Java Persistence API

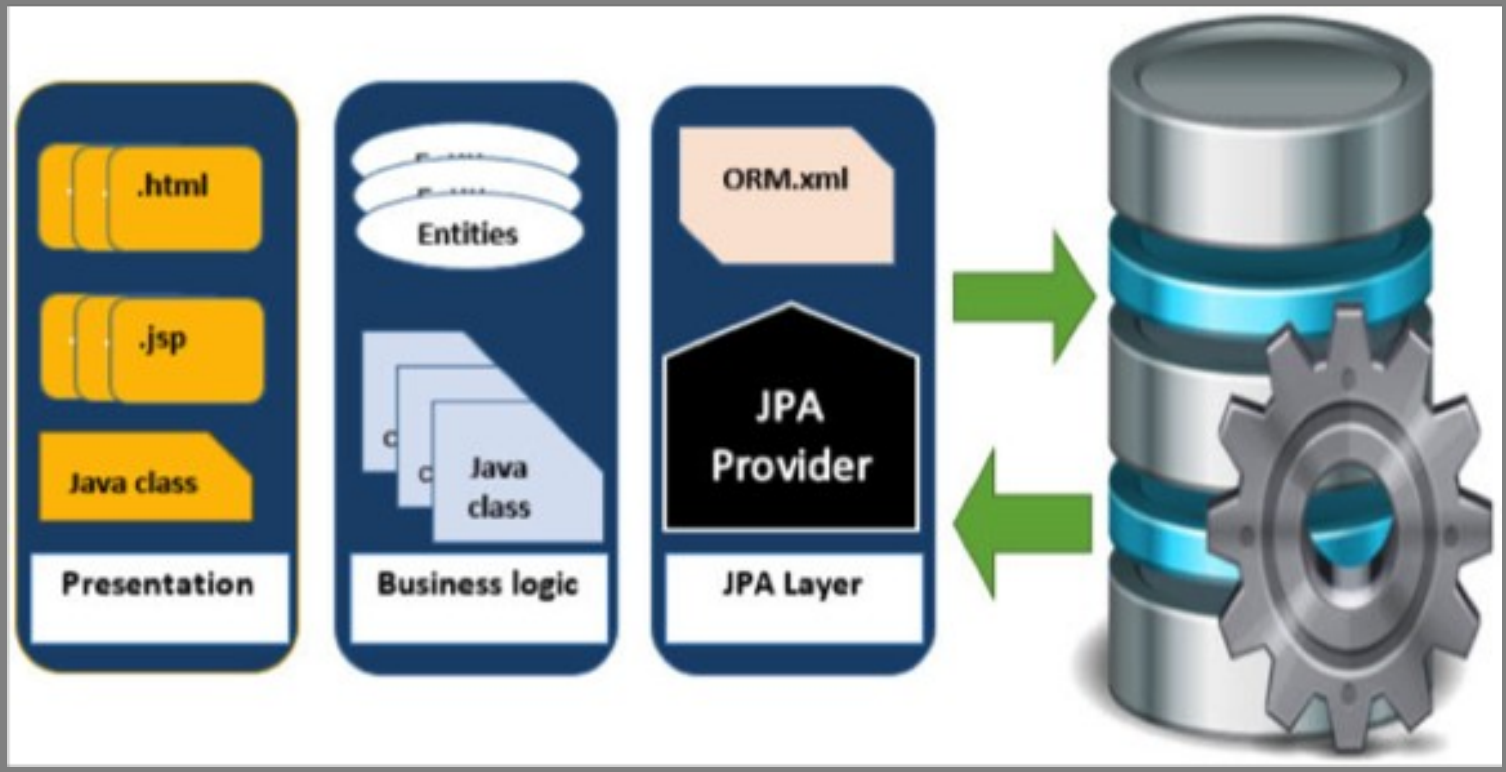
Java Persistence API

La Java Persistence API è una specifica Java per l'accesso, la persistenza e la gestione dei dati tra oggetti Java e database relazionali.

