



Corso di Laurea in Informatica
III Anno Triennale
Programmazione Distribuita



Java Persistence API

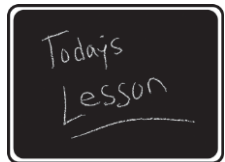
Delfina Malandrino

dmalandrino@unisa.it

<http://www.unisa.it/docenti/delfinamalandrino>

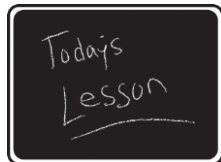
Organizzazione della lezione

- Introduzione
- Le Entità
 - Definizione
 - Anatomia
 - Queries
- Object-Relational Mapping (ORM)
 - Entity Manager
 - La Persistenza (Persistence Unit)
 - Ciclo di vita delle Entità
- JPA Specification
- Putting It All Together
- Managing Persistent Objects
- Conclusioni



Organizzazione della lezione

- Introduzione
- Le Entità
 - Definizione
 - Anatomia
 - Queries
- Object-Relational Mapping (ORM)
 - Entity Manager
 - La Persistenza (Persistence Unit)
 - Ciclo di vita delle Entità
- JPA Specification
- Putting It All Together
- Managing Persistent Objects
- Conclusioni

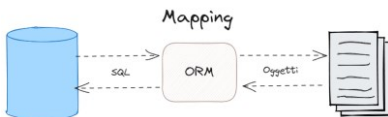


Introduzione

- Le applicazioni sono fatte di logica, interazioni con altri sistemi, user interfaces, etc.
 - ma anche di dati
- I DB assicurano la persistenza dei dati
- I dati sono di solito memorizzati (in DB) reperiti e analizzati
 - tabelle, righe, colonne, chiavi primarie, indici, join, ...
- ... **vocabolario completamente diverso da quello di linguaggi di programmazione OO!**
 - classi, oggetti, variabili, riferimenti, metodi attributi, ...

Introduzione

- La vera grande differenza, la persistenza!
 - quando il Garbage Collector decide di eliminare un elemento dalla memoria, è andato per sempre!
 - I database permettono invece la loro persistenza in maniera permanente
- L'Object-Relational Mapping (ORM)** mette insieme i due mondi

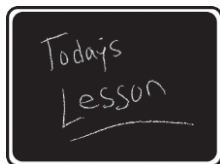


La Specifiche di JPA

- Creata con Java EE 5 per **portare insieme il modello OO e i DB**
- JPA è una astrazione su JDBC che lo rende indipendente da SQL
- Contenuto nel package `javax.persistence`
- Definisce l'Object-Relational Mapping
- Componente fondamentale è l'Entity Manager che fa operazioni CRUD (**create, read, update and delete**) su DB
- Linguaggio per query (Java Persistence Query Language)
- Meccanismo per le transazioni con Java Transactions API
- Callback e listener per fare reagire la logica di business agli eventi di un oggetto persistente

Organizzazione della lezione

- Introduzione
- Le Entità
 - Definizione
 - Anatomia
 - Queries
- Object-Relational Mapping (ORM)
 - Entity Manager
 - La Persistenza (Persistence Unit)
 - Ciclo di vita delle Entità
- JPA Specification
- Putting It All Together
- Managing Persistent Objects
- Conclusioni



Entità: queste sconosciute

- **Un oggetto** è una istanza di una classe che risiede in memoria e pertanto è di vita breve
- **Una entità** è un oggetto che vive anche persistentemente in un database
- Obiettivo: rendere le entità persistenti, crearle, rimuoverle, fare interrogazioni
 - Possibile usare Java Persistence Query Language (JPQL) per gestire le entità
- Nel modello JPA, **una entità è un POJO**, dichiarata, istanziata ed usata come altre classi Java

Un primo esempio di POJO con JPA

```
@Entity  
public class Book {  
    @Id @GeneratedValue  
    private Long id;  
    private String title;  
    private Float price;  
    private String description;  
    private String isbn;  
    private Integer nbOfPage;  
    private Boolean illustrations;  
  
    public Book() {  
    }  
  
    //Getters, setters  
}
```

Annotazione per definire l'entità

Un primo esempio di POJO con JPA

```
@Entity  
public class Book {  
    @Id @GeneratedValue  
    private Long id;  
    private String title;  
    private Float price;  
    private String description;  
    private String isbn;  
    private Integer nbOfPage;  
    private Boolean illustrations;  
  
    public Book() {  
    }  
  
    //Getters, setters  
}
```

Annotazione per definire l'entità
Classe

Un primo esempio di POJO con JPA

```
@Entity
public class Book {
    @Id @GeneratedValue
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;

    public Book() {
    }

    //Getters, setters
}
```

Annotazione per definire l'entità
Classe

La chiave primaria, particolare

Un primo esempio di POJO con JPA

```
@Entity
public class Book {
    @Id @GeneratedValue
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;

    public Book() {
    }

    //Getters, setters
}
```

Annotazione per definire l'entità
Classe

La chiave primaria, particolare

Una serie di altri attributi

Un primo esempio di POJO con JPA

```
@Entity
public class Book {
    @Id @GeneratedValue
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;

    public Book() { }
}

//Getters, setters
```

Annotazione per definire l'entità
Classe

La chiave primaria, particolare

Una serie di altri attributi

Un costruttore di default

Un primo esempio di POJO con JPA

```
@Entity
public class Book {
    @Id @GeneratedValue
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;

    public Book() { }

    //Getters, setters
}
```

Annotazione per definire l'entità
Classe

La chiave primaria, particolare

Una serie di altri attributi

Un costruttore di default

E altri metodi vari

- Come fa questa classe a essere mappata in una tabella?
 - Grazie alle annotazioni!!!

Anatomia di una Entità: definizione

- Cosa è una entità?
 - Una classe annotata con `@javax.persistence.Entity`
 - `@javax.persistence.Entity.id` definisce la ID univoca dell'oggetto
- In questo modo il persistence provider la considera come una classe persistente e non come un POJO

Anatomia di una Entità: regole

- Regole per essere una entità:
 - Annotata con `@javax.persistence.Entity`
 - `@javax.persistence.Id` per la chiave primaria
 - Deve esserci un costruttore senza argomenti, che sia `public` o `protected`
 - Possono esserci altri costruttori
 - Una entity class deve essere una top-level class. Enum o interfacce non possono essere designate come entità
 - Non deve essere final e nessun metodo/attributo persistente deve essere final
- Se deve essere passata per valore (come in un metodo remoto) deve implementare `Serializable`

Il ruolo dei metadati in ORM

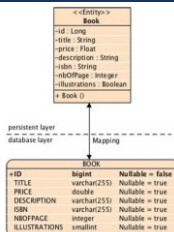
- Il principio di ORM è quello di delegare a tools esterni o frameworks (nel nostro caso JPA) il compito di creare una corrispondenza fra oggetti e tabelle
- Come fa JPA a mappare oggetti in un database?
 - Attraverso i metadati!!!
- I metadati possono essere scritti in due formati differenti
 - Annotazioni
 - Descrittori XML

Il ruolo dei metadati in ORM

- Java EE 5 ha introdotto l'idea della:
 - **Configuration by exception**: importante tecnica in cui le regole di default vengono applicate dal container, se non altrimenti specificate
 - Anche conosciuta come convention over configuration
- Fornire una configurazione (personalizzazione) rappresenta una eccezione alla regola
- Con questa tecnica, il POJO di **Book** che abbiamo visto cosa diventa???????

