT m FROM ministri m WHERE m.sottosegretario > 2");
- ④ NamedQuery (name = FIND-BY-DIPENDENTI, query = "SELECT m FROM ministri m WHERE m.dipendenti > dipendenti");

}

public class Ministri implements Serializable {

```

public static final String FIND-ALL = "Ministri.findAll";
public static final String FIND-BY-ID = "Ministri.findById";
public static final String FIND-BY-COGNOME = "Ministri.findByCognome";
public static final String FIND-BY-PARTITO = "Ministri.findByPartito";
public static final String FIND-BY-MINISTERO = "Ministri.findByMinistero";
public static final String FIND-BY-PORTAFOGLI = "Ministri.findByPortafogli";
public static final String FIND-BY-SOTTOSEGRETARIO = "Ministri.findBySottosegretario";
public static final String FIND-BY-DIPENDENTI = "Ministri.findByDipendenti";

```

③ ID

③ Generated Value

int id;

③ not null

```

private String cognome;
private String nome;
private String ministero;
private int mSottosegretario;
private int recoveryFound;
private boolean portafogli;

```

```

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

```

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

PARTITO.JAVA

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

```
    MSS,
    PD,
    FI,
    Lega,
    LEU,
    IV,
    Tecni.
```

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> findByCognome(String c);
```

```
    List<Ministro> findByPartito(Partito p);
```

```
find Ministro findByMinistero(String m);
```

```
    Ministro findByPortafogli(boolean true);
```

```
    List<Ministro> findBySottosegretario(boolean true);
```

```
    List<Ministro> findByDipendenti(int d);
```

MINISTRI.ETB.TAVA

@Stateless

@LocalBean

public class Ministro.ETB implements MinistroRemote {

@Inject

private EntityManager em;

@Override

```
public void add Ministro (Ministro m) {
    em.persist(c);
}
```

@Override

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

@Override

```
public void update Ministro (Ministro m) {
    em.merge(c);
}
```

@Override Ministro

public List<~~Ministro~~> 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

(Strange)

public ~~Ministro~~ List<Ministro> findByCognome {

```
TypedQuery<Ministro> query = em.createNamedQuery(Ministro.FIND_BY_COGNOME, class)
query.setParameter("cognome", c);
return query.getResultList();
}
```

@Override

public List<Ministro> findByPartito (Partito p) {

```
TypedQuery<Ministro> query = em.createNamedQuery(Ministro.FIND_BY_PARTITO, class)
query.setParameter("partito", p);
return query.getResultList();
}
```

@Override

```
public Ministri findByMinistero(String m) {  
    TypedQuery<Ministri> query = em.createNamedQuery(Ministri.FIND_BY_MINISTERO  
        .class);  
    query.setParameter("ministero", m);  
    return query.getSingleResult();  
}
```

@Override

```
public Ministri findByPortafoglio(boolean true) {  
    TypedQuery<Ministri> query = em.createNamedQuery(Ministri.FIND_BY_PORTAFOGLIO  
        .class);  
    query.setParameter("portafoglio", true);  
    return query.getSingleResult();  
}
```

@Override

```
public List<Ministri> findBySottosegretario() {  
    TypedQuery<Ministri> query = em.createNamedQuery(Ministri.FIND_BY_SOTTOSEGRETARIO  
        .class);  
    return query.getResultList();  
}
```

int d

@Override

```
public List<Ministri> findByDipendenti X {  
    TypedQuery<Ministri> query = em.createNamedQuery(Ministri.FIND_BY_DIPENDENTI  
        .class);  
    query.setParameter("dipendenti", d);  
    return query.getResultList();  
}
```

DATABASE PRODUCER.TAVA

```

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

```

DATABASE POPULATOR.TAVA

```

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

public class DatabasePopulator {
    @Inject
    private MinistriEJB ejb;
    private Ministri m1, m2, m3;

    @PostConstruct
    private void populateDB() {
        m1 = new Ministro("Di Maio", "Luigi", Partito.M5S, "Estero", 3, true, 1200, 9);
        m2 = new Ministro("Lamorgese", "Luciana", Partito.Tecnica, "Interno", 7, true, 800, 4);
        m3 = new Ministro("Brunetta", "Renato", Partito.PD, "Amministrazione", 1, false, 0, 4);

        ejb.addMinistro(m1);
        ejb.addMinistro(m2);
        ejb.addMinistro(m3);
    }