Fornisce:

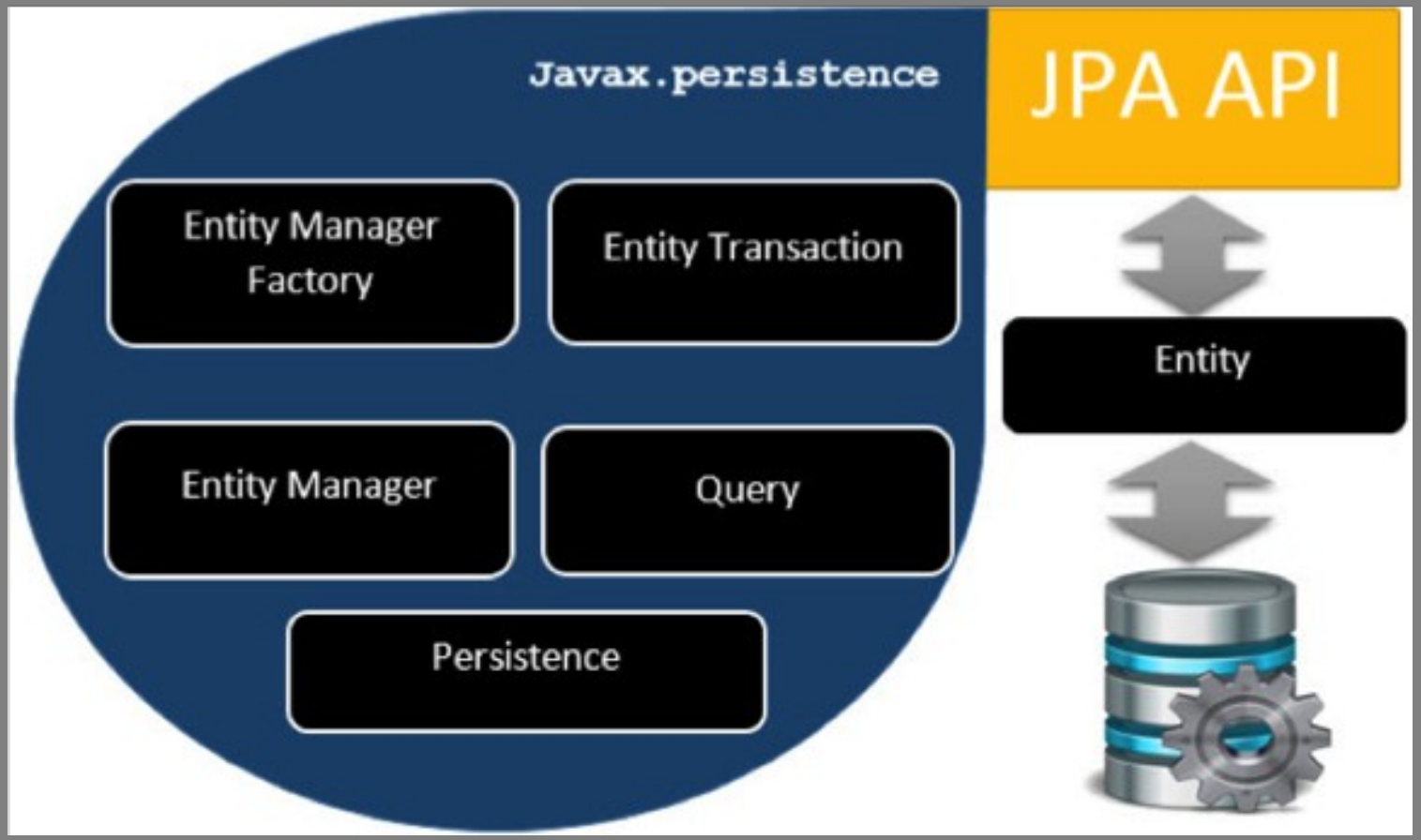
- Modello di persistenza per POJO (Plain Old Java Object) standard
- ORM (Object Relational Mapping): sistema per mappare classi e oggetti Java su tabelle e record di un database relazionale

Dove usare JPA



- Per semplificare la scrittura del codice per la gestione degli oggetti relazionali
- consente una facile interazione con l'istanza del database