Il mapping di Book

- Nome dell'entità diventa il nome della tabella
- Il nome degli attributi diventano il nome della colonne
- Mapping primitive Java a tipi di dati relazionali: String a VARCHAR, Long a BIGINT, etc. (ma può essere dipendente dal DB)
- Informazioni sul database fornite nel file `persistence.xml`
- Questo mapping di default segue il principio del configuration by exception



Il mapping di Book

- Senza annotazioni, il `Book` entity verrebbe trattato come POJO e pertanto non come classe persistente
- Questa è la regola:
 - Se nessuna speciale configurazione viene indicata, il **comportamento di default** viene applicato
 - Il default per il persistence provider è che **la classe Book non ha una database representation**
 - Cambiare questo comportamento di default si traduce in annotare la classe con `@Entity`**
 - Lo stesso vale per l'identifier: è necessario un modo per dire al persistence provider che l'attributo `id` deve essere mappato in una primary key
 - Si annota pertanto con `@Id`, ed il valore di questo identifier è automaticamente generato dal persistent provider, usando l'annotazione opzionale `@GeneratedValue`

Il mapping di Book

- Seguendo queste regole, l'entità Book sarà mappata in una tabella Derby con la seguente struttura:

Listing 4-2. Script Creating the BOOK Table Structure

Listing 4-2. Script Creating the BOOK Table Structure

```
CREATE TABLE BOOK (
  ID BIGINT NOT NULL,
  TITLE VARCHAR(255),
  PRICE FLOAT,
  DESCRIPTION VARCHAR(255),
  ISBN VARCHAR(255),
  NOOFFPAGE INTEGER,
  ILLUSTRATIONS SMALLINT DEFAULT 0,
  PRIMARY KEY (ID)
)
```

Come fare le query di entità

- JPA permette di assegnare entità a DB e di fare query su di loro ...
- ... **ma sfruttando il linguaggio Java (e non SQL per il DB)**
- Per orchestrare il tutto, serve un **EntityManager** che fornisce le operazioni CRUD (Create, Read, Update, Delete) e le query JPQL
- Il seguente codice permette di ottenere un entity manager e rendere persistente un oggetto nel DB:

EntityManager: interfaccia la cui implementazione è fatta dal persistence provider (EclipseLink)

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("chapter04PU");
EntityManager em = emf.createEntityManager();
em.persist(book);
```

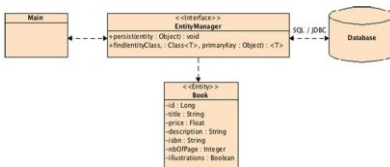
Organizzazione della lezione

- Introduzione
- Le Entità
 - Definizione
 - Anatomia
 - Queries
- Object-Relational Mapping (ORM)
 - Entity Manager
 - La Persistenza (Persistence Unit)
 - Ciclo di vita delle Entità
- JPA Specification
- Putting It All Together
- Managing Persistent Objects
- Conclusioni



Il ruolo dell'Entity Manager

- L'EntityManager interagisce con l'entità e con il database sottostante

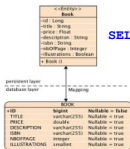


Un esempio di query

- L'entity manager permette di fare query
- In questo caso una query è simile ad una database query ma viene eseguita usando JPQL e non SQL
- **Esempio: vogliamo tutti i libri che abbiamo il titolo: "H2G2"**

Un esempio di query

- **Esempio: vogliamo tutti i libri che abbiamo il titolo: "H2G2"**



`SELECT b FROM Book b WHERE b.title = 'H2G2'`

Nome di un attributo di **Book**, non il nome di una colonna nel database

Un esempio di query

```
SELECT b FROM Book b WHERE b.title = 'H2G2'
```

Select b

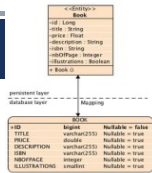
Questa parte indica che stiamo selezionando oggetti di tipo Book, **b** è un alias per l'entità Book

From Book b

Specifica l'entità da cui stiamo selezionando. In questo caso, stiamo selezionando dalla classe entità Book, e assegnando l'alias **b** a questa entità

WHERE b.title = 'H2G2'

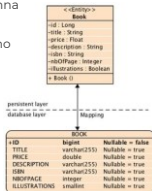
Questa è la clausola di condizione. Stiamo filtrando i libri in base al loro titolo. Cerchiamo specificamente un libro il cui titolo sia esattamente "H2G2".



Ricapitolando...

```
SELECT b FROM Book b WHERE b.title = 'H2G2'
```

- **title** è il nome di un attributo di **Book**, non il nome di una colonna nel database
- Java Persistence Query Language (JPQL) statements manipolano oggetti ed attributi
 - non tabelle e colonne
- Una istruzione JPQL può essere eseguita:
 - con query dinamiche
 - create dinamicamente e run-time
 - con query statiche
 - definite staticamente a tempo di compilazione
 - oppure eseguita con native SQL statement o stored procedures

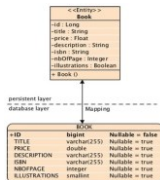


Ricapitolando...

- Le query statiche (named queries) sono definite usando:
 - Annotazioni (@NamedQuery)
 - XML metadata
- L'esempio:

```
SELECT b FROM Book b WHERE b.title = 'H2G2'
```

- Può essere definito come una named query sull'entity **Book**



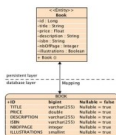
Un esempio di named query

Listing 4-3

```
@Entity
@NamedQuery(
    name = "findBookH2G2",
    query = "SELECT b FROM Book b WHERE b.title = 'H2G2'"
)
public class Book {
    @Id @GeneratedValue
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;

    //Constructors, getters, setters
}
```

Annotazione



Un esempio di named query

Listing 4-3

```

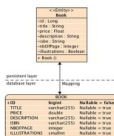
@Entity
@NamedQuery(
    name = "findBookH2G2",
    query = "SELECT b FROM Book b WHERE b.title = 'H2G2'"
)
public class Book {
    @Id @GeneratedValue
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;

    //Constructors, getters, setters
}

```

Annotazione

Definizione di una query con nome



Un esempio di named query

Listing 4-3

```

@Entity
@NamedQuery(
    name = "findBookH2G2",
    query = "SELECT b FROM Book b WHERE b.title = 'H2G2'"
)
public class Book {
    @Id @GeneratedValue
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;

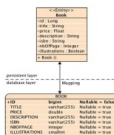
    //Constructors, getters, setters
}

```

Annotazione

Definizione di una query con nome

Cosa fa la query



Un esempio di named query

Listing 4-3

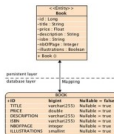
```

@Entity
@NamedQuery(
    name = "findBookH2G2",
    query = "SELECT b FROM Book b WHERE b.title = 'H2G2'"
)
public class Book {
    @Id @GeneratedValue
    private Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;

    //Constructors, getters, setters
}

```

Annotazione
Definizione di una query con nome
Cosa fa la query
Tutto il resto è uguale

Main class: persisting and retrieving a **Book** Entity

```

public class Main {
    public static void main(String[] args) {
        // 1-Creates an instance of book
        Book book = new Book("H2G2", "The Hitchhiker's Guide to the Galaxy", 12.99, "93-84023-742-2", 354, false);
        // 2-Obtains an entity manager and a transaction
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("Chapter04PU2PU");
        EntityManager em = emf.createEntityManager();
        // 3-Persists the book to the database
        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(book);
        tx.commit();
        // 4-Executes the named query for H2G2
        book = em.createNamedQuery("findBookH2G2", Book.class).getSingleResult();
        System.out.println("Query per H2G2");
        System.out.println("***** " + book.getDescription());

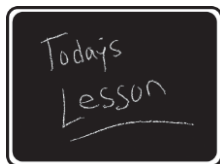
        // 5-Closes the entity manager and the factory
        em.close();
        emf.close();
    }
}

```



Organizzazione della lezione

- Introduzione
- Le Entità
 - Definizione
 - Anatomia
 - Queries
- Object-Relational Mapping (ORM)
 - Entity Manager
 - La Persistenza (Persistence Unit)
 - Ciclo di vita delle Entità
- JPA Specification
- Putting It All Together
- Managing Persistent Objects
- Conclusioni



Persistence Unit

- Mancano alcune informazioni importanti: come si chiama il Database? Che driver JDBC deve essere usato, come connettersi al database?
- Nell'esempio precedente:

