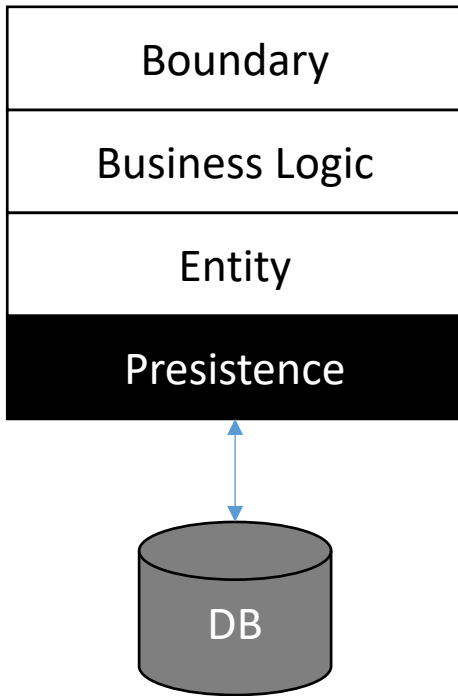


Hibernate Tutorial

Esercitazione del Corso di Progettazione e Sviluppo Sistemi Software
(Prof. Fasolino)

A cura di: ing. Domenico Amalfitano

Introduzione (1)



- ORM è acronimo di Object-Relational Mapping
- Trasporta concetti del mondo dei database relazionali, nel mondo degli oggetti
- Hibernate è diventato il framework ORM più utilizzato per lo sviluppo Java

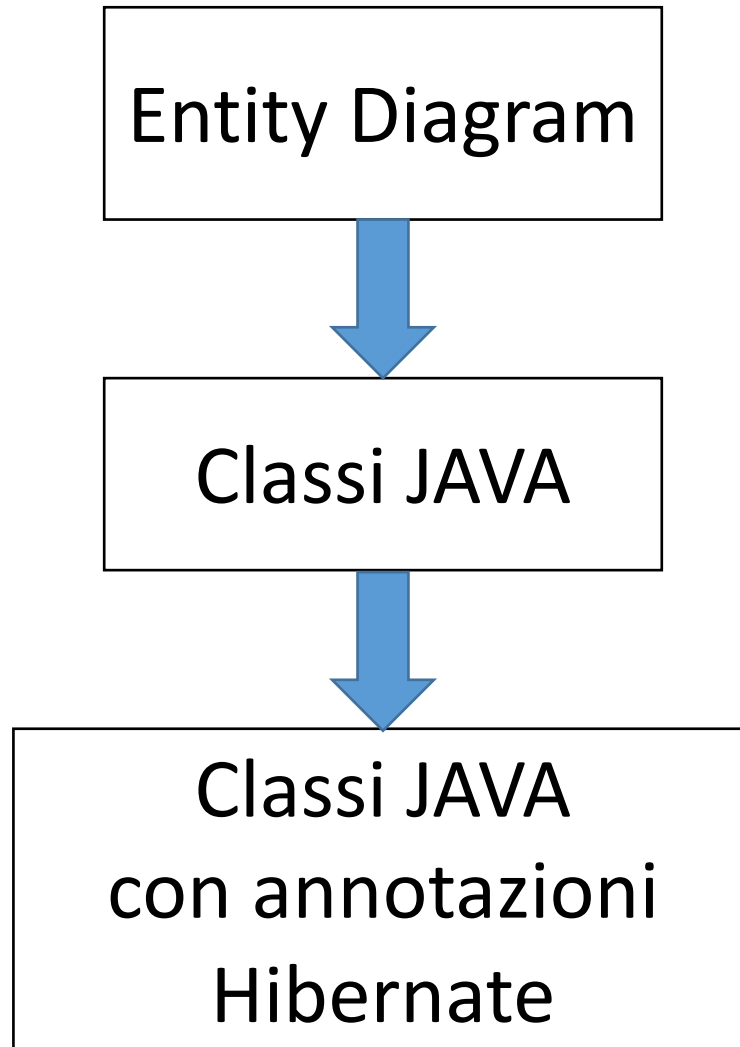
Introduzione (2)