Architettura



- Le suddette classi e interfacce vengono utilizzate per archiviare le entità in un database come record.
- Aiutano i programmatori riducendo i loro sforzi per scrivere codici per l'archiviazione dei dati in un database

Architettura

Entity	Le Entity rappresentano gli oggetti di persistenza, memorizzati come record all'interno del database
EntityManager	Rappresenta un'interfaccia che gestisce le operazioni di persistenza sugli oggetti.
EntityManagerFactory	Rappresenta una factory di EntityManager. Crea e gestisce istanze multiple di EntityManager
EntityTransaction	E' in relazione uno-ad-uno con EntityManager. Per ogni EntityManager, controlla le transazioni.
Persistence	Questa classe contiene metodi statici per ottenere l'istanza di EntityManagerFactory
Query	Questa interfaccia viene implementata da ciascun fornitore JPA per ottenere oggetti relazionali che soddisfano i criteri.

Entity

- Un **entity** è a plain old java object (POJO)
- La Class rappresenta una tabella in un database relazionale
- Istanze corrispondono a tuple
- Requisiti:
  - annotato con l'annotazione `javax.persistence.Entity`
  - La classe non deve essere dichiarata `final`
  - I metodi e le variabili persistenti di istanza non devono essere dichiarati `final`

Requisiti per Entity (cont.)

- Può essere **`Serializable`**, ma non necessario
- Entities possono extend sia classi entity che non-entity
- Classi non-entity possono extend classi entity
- Le variabili di istanza persistenti devono essere dichiarate private, protected, o (default)

- Esempio:

```
@Entity
class Person{
   . . .
}
```

Persistent Fields and Properties

- Lo stato di un entity può essere acceduto:
  - attraverso le instance variables
  - attraverso JavaBeans-style properties (getters/setters)
- Tipi supportati:
  - Tipi primitivi, String, tipi enumerati
  - Altre entities e/o collezioni di entities
  - Classi embeddable
- Tutti i campi non annotati con `@Transient` saranno memorizzati nel database!

Primary Keys in Entity

- Ogni entity deve avere un oggetto identificatore univoco (persistent identifier)

```
@Entity
public class Employee {
   @Id private int id;
   private String name;
   private Date age;

   public int getId() { return id; }
   public void setId(int id) { this.id = id; }
   . . .
}
```

Persistent Identity

- Identificatore (id) in entity = primary key nel database
- Identifica univocamente un entity nella memoria e nel DB
- Tipi di identificatori:
  - Semplice – single field/property
    ```
    @Id int id;
    ```
  - Composito – multiple fields/properties
    ```
    @Id int id;
    @Id String name;
    ```
  - Embedded id – single field of PK class type
    ```
    @EmbeddedId EmployeePK id;
    ```

Generazione degli identificatori

Gli identificatori possono essere generati nel database specificando **`@GeneratedValue`** sull'dentificatore

- 4 strategie di generazione pre-definite
  - AUTO, IDENTITY, SEQUENCE, TABLE

- Specificare la strategia AUTO indica che l'implementazione del API sceglierà una strategia

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private int id;
```

Personalizzare gli oggetti Entity

- In molti casi, le caratteristiche di defaults sono sufficienti
- Di default il nome della tabella corrisponde al nome della classe
- Personalizzazione:

```
@Entity
@Table(name = “FULLTIME_EMPLOYEE")
public class Employee{ …… }
```

- Le caratteristiche di defaults delle colonne possono essere personalizzate usanto l’annotazione **@Column**

```
@Id @Column(name = “EMPLOYEE_ID”, nullable = false)
private String id;