```
// 2-Obtains an entity manager and a transaction
```

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("chapter04PU");
EntityManager em = emf.createEntityManager();
```

- Quando la Main class (Listing 4-4) crea un EntityManagerFactory, passa il nome di una persistence unit come parametro
 - Chiamato **chapter04PU**
- Il persistence unit indica all' entity manager il tipo di database da usare, ed i connection parameters, definiti in un file XML
 - Chiamato **persistence.xml**

Persistence Unit

- Le informazioni che è possibile inserire per ogni PU:
 - Nome (della Persistence Unit)
 - Classe a cui si riferisce (Entità a cui si riferisce)
 - Tipo di database (per scegliere il giusto driver JDBC)
 - La posizione (URL)
 - Modalità per autenticazione
- Senza queste specifiche, un POJO può essere usato "semplicemente" come classe per istanze di oggetti Java tradizionali

Un file `persistence.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">
  <persistence-unit name="chapter04PU">
    <transaction-type>"RESOURCE_LOCAL"</transaction-type>
    <provider>
      org.eclipse.persistence.jpa.PersistenceProvider
    </provider>
    <class>org.agoncal.book.javaee7.chapter04.Book</class>
    <properties>
      <property
        name="javax.persistence.schema-generation-action"
        value="drop-and-create"/>
      <property
        name="javax.persistence.schema-generation-target"
        value="database"/>
      <property name="javax.persistence.jdbc.driver"
        value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:derby://localhost:1527/chapter04DB;
        create=true"/>
      <property name="javax.persistence.jdbc.user"
        value="APP"/>
      <property name="javax.persistence.jdbc.password"
        value="APP"/>
    </properties>
  </persistence-unit>
```

Nome della PU

Un file persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">
  <persistence-unit name="chapter04PU"
    transaction-type="RESOURCE_LOCAL">
    <provider>
      org.eclipse.persistence.jpa.PersistenceProvider
    </provider>
    <class>org.agoncal.book.javaee7.chapter04.Book</class>
    <properties>
      <property
        name="javax.persistence.schema-generation-action"
        value="drop-and-create"/>
      <property
        name="javax.persistence.schema-generation-target"
        value="database"/>
      <property name="javax.persistence.jdbc.driver"
        value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:derby://localhost:1527/chapter04DB;
          create=true"/>
      <property name="javax.persistence.jdbc.user"
        value="APP"/>
      <property name="javax.persistence.jdbc.password"
        value="APP"/>
    </properties>
  </persistence-unit>
```

Nome della PU

La classe dell'Entità

Un file persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">
  <persistence-unit name="chapter04PU"
    transaction-type="RESOURCE_LOCAL">
    <provider>
      org.eclipse.persistence.jpa.PersistenceProvider
    </provider>
    <class>org.agoncal.book.javaee7.chapter04.Book</class>
    <properties>
      <property
        name="javax.persistence.schema-generation-action"
        value="drop-and-create"/>
      <property
        name="javax.persistence.schema-generation-target"
        value="database"/>
      <property name="javax.persistence.jdbc.driver"
        value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:derby://localhost:1527/chapter04DB;
          create=true"/>
      <property name="javax.persistence.jdbc.user"
        value="APP"/>
      <property name="javax.persistence.jdbc.password"
        value="APP"/>
    </properties>
  </persistence-unit>
```

Nome della PU

La classe dell'Entità

Il driver JDBC

Un file persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">
  <persistence-unit name="chapter04PU">
    <transaction-type="RESOURCE_LOCAL">
      <provider>
        org.eclipse.persistence.jpa.PersistenceProvider
      </provider>
      <class>org.agoncal.book.javaee7.chapter04.Book</class>
      <properties>
        <property
          name="javax.persistence.schema-generation-action"
          value="drop-and-create"/>
        <property
          name="javax.persistence.schema-generation-target"
          value="database"/>
        <property name="javax.persistence.jdbc.driver"
          value="org.apache.derby.jdbc.ClientDriver"/>
        <property name="javax.persistence.jdbc.url"
          value="jdbc:derby://localhost:1527/chapter04DB;
            create=true"/>
        <property name="javax.persistence.jdbc.user"
          value="APP"/>
        <property name="javax.persistence.jdbc.password"
          value="APP"/>
      </properties>
    </persistence-unit>
  </persistence>
```

- > Nome della PU
- > La classe dell'Entità
- > Il driver JDBC
- > Dove si trova il database

Un file persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">
  <persistence-unit name="chapter04PU">
    <transaction-type="RESOURCE_LOCAL">
      <provider>
        org.eclipse.persistence.jpa.PersistenceProvider
      </provider>
      <class>org.agoncal.book.javaee7.chapter04.Book</class>
      <properties>
        <property
          name="javax.persistence.schema-generation-action"
          value="drop-and-create"/>
        <property
          name="javax.persistence.schema-generation-target"
          value="database"/>
        <property name="javax.persistence.jdbc.driver"
          value="org.apache.derby.jdbc.ClientDriver"/>
        <property name="javax.persistence.jdbc.url"
          value="jdbc:derby://localhost:1527/chapter04DB;
            create=true"/>
        <property name="javax.persistence.jdbc.user"
          value="APP"/>
        <property name="javax.persistence.jdbc.password"
          value="APP"/>
      </properties>
    </persistence-unit>
  </persistence>
```

- > Nome della PU
- > La classe dell'Entità
- > Il driver JDBC
- > Dove si trova il database
- > Con che credenziali va fatta la connessione (user/password)

Riassumendo...

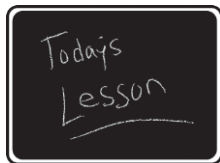
- La persistence unit "chapter04PU" definisce una
 - Connessione JDBC
 - Per il database Derby chapter04DB
 - In esecuzione su localhost e porta 1527
 - Connette un utente (APP) con password (APP) ad una data URL
 - Il tag <class> tag dice al persistence provider di gestire la classe Book