- Semplifica la scrittura del codice di interazione con il Database
- Permette di collegare entità dell'applicazione a tabelle del database
- Elimina la complessità legata all'utilizzo di JDBC
- Gestisce automaticamente le connessioni al database e la gestione delle eccezioni

Introduzione (3)

- L'associazione tra oggetti e tabelle del database è mediato da un mapping
- Il mapping può essere realizzato
 - Mediante file xml
 - Mediante annotazioni di classi, metodi e attributi
- Un ulteriore file di configurazione è necessario per configurare la connessione al database: hibernate.cfg.xml

Da Entity a Database (1)



Da Entity a Database (2)

Impiegato
-matricola : Integer -nome : String -cognome : String
+getMatricola() : Integer +setMatricola(matricola : Integer) : void +getNome() : String +setNome(nome : String) : void +getCognome() : String +setCognome(cognome : String) : void +Impiegato() +Impiegato(matricola : Integer)

Da Entity a Database (3)

Impiegato
-matricola : Integer -nome : String -cognome : String
+getMatricola() : Integer +setMatricola(matricola : Integer) : void +getNome() : String +setNome(nome : String) : void +getCognome() : String +setCognome(cognome : String) : void +Impiegato() +Impiegato(matricola : Integer)



```
public class Impiegato {  
    Integer matricola;  
    String nome;  
    String cognome;  
}
```

Da Entity a Database (4)

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;


public class Impiegato {
    Integer matricola;
    String nome;
    String cognome;
}

@Entity
public class Impiegato {

    @Id
    Integer matricola;

    @Column
    String nome;

    @Column
    String cognome;
}
```



Da Entity a Database (5)

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;
```

```
@Entity
public class Impiegato {
```

```
    @Id
    Integer matricola;
```

```
    @Column
    String nome;
```

```
    @Column
    String cognome;
```



```
create table Impiegato (
    matricola integer not null,
    cognome varchar(255),
    nome varchar(255),
    primary key (matricola)
)
```

**Questa trasformazione viene fatta da
Hibernate se opportunamente configurato!**

Da Entity a Database (6)

- L'annotazione **@Entity** comunica ad Hibernate che la classe Impiegato corrisponde ad una tabella del database
 - Se vogliamo specificare il nome della tabella, dobbiamo annotare la classe anche con **@Table(name = "nome_tabella")**
- L'annotazione **@Id** di una variabile membro la denota come chiave primaria
 - Se si vuole specificare un campo AUTO_INCREMENT bisogna annotare ulteriormente la variabile con **@GeneratedValue**
 - Se si vuole specificare un nome diverso per la chiave primaria bisogna aggiungere l'annotazione **@Column**
- L'annotazione **@Column** di una variabile membro di una classe la denota come campo del database
 - Se una variabile membro non è annotata non viene considerata da Hibernate
 - È possibile specificare il nome della colonna della tabella collegata **@Column(name = "nome_colonna")**
 - È possibile specificare gli attributi della colonna **@Column(name = "nome_colonna", nullable=false, unique=true)**
- L'entità deve essere inoltre dichiarata nell'apposita sezione del file di configurazione **hibernate.cfg.xml**

hibernate.cfg.xml (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Connessione al database -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost:3307/accesso_camere</property>

        <!-- Credenziali -->
        <property name="hibernate.connection.username">user</property>
        <property name="connection.password">password</property>
```

hibernate.cfg.xml (2)

```
<!-- JDBC connection pool (use the built-in) -->
```

```
  <property name="connection.pool_size">1</property>
```

```
<!-- SQL dialect -->
```

```
  <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

```
<!-- Enable Hibernate's automatic session context management -->
```

```
  <property name="current_session_context_class">thread</property>
```

```
<!-- Disable the second-level cache -->
```

```
  <property  
name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
```

```
  <property name="show_sql">true</property>
```

```
  <property name="hibernate.hbm2ddl.auto">create</property>
```

hibernate.cfg.xml (3)

```
<!-- Entity -->
```

```
    <mapping class="it.ufficio.Impiegato" />
```