    @PreDestroy
    private void cleanDB() {
        ejb.removeMinistro(m1);
        ejb.removeMinistro(m2);
        ejb.removeMinistro(m3);
    }
}

```

MINISTRI STANDARD CLIENT - JAVA

* nome EJB Module
+ nome pacchetto

```
public class MinistriStandardClient {
```

```
    @Inject
```

```
    private static MinistriRemote ejb;
```

```
    public static void main(String[] args) {
```

```
        Context context = new InitialContext();
```

```
        ejb = context.lookup("java:global/MinistriEJB/");
```

```
        ejb = context.lookup("java:global/GovernoEJB/MinistriEJB!Ministri  
        * MinistriRemote");
```

```
        System.out.println("\n");
```

```
        System.out.println("Adesso stampo tutti i ministri con più di due sottosegretari");
```

```
        for (Ministri m : ejb.findBySottosegretario()) {
```

```
            System.out.println(m);
```

```
        }
```

```
        System.out.println("\n");
```

```
        System.out.println("Adesso ti stampo tutti i ministri del PD e del HSS");
```

```
        for (Ministri m : ejb.findByPartito(Partito.HSS)) {
```

```
            System.out.println(m);
```

```
        }
```

```
        for (Ministri m : ejb.findByPartito(Partito.PD)) {
```

```
            System.out.println(m);
```

```
        }
```

```
        System.out.println("\n");
```

```
        System.out.println("Vedi un ministro che deve avere almeno quanti  
        dipendente?");
```

```
        Scanner mnt = new Scanner(System.in);
```

```
        int  
3255 dipendente;
```

```
        dipendente = mnt.nextInt();
```

```
        for (Ministro m : ejb.findByDipendente(dipendente)) {  
            System.out.println(m);
```

MINISTRI JMS CLIENT

```

public class MinistriJMSClient {
    Context context = new InitialContext();
    ConnectionFactory cf = context.lookup("jms/jmscf/ConnectionFactory");
    Destination destination = context.lookup("jms/jmscf/topic");
    MessageWrapper msg = new MessageWrapper();
    try {
        JMSContext jmsc = cf.createContext();
        jmsc.createProducer().send(topic, msg);
    }
}

```

MESSAGE WRAPPER.JAVA

```

public class MessageWrapper implements Serializable {
    int id;
    int dependenti;
    int recoverySound;
    // getters & setters    Partito partito;
    // costruttore
    // toString
}

```

MINISTRI MDB.JAVA

① MessageDriven(mappedName = "jms/jmscf/topic")

```

public class MinistriMDB implements MessageListener {
    @Inject
    EntityManager em;
    Event<Ministri> updateDataEvent;
    Event<Ministri> recoverySound(Event<Ministri> m) {
        m.setRecoverySound(1);
    }
    Event<Ministri> moltoFoglia(Event<Ministri> m) {
        m.setMoltoFoglia(1);
    }
    private Ministri m;
    public void onMessage(Message msg) {
        try {
            MessageWrapper msgContent = msg.getBody(MessageWrapper.class);
            int id = msgContent.getId();
            Ministri m = em.findById(id);
            m.set int dependenti = msgContent.getDependenti();
            m.set m = em.findById(dependenti);
            m.setMoltoFoglia(dependenti);
            updateDataEvent

```

```

int interese = msgContent.setRecoveryFound getRecoveryFound()
m.setRecoveryFound(interese)
updateData.fire()

```

```

1) (msgContent.getIntereseRecoveryFound >= 7 && Partito.Lega {
    recoveryFoundEvent.fire();
2) (msgContent.getIntereseRecoveryFound > 5 && msgContent.getPortafogli == true)
    portafogliEvent.fire();
} catch (Throwable e)
    System.out.println(e)

```

UPDATEDATI EVENT. JAVA

```

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

```



RECOVERYFOUNDEVENT. JAVA

```

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

```

PORTAFOGLI EVENT. JAVA

```

public class PortafogliEvent {
    public void notify(@Observes Ministro m) {
        System.out.println(m.getId() + "Ecco un portafoglio");
    }
}

```

Normalmente, quando parliamo di comunicazione tra componenti Enterprise, ci riferiamo soltanto ai comunicazioni sincrone, dove che una classe ne chiama un'altra. Le comunicazioni asincrone invece, avvengono attraverso un API ben definito, chiamato JMS, acronimo di Java Messaging Service. Per scambiarsi messaggi asincroni, JavaEE 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 messaggistica,

- 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 chiamata topic e in cui un client "pubblica" un topic e altri consumatori possono "iscrivere".

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