```
<persistence-unit name="chapter04PU" transaction-type="RESOURCE_LOCAL">
  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
  <class>org.agoraal.book.jvane7.chapter04.Book</class>
  <properties>
    <property name="javax.persistence.schema-generation-action" value="drop-and-create"/>
    <property name="javax.persistence.schema-generation-target" value="database"/>
    <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.ClientDriver"/>
    <property name="javax.persistence.jdbc.url" value="jdbc:derby://localhost:1527:chapter04DB;create=true"/>
    <property name="javax.persistence.jdbc.user" value="APP"/>
    <property name="javax.persistence.jdbc.password" value="APP"/>
  </properties>
</persistence-unit>
</persistence>
```

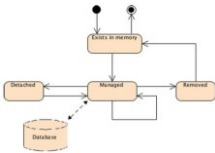
Senza una persistence unit le entità possono essere manipolate esclusivamente come POJO senza funzionalità di persistenza

Organizzazione della lezione

- Introduzione
- Le Entità
 - Definizione
 - Anatomia
 - Queries
- Object-Relational Mapping (ORM)
 - Entity Manager
 - La Persistenza (Persistence Unit)
 - Ciclo di vita delle Entità
- JPA Specification
- Putting It All Together
- Managing Persistent Objects
- Conclusioni



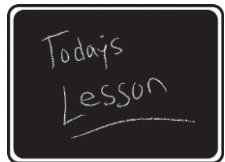
Ciclo di vita delle entità



- Quando si crea una istanza di una entity **Book** con l'operatore **new**, l'oggetto esiste in memoria e JPA non sa niente di lui
- Quando diventa '**managed**' dall'entity manager, la tabella BOOK mappa e sincronizza il suo stato
- Chiamare il metodo EntityManager.remove() cancella i dati dal database, ma gli oggetti Java continuano a rimanere in memoria fino all'intervento del garbage collector
- Le operazioni che è possibile eseguire sulle entità rientrano in 4 categorie:
 - persisting, updating, removing, e loading
- Che corrispondono alle operazioni di:
 - inserting, updating, deleting, e selecting su un database

Organizzazione della lezione

- Introduzione
- Le Entità
 - Definizione
 - Anatomia
 - Queries
- Object-Relational Mapping (ORM)
 - Entity Manager
 - La Persistenza (Persistence Unit)
 - Ciclo di vita delle Entità
- JPA Specification
- Putting It All Together
- Managing Persistent Objects
- Conclusioni



JPA Specification Overview

- JPA 1.0 è stato creato con Java EE 5 per risolvere il problema della persistenza dei dati
 - Mette insieme il modello ad oggetti con il modello relazionale
- In **Java EE 7 e EE 8, JPA 2.1** segue la stessa filosofia di semplicità e robustezza ed aggiunge nuove funzionalità
- JPA è un'astrazione di JDBC e permette indipendenza da SQL
- Tutte le classi e le annotazioni sono contenute nel package `javax.persistence`

**L'ultima versione principale di JPA è JPA 3.1
 (e fa parte della specifica Jakarta EE 10)**

JPA Specification Overview

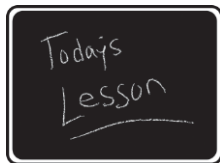
- Le principali componenti di JPA sono:
 - **Object-Relational Mapping (ORM)**: meccanismo che permette di mappare oggetti in dati memorizzati in un database
 - **Entity manager API**: per eseguire database-related operations (CRUD)
 - **Java Persistence Query Language (JPQL)**: permette di recuperare dati con un object-oriented query language
 - **Transaction e Locking mechanisms** che Java Transaction API (JTA) fornisce per gestire l'accesso concorrente ai dati
 - JPA supporta anche resource-local (non-JTA) transactions
 - **Callbacks e listeners**: per agganciare (to hook) business logic nel ciclo di vita di un oggetto persistente

JPA reference implementation

- EclipseLink 2.5 è la reference implementation open source di JPA 2.1
- Fornisce un framework potente e flessibile per memorizzare oggetti Java in un database relazionale
- EclipseLink è la JPA reference implementation e persistence framework usato negli esempi che vedremo
 - Sarà indicato anche con il nome di persistence provider, o semplicemente provider
- JPA 2.1 è supportato anche da Hibernate

Organizzazione della lezione

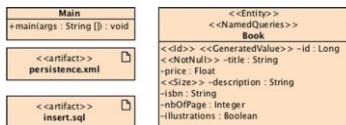
- Introduzione
- Le Entità
 - Definizione
 - Anatomia
 - Queries
- Object-Relational Mapping (ORM)
 - Entity Manager
 - La Persistenza (Persistence Unit)
 - Ciclo di vita delle Entità
- JPA Specification
- Putting It All Together
- Managing Persistent Objects
- Conclusioni



Esempio

- Creare un Entity **Book** e scrivere una piccola applicazione che memorizza l'entità in un database

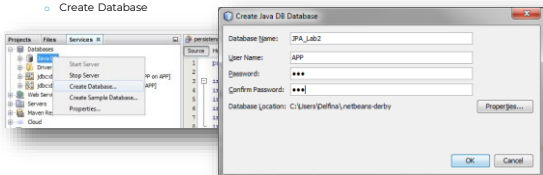
Schema



PASSO 1: CREIAMO IL DATABASE

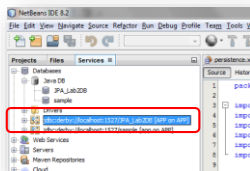
Creiamo il database

- Right-click su Java DB
 - Create Database



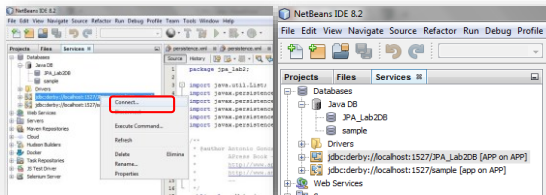
Creiamo la connessione con il database

- Bisogna connetterlo
 - Right-click



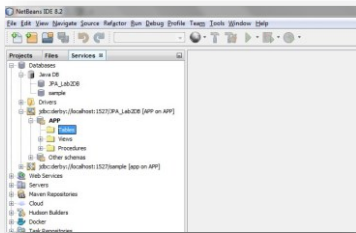
Creiamo la connessione con il database

- Bisogna connetterlo
 - Right-click



Creiamo la connessione con il database

- Nessuna tabella al momento



Writing the Book Entity

```
@Entity
@NamedQueries({
    @NamedQuery(name = "findAllBooks",
        query = "SELECT b FROM Book b"),
    @NamedQuery(name = "findBookH2G2",
        query = "SELECT b FROM Book b WHERE b.title = 'H2G2'")
})
public class Book {
    @Id @GeneratedValue private
    Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;
    //Constructors, getters, setters
}
```

Definizione delle query

Writing the Book Entity

```
@Entity
@NamedQueries({
    @NamedQuery(name = "findAllBooks",
        query = "SELECT b FROM Book b"),
    @NamedQuery(name = "findBookH2G2",
        query = "SELECT b FROM Book b WHERE b.title = 'H2G2'")
})
public class Book {
    @Id @GeneratedValue private
    Long id;
    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;
    //Constructors, getters, setters
}
```

Definizione delle query

Query di tutti i libri

Writing the Book Entity

```
@Entity
@NamedQuery({
    @NamedQuery(name = "findAllBooks",
        query = "SELECT b FROM Book b"),
    @NamedQuery(name = "findBookKH2G2",
        query = "SELECT b FROM Book b WHERE b.title = 'H2G2'")
})
public class Book {
    @Id @GeneratedValue private
    Long id;

    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;
    //Constructors, getters, setters
}
```

Definizione delle query

Query di tutti i libri

Query di un libro specifico

Writing the Book Entity

```
@Entity
@NamedQuery({
    @NamedQuery(name = "findAllBooks",
        query = "SELECT b FROM Book b"),
    @NamedQuery(name = "findBookKH2G2",
        query = "SELECT b FROM Book b WHERE b.title = 'H2G2'")
})
public class Book {
    @Id @GeneratedValue private
    Long id;