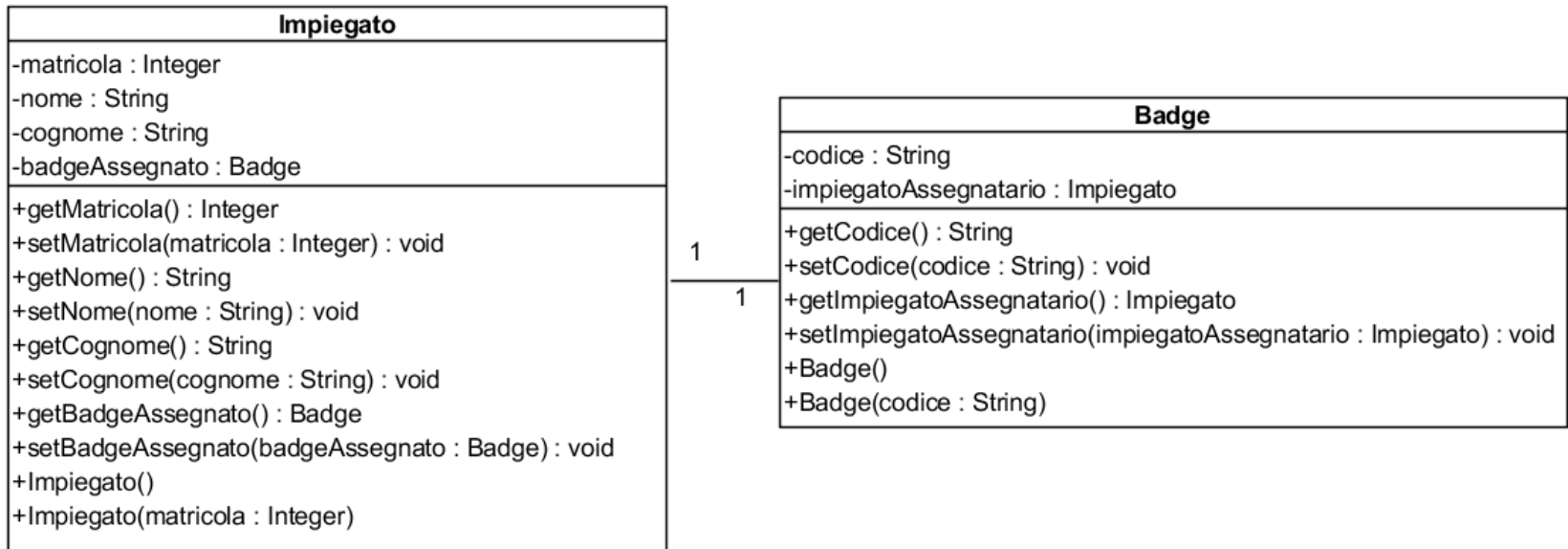
```
</session-factory>
```

```
</hibernate-configuration>
```

Librerie da Includere nel progetto Eclipse

- Scaricare Hibernate da <http://goo.gl/Nu8wtC>
- Le librerie necessarie sono nella sottodirectory "required":
 - antlr-2.7.7.jar
 - dom4j-1.6.1.jar
 - hibernate-commons-annotations-4.0.4.Final.jar
 - hibernate-core-4.3.5.Final.jar
 - hibernate-jpa-2.1-api-1.0.0.Final.jar
 - jandex-1.1.0.Final.jar
 - javassist-3.18.1-GA.jar
 - jboss-logging-3.1.3.GA.jar
 - jboss-logging-annotations-1.2.0.Beta1.jar
 - jboss-transaction-api_1.2_spec-1.0.0.Final.jar
 - mysql-connector-java-5.1.28.jar (<http://goo.gl/yn487w>)
- NOTA: i numeri di versione possono cambiare