@Column(name = “FULL_NAME” nullable = true, length = 100)
private String name;
```

Relazioni tra Entity

- Esistono quattro tipi di relazioni:
  - **`@OneToOne`**
  - **`@OneToMany`**
  - **`@ManyToOne`**
  - **`@ManyToMany`**
- La direzione di una relazione può essere:
  - **bidirezionale** – il riferimento tra gli oggetti è reciproco
  - **unidirezionale** – un oggetto fa riferimento ad un altro ma non avviene il contrario

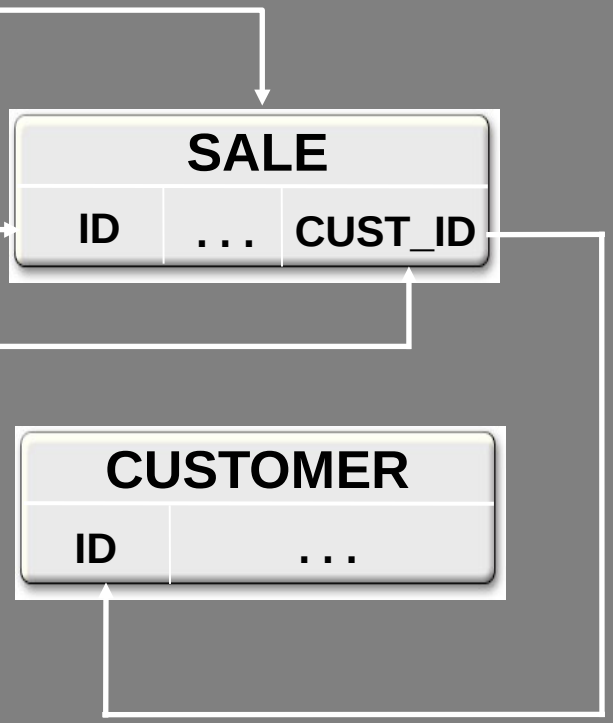
Attributi delle relazioni tra Entity

- JPA supporta cascading updates/deletes: la propagazione delle operazioni di aggiornamento e rimozione su oggetti collegati tra loro da relazioni
  - CascadeType
    - **ALL, PERSIST, MERGE, REMOVE, REFRESH**
- Possiamo dichiarare una strategia di fetch dei dati
  - FetchType
    - **LAZY: carica gli oggetti correlati solo se servono**
    - **EAGER: carica tutti gli oggetti del grafo delle relazioni**

```
@ManyToMany(
    cascade = {CascadeType.PERSIST, CascadeType.MERGE},
    fetch = FetchType.EAGER)
```

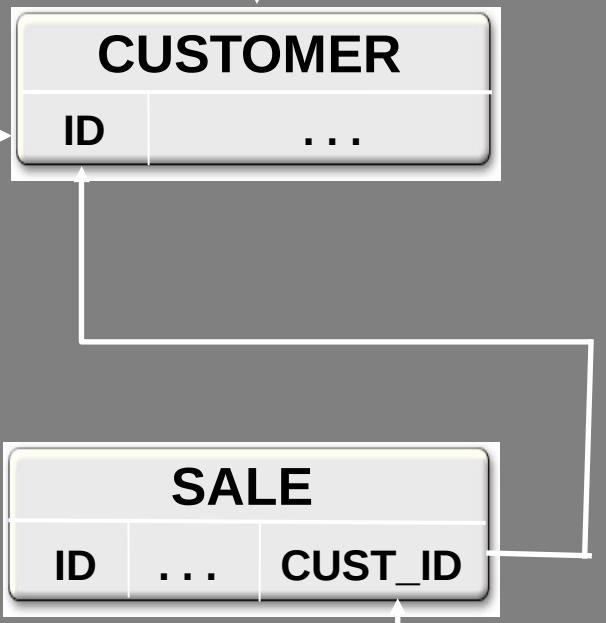
ManyToOne Mapping

```
@Entity
public class Sale {
    @Id
    int id;

    ...