    private String title;
    private Float price;
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;
    //Constructors, getters, setters
}
```

Definizione delle query

Query di tutti i libri

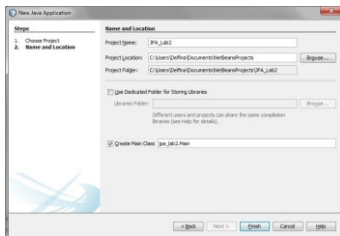
Query di un libro specifico

Definizione entità come vista in precedenza

L'annotazione `@GeneratedValue` informa il persistence provider di autogenerare la chiave primaria usando l'id utility del database sottostante

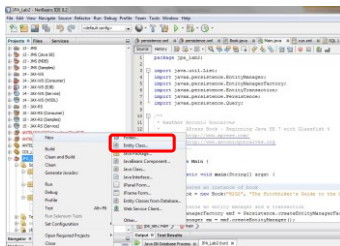
PASSO 2: CREIAMO IL PROGETTO JAVA

Creiamo un JAVA project

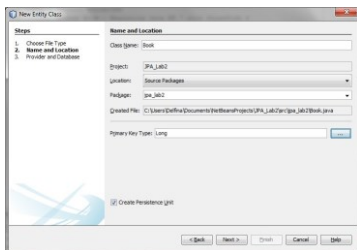


PASSO 3: CREIAMO UNA ENTITY CLASS AUTOMATICAMENTE

Creiamo una Entity class automaticamente

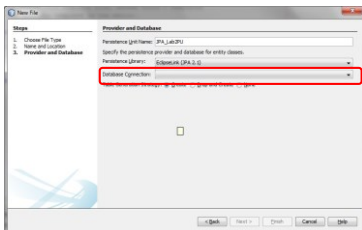


Creiamo una Entity class automaticamente



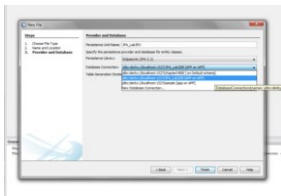
Creiamo una Entity class automaticamente

- Bisogna indicare il database



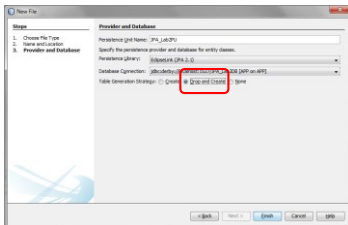
Creiamo una Entity class automaticamente

- Selezioniamo: jdbc://localhost:1527/JPA_Lab2DB[APP on APP]

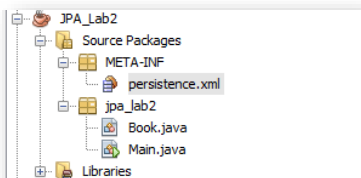


Creiamo una Entity class automaticamente

- Selezioniamo Drop and Create



La struttura del progetto



PASSO 4: MODIFICHIAMO IL FILE Persistence.xml

Il file xml per la persistenza

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
  <persistence-unit name="JPA_Lab2PU"
    transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider
    </provider>
    <class>jpa_lab2.Book</class>
    <properties>
      <property
        name="javax.persistence.schema-generation-action"
        value="drop-and-create"/>
      <property
        name="javax.persistence.schema-generation-target"
        value="database-and-scripts"/>
      <property name="javax.persistence.jdbc.driver"
        value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:derby://localhost:1527/JPA_Lab2DB;create=true"/>
      <property name="javax.persistence.jdbc.user"
        value="APP"/>
      <property name="javax.persistence.jdbc.password"
        value="APP"/>
      <property
        name="javax.persistence.sql-load-script-source"
        value="insert.sql"/>
    </properties>
  </persistence-unit>
```

Nome e tipo della PU
(Application Managed)

Il file xml per la persistenza

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
  <persistence-unit name="JPA_Lab2PU"
    transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider
    </provider>
    <class>jpa_lab2.Book</class>
    <properties>
      <property
        name="javax.persistence.schema-generation-action"
        value="drop-and-create"/>
      <property
        name="javax.persistence.schema-generation-target"
        value="database-and-scripts"/>
      <property name="javax.persistence.jdbc.driver"
        value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:derby://localhost:1527/JPA_Lab2DB;create=true"/>
      <property name="javax.persistence.jdbc.user"
        value="APP"/>
      <property name="javax.persistence.jdbc.password"
        value="APP"/>
      <property
        name="javax.persistence.sql-load-script-source"
        value="insert.sql"/>
    </properties>
  </persistence-unit>
```

Nome e tipo della PU
(Application Managed)

Derby DB

Il file xml per la persistenza

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
<persistence-unit name="JPA_Lab2PU"
transaction-type="RESOURCE_LOCAL">
<provider>org.eclipse.persistence.jpa.PersistenceProvider
</provider>
<class>jpa_lab2.Book</class>
<properties>
<property
name="javax.persistence.schema-generation-action"
value="drop-and-create"/>
<property
name="javax.persistence.schema-generation-target"
value="database-and-scripts"/>
<property name="javax.persistence.jdbc.driver"
value="org.apache.derby.jdbc.ClientDriver"/>
<property name="javax.persistence.jdbc.url"
value="jdbc:derby://localhost:1527/JPA_Lab2DB;create=true"/>
<property name="javax.persistence.jdbc.user"
value="APP"/>
<property name="javax.persistence.jdbc.password"
value="APP"/>
<property
name="javax.persistence.sql-load-script-source"
value="insert.sql"/>
</properties>
</persistence-unit>
```

Nome e tipo della PU
(Application Managed)

Derby DB

Locazione

Il file xml per la persistenza

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
<persistence-unit name="JPA_Lab2PU"
transaction-type="RESOURCE_LOCAL">
<provider>org.eclipse.persistence.jpa.PersistenceProvider
</provider>
<class>jpa_lab2.Book</class>
<properties>
<property
name="javax.persistence.schema-generation-action"
value="drop-and-create"/>
<property
name="javax.persistence.schema-generation-target"
value="database-and-scripts"/>
<property name="javax.persistence.jdbc.driver"
value="org.apache.derby.jdbc.ClientDriver"/>
<property name="javax.persistence.jdbc.url"
value="jdbc:derby://localhost:1527/JPA_Lab2DB;create=true"/>
<property name="javax.persistence.jdbc.user"
value="APP"/>
<property name="javax.persistence.jdbc.password"
value="APP"/>
<property
name="javax.persistence.sql-load-script-source"
value="insert.sql"/>
</properties>
</persistence-unit>
```

Nome e tipo della PU
(Application Managed)

Derby DB

Locazione

Username

Il file xml per la persistenza

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
<persistence-unit name="JPA_Lab2PU"
transaction-type="RESOURCE_LOCAL">
<provider>org.eclipse.persistence.jpa.PersistenceProvider
</provider>
<class>jpa_lab2.Book</class>
<properties>
<property
name="javax.persistence.schema-generation-action"
value="drop-and-create"/>
<property
name="javax.persistence.schema-generation-target"
value="database-and-scripts"/>
<property name="javax.persistence.jdbc.driver"
value="org.apache.derby.jdbc.ClientDriver"/>
<property name="javax.persistence.jdbc.url"
value="jdbc:derby://localhost:1527/JPA_Lab2DB;create=true"/>
<property name="javax.persistence.jdbc.user"
value="APP"/>
<property name="javax.persistence.jdbc.password"
value="APP"/>
<property
name="javax.persistence.sql-load-script-source"
value="insert.sql"/>
</properties>
</persistence-unit>
```

- > Nome e tipo della PU (Application Managed)
- > Derby DB
- > Locazione
- > Username
- > Password

Il file xml per la persistenza

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
<persistence-unit name="JPA_Lab2PU"
transaction-type="RESOURCE_LOCAL">
<provider>org.eclipse.persistence.jpa.PersistenceProvider
</provider>
<class>jpa_lab2.Book</class>
<properties>
<property
name="javax.persistence.schema-generation-action"
value="drop-and-create"/>
<property
name="javax.persistence.schema-generation-target"
value="database-and-scripts"/>
<property name="javax.persistence.jdbc.driver"
value="org.apache.derby.jdbc.ClientDriver"/>
<property name="javax.persistence.jdbc.url"
value="jdbc:derby://localhost:1527/JPA_Lab2DB;create=true"/>
<property name="javax.persistence.jdbc.user"
value="APP"/>
<property name="javax.persistence.jdbc.password"
value="APP"/>
<property
name="javax.persistence.sql-load-script-source"
value="insert.sql"/>
</properties>
</persistence-unit>
```

- > Nome e tipo della PU (Application Managed)
- > Derby DB
- > Locazione
- > Username
- > Password
- > Script SQL per creare un DB

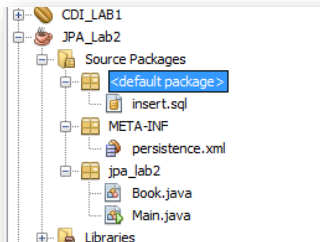
Creiamo lo script per caricare i dati nel DB

- Vogliamo indicare che esiste uno script che carica dati nel database

Listing 4-11. insert.sql File

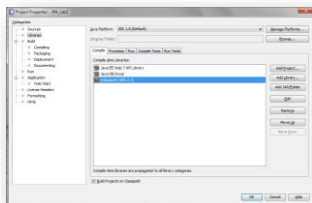
```
INSERT INTO BOOK(ID, TITLE, DESCRIPTION, ILLUSTRATIONS, ISBN, NBOFPAGE, PRICE) values ~
(1000, 'Beginning Java EE 6', 'Best Java EE book ever', 1, '1234-5678', 450, 49)
INSERT INTO BOOK(ID, TITLE, DESCRIPTION, ILLUSTRATIONS, ISBN, NBOFPAGE, PRICE) values ~
(1001, 'Beginning Java EE 7', 'No, this is the best ', 1, '5678-9012', 550, 53)
INSERT INTO BOOK(ID, TITLE, DESCRIPTION, ILLUSTRATIONS, ISBN, NBOFPAGE, PRICE) values ~
(1010, 'The Lord of the Rings', 'One ring to rule them all', 0, '9012-3456', 222, 23)
```

La struttura del progetto



PASSO 5: AGGIUNGIAMO LE LIBRERIE

Le librerie



PASSO 6: MODIFICHIAMO LA CLASSE MAIN

Writing the Main Class