Relazione uno a uno (1)



Relazione uno a uno (2)

```
@Entity
public class Impiegato {

    @Id
    Integer matricola;

    @Column
    String nome;

    @Column
    String cognome;

    @OneToOne
    @JoinColumn(name = "codice_badge")
    Badge badgeAssegnato;
}
```

Tipo di relazione

Colonna relativa alla relazione

Relazione uno a uno (3)

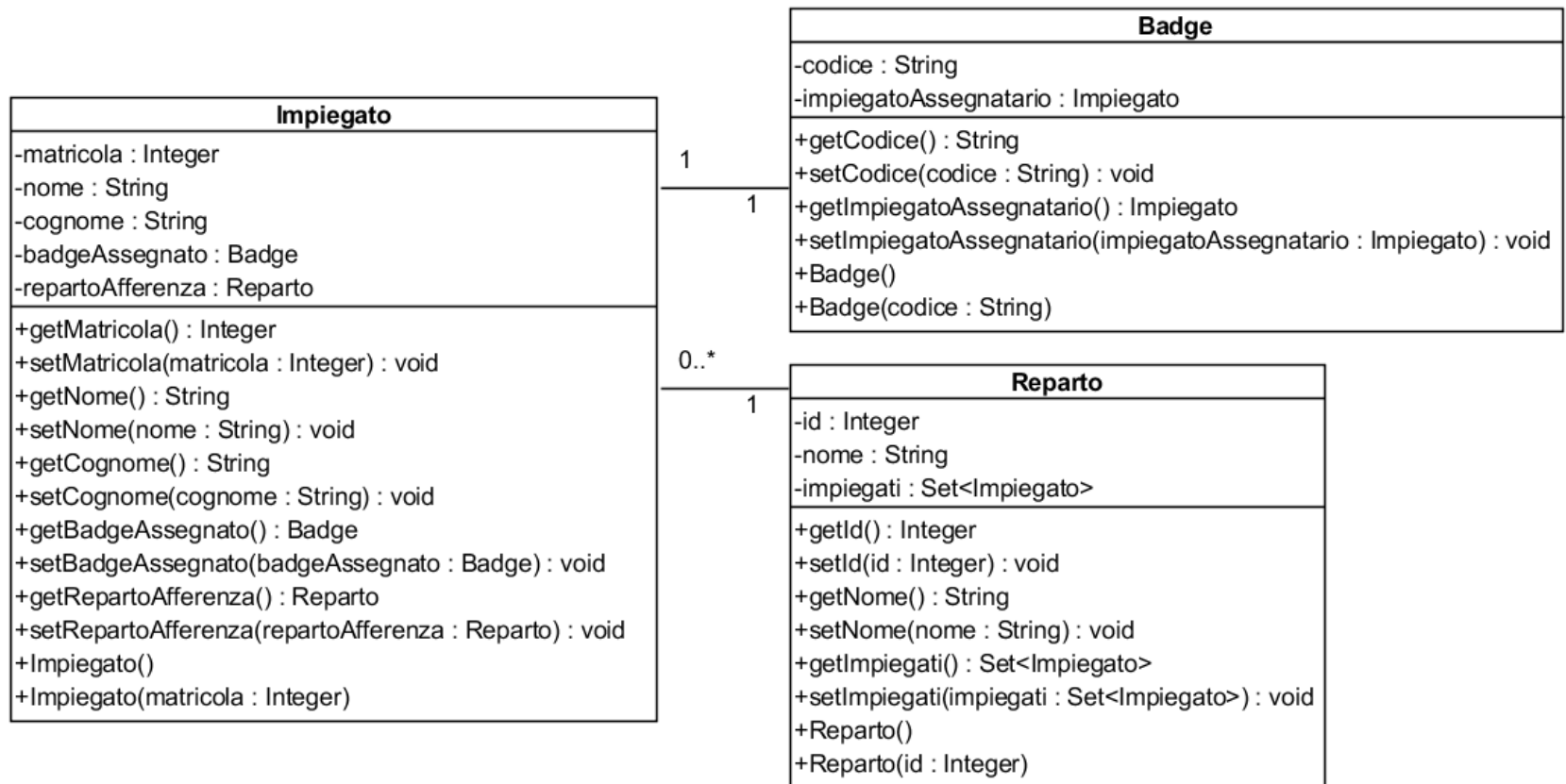
```
create table Badge (  
    codice varchar(255) not null,  
    matricola_impiegato integer,  
    primary key (codice)  
)
```

```
create table Impiegato (  
    matricola integer not null,  
    cognome varchar(255),  
    nome varchar(255),  
    reparto varchar(255),  
    codice_badge varchar(255),  
    primary key (matricola)  
)
```

```
alter table Badge add constraint FK_713uc29lqh8cc0xb6a0c1582a foreign key  
(matricola_impiegato) references Impiegato (matricola)
```

```
alter table Impiegato add constraint FK_novqlfxq3wptvde05kru11dg4 foreign key (codice_badge)  
references Badge (codice)
```