```
package org.agoncal.book.javaee7.chapter04;
public class Main {
    public static void main(String[] args) {
        Book book = new Book("H2G2", 12.5F,
            "The Hitchhiker's Guide to the Galaxy",
            "1-84023-742-2", 354, false);

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("chapter04PU");
        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(book);
        tx.commit();
        System.out.println("##### " + book.getDescription());
        em.close();
        emf.close();
    }
}
```

Metodomain

Writing the Main Class

```
package org.agoncal.book.javaee7.chapter04;
public class Main {
    public static void main(String[] args) {

        Book book = new Book("H2G2", 12.5F,
            "The Hitchhiker's Guide to the Galaxy",
            "1-84023-742-2", 354, false);

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("chapter04PU");
        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(book);
        tx.commit();
        System.out.println("##### " + book.getDescription());
        em.close();
        emf.close();
    }
}
```

Metodo statico

Crea istanza di Book

Writing the Main Class

```
package org.agoncal.book.javaee7.chapter04;
public class Main {
    public static void main(String[] args) {

        Book book = new Book("H2G2", 12.5F,
            "The Hitchhiker's Guide to the Galaxy",
            "1-84023-742-2", 354, false);

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("chapter04PU");
        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(book);
        tx.commit();
        System.out.println("##### " + book.getDescription());
        em.close();
        emf.close();
    }
}
```

Metodo statico

Crea istanza di Book

Ottiene un entity manager factory

Writing the Main Class

```
package org.agoncal.book.javaee7.chapter04;
public class Main {
    public static void main(String[] args) {

        Book book = new Book("H2G2", 12.5F,
            "The Hitchhiker's Guide to the Galaxy",
            "1-84023-742-2", 354, false);

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("chapter04PU");
        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(book);
        tx.commit();
        System.out.println("##### " + book.getDescription());
        em.close();
        emf.close();
    }
}
```

Metodo statico

Crea istanza di Book

Ottiene un entity manager factory

... da cui si ha un entity manager

Writing the Main Class

```
package org.agoncal.book.javaee7.chapter04;
public class Main {
    public static void main(String[] args) {

        Book book = new Book("H2G2", 12.5F,
            "The Hitchhiker's Guide to the Galaxy",
            "1-84023-742-2", 354, false);

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("chapter04PU");
        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(book);
        tx.commit();
        System.out.println("##### " + book.getDescription());
        em.close();
        emf.close();
    }
}
```

Metodo statico

Crea istanza di Book

Ottiene un entity manager factory

... da cui si ha un entity manager

... da cui si ha una transazione

Writing the Main Class

```
package org.agoncal.book.javaee7.chapter04;
public class Main {
    public static void main(String[] args) {

        Book book = new Book("H2G2", 12.5F,
            "The Hitchhiker's Guide to the Galaxy",
            "1-84023-742-2", 354, false);

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("chapter04PU");
        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(book);
        tx.commit();
        System.out.println("##### " + book.getDescription());
        em.close();
        emf.close();
    }
}
```

Metodo statico

Crea istanza di **Book**

Ottiene un entity manager factory

... da cui si ha un entity manager

... da cui si ha una transazione

... in cui si racchiude la "persistenza" del libro

Writing the Main Class

```
package org.agoncal.book.javaee7.chapter04;
public class Main {
    public static void main(String[] args) {

        Book book = new Book("H2G2", 12.5F,
            "The Hitchhiker's Guide to the Galaxy",
            "1-84023-742-2", 354, false);

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("chapter04PU");
        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(book);
        tx.commit();
        System.out.println("##### " + book.getDescription());
        em.close();
        emf.close();
    }
}
```

Metodo statico

Crea istanza di **Book**

Ottiene un entity manager factory

... da cui si ha un entity manager

... da cui si ha una transazione

... in cui si racchiude la "persistenza" del libro

Stampa della descrizione del libro

Writing the Main Class

```
package org.agoncal.book.javaee7.chapter04;
public class Main {
    public static void main(String[] args) {

        Book book = new Book("H2G2", 12.5F,
            "The Hitchhiker's Guide to the Galaxy",
            "1-84023-742-2", 354, false);

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("chapter04PU");
        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(book);
        tx.commit();
        System.out.println("##### " + book.getDescription());
        em.close();
        emf.close();
    }
}
```

Metodo statico

Crea istanza di Book

Ottiene un entity manager factory

... da cui si ha un entity manager

... da cui si ha una transazione

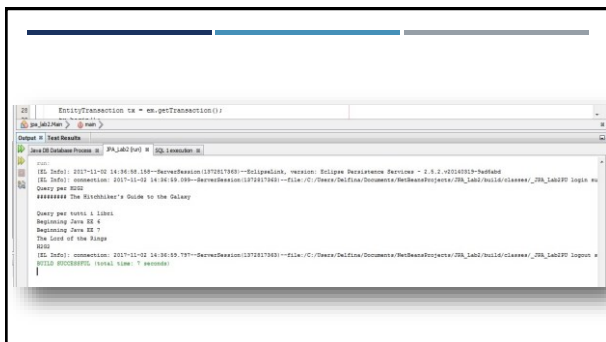
... in cui si racchiude la "persistenza" del libro

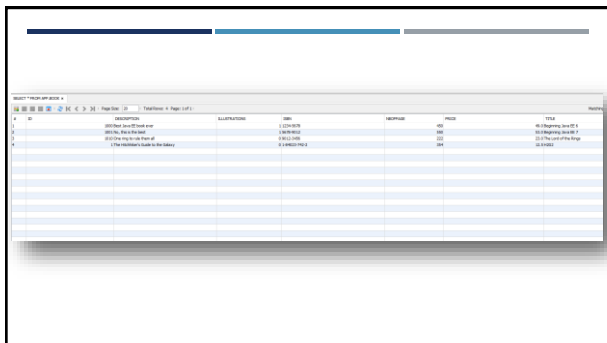
Stampa della descrizione del libro

Si chiude tutto