Relazione uno a molti (1)



Relazione uno a molti (2)

```
@Entity  
public class Impiegato {
```

```
    @Id  
    Integer matricola;
```

```
    @Column  
    String nome;
```

```
    @Column  
    String cognome;
```

```
    @OneToOne  
    @JoinColumn(name = "codice_badge")  
    Badge badgeAssegnato;
```

```
    @ManyToOne  
    @JoinColumn(name = "reparto_afferenza")  
    Reparto repartoAfferenza;
```

```
@Entity  
public class Reparto {
```

```
    @Id  
    Integer id;
```

```
    @Column  
    String nome;
```

```
    @OneToMany  
    @JoinColumn(name="reparto_afferenza")  
    Set<Impiegato> impiegati;
```

Le relazioni sono
simmetriche



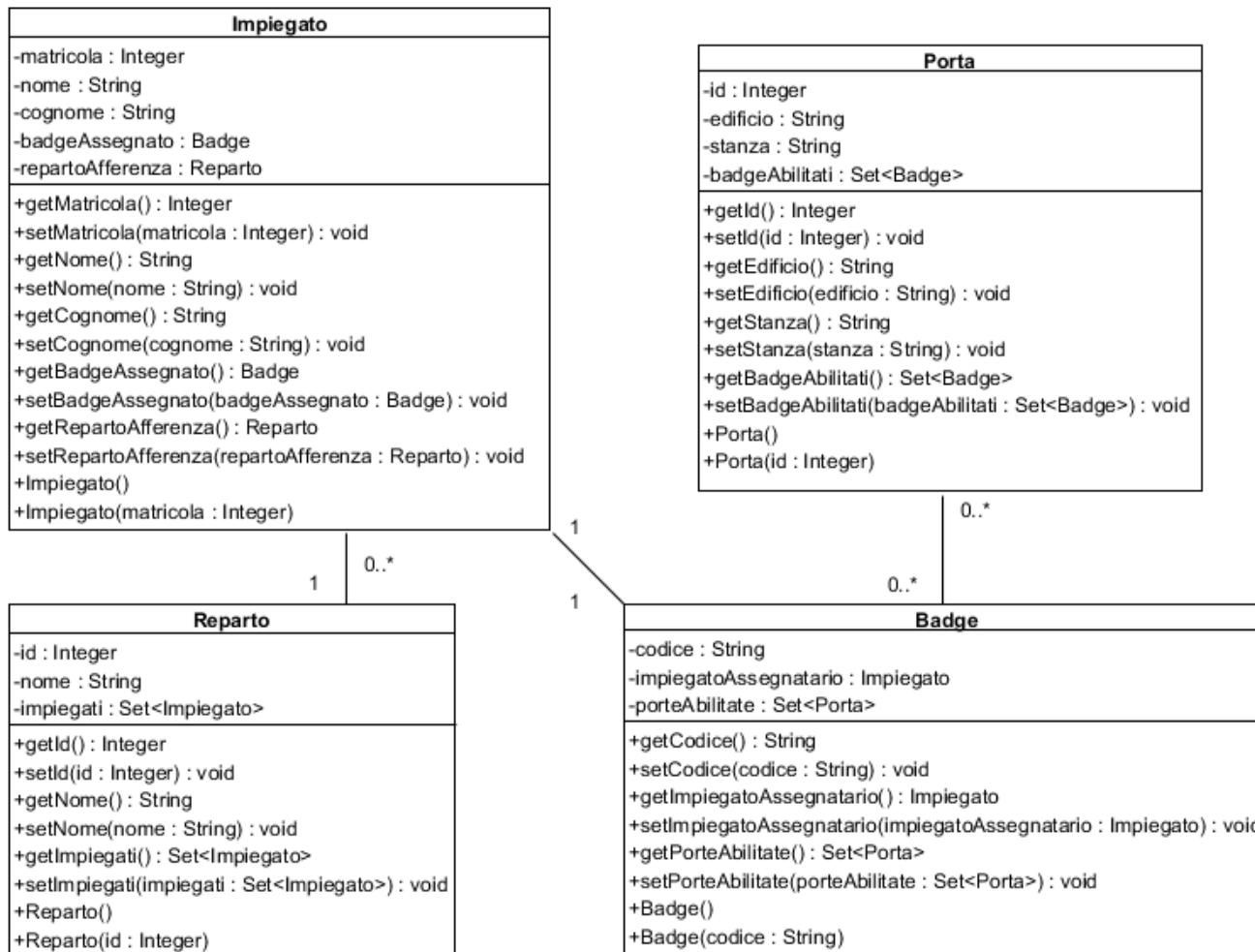
Relazione uno a molti (3)

```
create table Impiegato (  
    matricola integer not null,  
    cognome varchar(255),  
    nome varchar(255),  
    codice_badge varchar(255),  
    reparto_afferenza integer,  
    primary key (matricola)  
)
```

```
create table Reparto (  
    id integer not null,  
    nome varchar(255),  
    primary key (id)  