```
16 public class Main {
17
18     public static void main(String[] args) {
19
20         // 1-Creates an instance of book
21         Book book = new Book("H2G2", "The Hitchhiker's Guide to the Galaxy", 12.5F, "1-84023-742-2", 354, false);
22
23         // 2-Obtains an entity manager and a transaction
24         EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPA_Lab2PU");
25         EntityManager em = emf.createEntityManager();
26
27         // 3-Persist the book to the database
28         EntityTransaction tx = em.getTransaction();
29         tx.begin();
30         em.persist(book);
31         tx.commit();
32
33         // 4-Executes the named query for H2G2
34         book = em.createNamedQuery("findBookH2G2", Book.class).getSingleResult();
35         System.out.println("Query per H2G2");
36         System.out.println("##### " + book.getDescription());
37
38         // 5-Executes the named query for all Books
39         Query all = em.createNamedQuery("findAllBooks", Book.class);
40
41         List<Book> results = all.getResultList();
42         System.out.println("Query per tutti i libri");
43         for (Book b : results) {
44             System.out.println(b.getTitle());
45         }
46
47         // 6-Closes the entity manager and the factory
48         em.close();
49         emf.close();
50     }
51 }
```

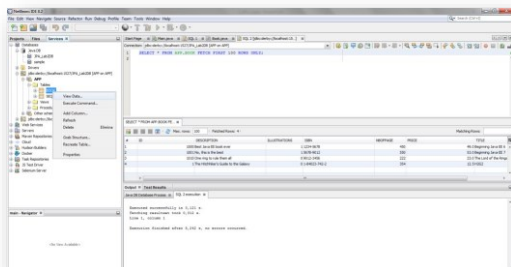
PASSO 7: ESECUZIONE





ID	DESCRIPTION	ILLUSTRATIONS	ISBN	REPRINTS	PRICE	TITLE
1	500 Best Java EE book ever	1	0-13-041878-9	400	45.0	Beginning Java EE 6
2	500 No. 1 book in the field	1	0-13-041878-9	300	50.0	Beginning Java EE 7
3	500 One stop to rule them all	1	0-13-041878-9	200	55.0	The Lord of the Rings
4	The Hobbit's Guide to the Galaxy	1	0-13-041878-9	304	55.0	500

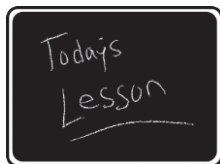
Ora la tabella esiste



The screenshot shows the Oracle JDeveloper IDE interface. The 'Table Editor' window is open, displaying the table 'SELECT * FROM APP.BOOKS'. The table structure is visible, including columns ID, DESCRIPTION, ILLUSTRATIONS, ISBN, REPRINTS, PRICE, and TITLE. The 'Table Editor' window also shows the 'Table Properties' tab with various settings.

Organizzazione della lezione

- Introduzione
- Le Entità
 - Definizione
 - Anatomia
 - Queries
- Object-Relational Mapping (ORM)
 - Entity Manager
 - La Persistenza (Persistence Unit)
 - Ciclo di vita delle Entità
- JPA Specification
- Putting It All Together
- Managing Persistent Objects
- Conclusioni



Le caratteristiche dell'Entity Manager

- Punto centrale di JPA
- Gestisce stato e ciclo di vita delle entità
- Fa query di entità all'interno di un persistence context
- Tra le altre cose, protegge da accessi concorrenti utilizzando tecniche di locking

Le caratteristiche dell'Entity Manager

- Quando un Entity Manager ottiene un riferimento ad una entità viene detto **'managed'**
- Fino a quel momento l'entità è vista come un regolare POJO (i.e., **detached**)
- La potenza di JPA è che le entità possono essere usate come oggetti regolari da differenti layer di un'applicazione e diventare "managed" dall'entity manager quando bisogna caricare o inserire dati in un database
 - Quando l'oggetto è 'managed' si possono eseguire persistence operations, e l'entity manager **automaticamente sincronizzerà lo stato dell'entity con il database**
 - Quando l'entity è 'detached' (i.e., not managed), ritorna ad essere un regolare POJO e quindi può essere usato da altri layers (e.g., a JavaServer Faces, or JSF, presentation layer)
 - Senza sincronizzare il suo stato con il database

Le caratteristiche dell'Entity Manager

Alcune EntityManager API

```
// Factory to create an entity manager, close it and check if it's open
EntityManagerFactory getEntityManagerFactory();
void close();
boolean isOpen();

// Returns an entity transaction
EntityTransaction getTransaction();

// Persists, merges and removes an entity to/from the database
void persist(Object entity);
<T> T merge(T entity);
void remove(Object entity);

// Finds an entity based on its primary key (with different lock mechanisms)
<T> T find(Class<T> entityClass, Object primaryKey);
<T> T find(Class<T> entityClass, Object primaryKey, LockModeType lockMode);
<T> T getReference(Class<T> entityClass, Object primaryKey);
```

Le caratteristiche dell'Entity Manager

Alcune EntityManager API

```
// Synchronizes the persistence context to the underlying database
void flush();
void setFlushMode(FlushModeType flushMode);
FlushModeType getFlushMode();

// Refreshes the state of the entity from the database, overwriting any changes made
void refresh(Object entity);
void refresh(Object entity, LockModeType lockMode);

// Clears the persistence context and checks if it contains an entity
void clear();
void detach(Object entity);
boolean contains(Object entity);

// Sets and gets an entity manager property or hint
void setProperty(String propertyName, Object value);
Map<String, Object> getProperties();

// Creates an instance of Query or TypedQuery for executing a JPQL statement
Query createQuery(String qlString);
<T> TypedQuery<T> createQuery(CriteriaQuery<T> criteriaQuery);
<T> TypedQuery<T> createQuery(String qlString, Class<T> resultClass);
```

Come si ottiene un Entity Manager

- L'entity manager rappresenta l'interfaccia principale per interagire con le entità, ma prima deve essere ottenuta dall'applicazione
- Il codice può essere differente a seconda che l'ambiente sia:
 - Application Managed
 - Container Managed
- Ad esempio, in un **container-managed environment**, le transazioni sono gestite dal container
 - Non si devono scrivere commit o rollback, necessari al contrario per un **application-managed environment**

Come si ottiene un Entity Manager

- Definizioni:

- “Gestito dalla applicazione”: l'applicazione è responsabile per l'istanza specifica di Entity Manager e per gestirne il ciclo di vita
- “Gestito dal container”: quando l'applicazione è una Servlet o un Enterprise Java Bean e quindi ci si affida a risorse iniettate

Esempio di EM Application Managed

```
public class Main {
    public static void main(String[] args) {
        Book book = new Book("H2G2", "The Hitchhiker's Guide",
            12.5F, "1-84023-742-2", 354, false);

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("chapter06PU");
        EntityManager em = emf.createEntityManager();

        EntityTransaction tx = em.getTransaction();
        tx.begin();
        em.persist(book);
        tx.commit();
        System.out.println("##### " + book.getDescription());
        em.close();
        emf.close();
    }
}
```

L'esempio che
abbiamo appena
visto!!!

A Main Class Creating an EntityManager with an EntityManagerFactory

Esempio di EM Container Managed

```
@Stateless
public class BookEJB {

    @PersistenceContext(unitName = "chapter06PU")
    private EntityManager em;

    public void createBook() {

        Book book = new Book("H2G2", "The Hitchhiker's
            Guide", 12.5F, "1-84023-742-2", 354, false);

        em.persist(book);
    }
}
```

Un EJB: richiesta di iniezione di un persistence context

A Stateless EJB Injected with a Reference of an Entity Manager

Esempio di EM Container Managed

```
@Stateless
public class BookEJB {

    @PersistenceContext(unitName = "chapter06PU")
    private EntityManager em;

    public void createBook() {

        Book book = new Book("H2G2", "The Hitchhiker's
            Guide", 12.5F, "1-84023-742-2", 354, false);

        em.persist(book);
    }
}
```

Un EJB: richiesta di iniezione di un persistence context

L'entity manager iniettato

A Stateless EJB Injected with a Reference of an Entity Manager

Esempio di EM Container Managed

```
@Stateless
public class BookEJB {

    @PersistenceContext(unitName = "chapter06PU")
    private EntityManager em;

    public void createBook() {
        Book book = new Book("H2G2", "The Hitchhiker's
            Guide", 12.5F, "1-84023-742-2", 354, false);
        em.persist(book);
    }
}
```

Un EJB: richiesta di iniezione di un persistence context

L'entity manager iniettato

Un metodo per creare un libro

A Stateless EJB Injected with a Reference of an Entity Manager

Esempio di EM Container Managed

```
@Stateless
public class BookEJB {

    @PersistenceContext(unitName = "chapter06PU")
    private EntityManager em;

    public void createBook() {
        Book book = new Book("H2G2", "The Hitchhiker's
            Guide", 12.5F, "1-84023-742-2", 354, false);
        em.persist(book);
    }
}
```

Un EJB: richiesta di iniezione di un persistence context

L'entity manager iniettato

Un metodo per creare un libro

Reso persistente

A Stateless EJB Injected with a Reference of an Entity Manager

Il Persistence Context

- E' un insieme di istanze di entità gestite in un certo tempo per una certa transazione utente
 - Solo una entità con la stessa ID può esistere in un persistence context
 - Se un libro con ID 12 esiste nel persistence context, non potrà esistere nessun altro libro con lo stesso ID
- L'EM aggiorna o consulta il persistence context quando viene chiamato un metodo dell'interfaccia **javax.persistence.EntityManager**
 - Ad esempio, quando il metodo `persist()` viene invocato l'entità passata come argomento verrà aggiunta al persistence context (se non esiste già)
 - Quando si cerca una entità per primary key, l'EM prima controlla che l'entità richiesta non sia nel persistence context
- Una sorta di first-level cache: spazio dove l'entity manager memorizza entità prima della scrittura (`flush()`) nel database

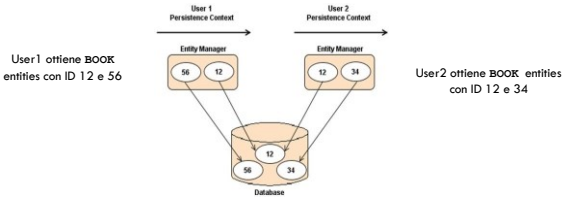
Il Persistence Context

- Ogni utente ha il suo persistence context... che ha vita breve, visto che rappresenta la durata di una transazione

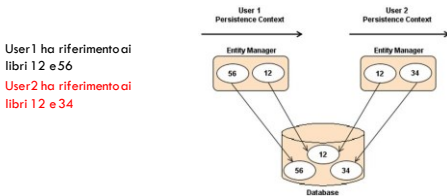
Esempio:

- 2 utenti hanno necessità di accedere all'entità i cui dati sono memorizzati in un database.
 - Ogni utente ha il suo persistence context che ha vita per la durata della sua transazione

Un esempio

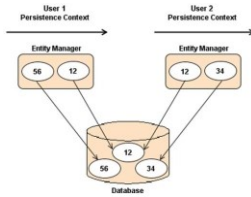


Un esempio



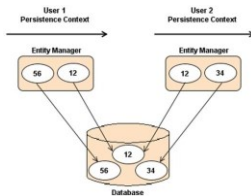
Un esempio

User1 ha riferimento a
libri 12 e 56
User2 ha riferimento a
libri 12 e 34
Il libro 12 appartiene a
entrambi i contesti



Un esempio

User1 ha riferimento a
libri 12 e 56
User2 ha riferimento a
libri 12 e 34
Il libro 12 appartiene a
entrambi i contesti
Al termine della transazione,
il persistence context termina
e le entità eliminate



PU come Bridge fra Context e DB

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence>
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">
<persistence-unit name="chapter06PU">
  transaction-type="RESOURCE_LOCAL">
<provider>org.eclipse.persistence.jpa.PersistenceProvider
</provider>
<class>org.agoncal.book.javaee7.chapter06.Book</class>
<class>org.agoncal.book.javaee7.chapter06.Customer</class>
<class>org.agoncal.book.javaee7.chapter06.Address</class>
<properties>
<property>
  name="javax.persistence.schema-generation.database.action"
  value="drop-and-create"/>
<property name="javax.persistence.jdbc.driver"
  value="org.apache.derby.jdbc.EmbeddedDriver"/>
<property name="javax.persistence.jdbc.url"
  value="jdbc:derby:memory:chapter06DB;create=true"/>
<property name="eclipseLink.logging.level" value="INFO"/>
</properties>
</persistence-unit>
</persistence>
```

Bridge tra le classi gestite come entità
(entità che possono essere gestite nel
persistence context)...

PU come Bridge fra Context e DB

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence>
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">
<persistence-unit name="chapter06PU">
  transaction-type="RESOURCE_LOCAL">
<provider>org.eclipse.persistence.jpa.PersistenceProvider
</provider>
<class>org.agoncal.book.javaee7.chapter06.Book</class>
<class>org.agoncal.book.javaee7.chapter06.Customer</class>
<class>org.agoncal.book.javaee7.chapter06.Address</class>
<properties>
<property>
  name="javax.persistence.schema-generation.database.action"
  value="drop-and-create"/>
<property name="javax.persistence.jdbc.driver"
  value="org.apache.derby.jdbc.EmbeddedDriver"/>
<property name="javax.persistence.jdbc.url"
  value="jdbc:derby:memory:chapter06DB;create=true"/>
<property name="eclipseLink.logging.level" value="INFO"/>
</properties>
</persistence-unit>
</persistence>
```

Bridge tra le classi gestite come entità
(entità che possono essere gestite nel
persistence context)...

... e il database (con le proprietà)

PU come Bridge fra Context e DB

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">
  <persistence-unit name="chapter06PU">
    <transaction-type>RESOURCE_LOCAL</transaction-type>
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>org.agoncal.book.javaee7.chapter06.Book</class>
    <class>org.agoncal.book.javaee7.chapter06.Customer</class>
    <class>org.agoncal.book.javaee7.chapter06.Address</class>
    <properties>
    <property
      name="javax.persistence.schema-generation.database.action"
      value="drop-and-create"/>
    <propertyname="javax.persistence.jdbc.driver"
      value="org.apache.derby.jdbc.EmbeddedDriver"/>
    <propertyname="javax.persistence.jdbc.url"
      value="jdbc:derby:memory:chapter06DB;create=true"/>
    <propertyname="eclipseLink.logging.level" value="INFO"/>
    </properties>
  </persistence-unit>
</persistence>
```

Bridge tra le classi gestite come entità
(entità che possono essere gestite nel
persistence context)...

... e il database (con le proprietà)

Definizione dell'Application
Managed environment

I metodi dell'Entity Manager

Metodi per la manipolazione delle entità

Method	Description
<code>void persist(Object entity)</code>	Makes an instance managed and persistent
<code><T> T find(Class<T> entityClass, Object primaryKey)</code>	Searches for an entity of the specified class and primary key
<code><T> T getReference(Class<T> entityClass, Object primaryKey)</code>	Gets an instance, whose state may be lazily fetched
<code>void remove(Object entity)</code>	Removes the entity instance from the persistence context and from the underlying database

I metodi dell'Entity Manager

Metodi per la manipolazione delle entità

Method	Description
<code><T> T merge(T entity)</code>	Merges the state of the given entity into the current persistence context
<code>void refresh(Object entity)</code>	Refreshes the state of the instance from the database, overwriting changes made to the entity, if any
<code>void flush()</code>	Synchronizes the persistence context to the underlying database
<code>void clear()</code>	Clears the persistence context, causing all managed entities to become detached
<code>void detach(Object entity)</code>	Removes the given entity from the persistence context, causing a managed entity to become detached
<code>boolean contains(Object entity)</code>	Checks whether the instance is a managed entity instance belonging to the current persistence context



Corso di Laurea in Informatica
III Anno Triennale
Programmazione Distribuita



Java Persistence API

Delfina Malandrino

dmalandrino@unisa.it

<http://www.unisa.it/docenti/delfinamalandrino>