)
```

```
alter table Impiegato add constraint FK_1uef2jae1v187an9mubj7o1q0 foreign key  
(reparto_afferenza) references Reparto (id)
```

Relazione molti a molti (1)



Relazione molti a molti (2)

```
@Entity
public class Badge {
```

```
    @Id
    String codice;
```

```
    @OneToOne
    @JoinColumn(name = "matricola_impiegato")
    Impiegato impiegatoAssegnatario;
```

Tipo di
relazione

Nome della tabella
di relazione

```
    @ManyToMany(cascade = {CascadeType.ALL})
    @JoinTable(name="BADGE_PORTA",
        joinColumns={@JoinColumn(name="id_badge")},
        inverseJoinColumns={@JoinColumn(name="id_porta")})
    Set<Porta> porteAbilitate;
```

Variabile membro associata
alla relazione

Nome delle colonne nel database

Relazione molti a molti (3)

```
@Entity  
public class Porta {
```

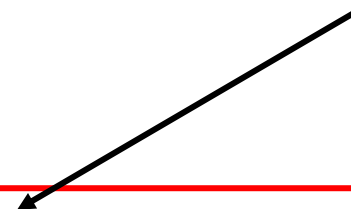
```
    @Id  
    Integer id;
```

```
    @Column  
    String edificio;
```

```
    @Column  
    String stanza;
```

```
    @ManyToMany(mappedBy="porteAbilitate")  
    Set<Badge> badgeAbilitati;
```

Riferita alla variabile membro
dichiarata nella classe Badge



Relazione molti a molti (4)

```
create table BADGE_PORTA (  
    id_badge varchar(255) not null,  
    id_porta integer not null,  
    primary key (id_badge, id_porta))  
create table Badge (  
    codice varchar(255) not null,  
    matricola_impiegato integer,  
    primary key (codice))  
create table Porta (  
    id integer not null,  
    edificio varchar(255),  
    stanza varchar(255),  
    primary key (id))  
alter table BADGE_PORTA add constraint FK_tb7p15h9e6b32kni46e49baay  
foreign key (id_porta) references Porta (id)  
alter table BADGE_PORTA add constraint FK_6s6m65v92bt95d4lmgi9hikey  
foreign key (id_badge) references Badge (codice)
```


Session (1)

- Per interagire con il Database tramite gli oggetti, Hibernate mette a disposizione l'oggetto Session
- Permette di effettuare tutte le operazioni CRUD
- Per ottenere un'istanza di Session si deve creare un'istanza di SessionFactory
- Per ottenere questa istanza si può utilizzare la classe HibernateUtil.java fornita con il codice d'esempio
- Ogni volta che si vuole effettuare un'operazione si deve aprire e chiudere una sessione ed una transazione
- È possibile realizzare più transazioni all'interno della stessa sessione

Session (2)

```
SessionFactory sf = HibernateUtil.getSessionFactory();  
Session session = sf.openSession();  
session.beginTransaction();
```

```
//utilizzo i metodi della Session  
//...
```

```
//chiudo la transazione e la sessione  
session.getTransaction().commit();  
session.close();
```

Inserimento

```
Impiegato impiegato1 = new Impiegato();  
impiegato1.setNome("Mario");  
impiegato1.setCognome("Rossi");  
//...
```

```
SessionFactory sf = HibernateUtil.getSessionFactory();  
Session session = sf.openSession();  
session.beginTransaction();
```

```
session.save(impiegato1);
```

```
//chiudo la transazione e la sessione  
session.getTransaction().commit();  
session.close();
```

Ricerca per id

```
SessionFactory sf = HibernateUtil.getSessionFactory();  
Session session = sf.openSession();  
session.beginTransaction();
```

```
Impiegato impiegato2 = (Impiegato )session.get(Impiegato.class,  
"matricola_001");
```

```
//chiudo la transazione e la sessione  
session.getTransaction().commit();  
session.close();
```

Aggiornamento

```
//cambio il nome all'impiegato  
impiegato2.setNome("Stefano");
```

```
SessionFactory sf = HibernateUtil.getSessionFactory();  
Session session = sf.openSession();  
session.beginTransaction();
```

```
session.update(impiegato2);
```

```
//chiudo la transazione e la sessione  
session.getTransaction().commit();  
session.close();
```

Cancellazione

```
SessionFactory sf = HibernateUtil.getSessionFactory();  
Session session = sf.openSession();  
session.beginTransaction();
```

```
session.delete(impiegato2);
```

```
//chiudo la transazione e la sessione  
session.getTransaction().commit();  
session.close();
```

Query (1)

- Per effettuare le Query in Hibernate si può utilizzare HQL (Hybernate Query Language)
- HQL è molto simile ad SQL
- A differenza di SQL, una query HQL va implementata considerando il modello ad oggetti
- Ad esempio, se vogliamo ricercare l'Impiegato a partire dal codice Badge
 - in SQL scriviamo:
SELECT * FROM Impiegato WHERE codice_badge="000000"
 - in HQL
FROM Impiegato WHERE Impiegato.badgeAssociato="000000"
- In questo caso "Impiegato" non si riferisca alla tabella, ma alla classe (in questo caso nome della classe e della tabella coincidono)

Query (2)

//cerca gli impiegati il cui nome è "Massimo"

```
SessionFactory sf = HibernateUtil.getSessionFactory();  
Session session = sf.openSession();  
session.beginTransaction();
```

```
Query query = session.createQuery("from Impiegato as i where i.nome =  
:nome");  
query.setParameter(" nome ", "Massimo");  
ArrayList<Impiegato> result = (ArrayList<Impiegato>)query.list();
```

```
//chiudo la transazione e la sessione  
session.getTransaction().commit();  
session.close();
```