    @ManyToOne
    Customer cust;
}
}
```



OneToMany Mapping

```
@Entity
public class Customer {
   @Id
   int id;
   ...
   @OneToMany(mappedBy="cust")
   Set<Sale> sales;
}
@Entity
public class Sale {
   @Id
   int id;
   ...
   @ManyToOne
   Customer cust;
}
```



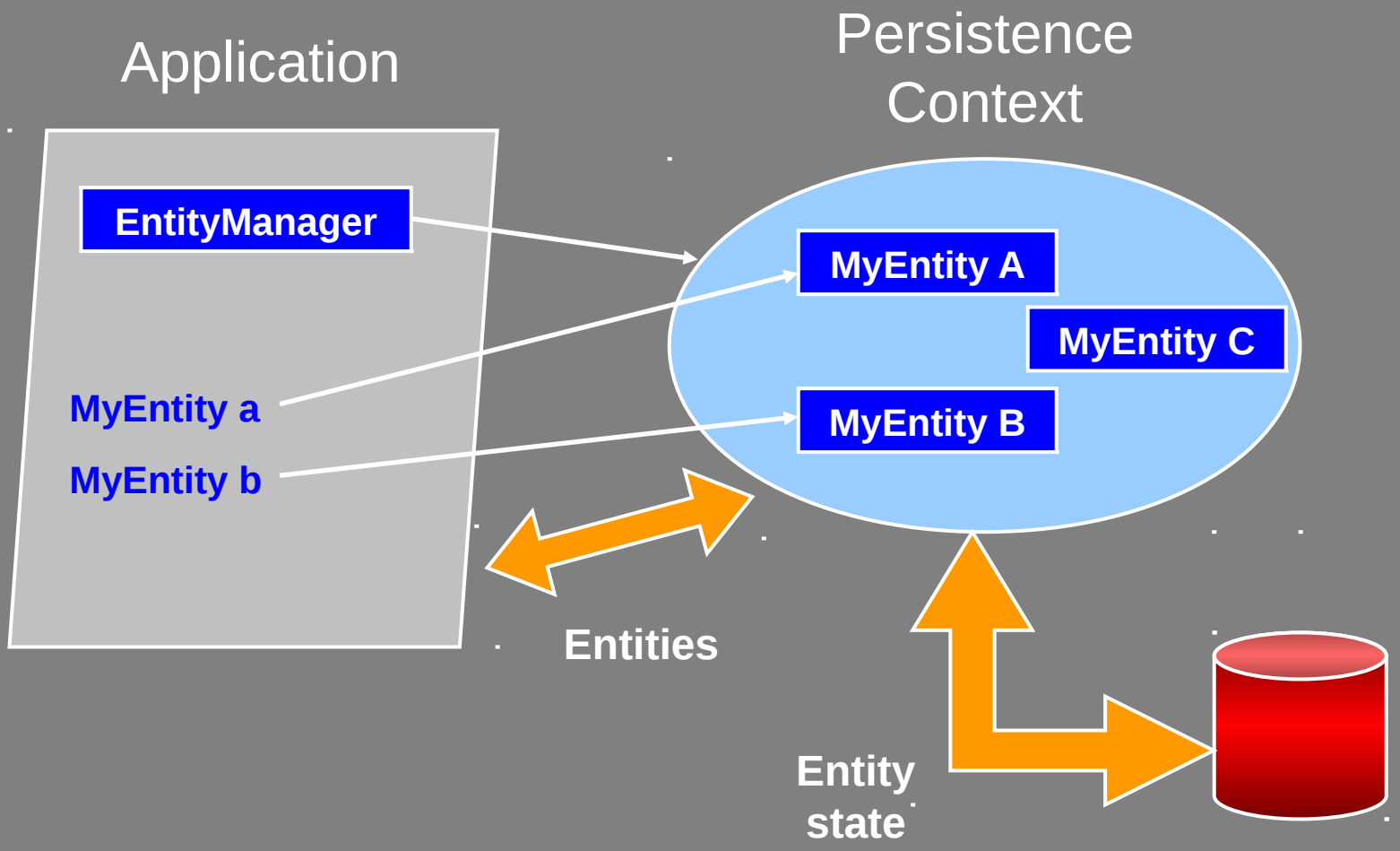
Ereditarietà degli Entity

- JPA suppporta l'ereditarietà ed il polimorfismo
- Ogni Entities può ereditare da un'altra entities e da non-entities
- L'annotazione **`@Inheritance`** identifica una strategia di mapping:
  - SINGLE_TABLE
  - JOINED
  - TABLE_PER_CLASS

Gestione di Entities

- Entities sono gestite dal **entity manager**
- Entity manager è rappresentato da un istanza `javax.persistence.EntityManager`
- Ogni istanza di EntityManager è associata ad un **persistence context**
- Un persistence context definisce lo scope all'interno del quale le istanze di Entity sono create, gestite e rimosse

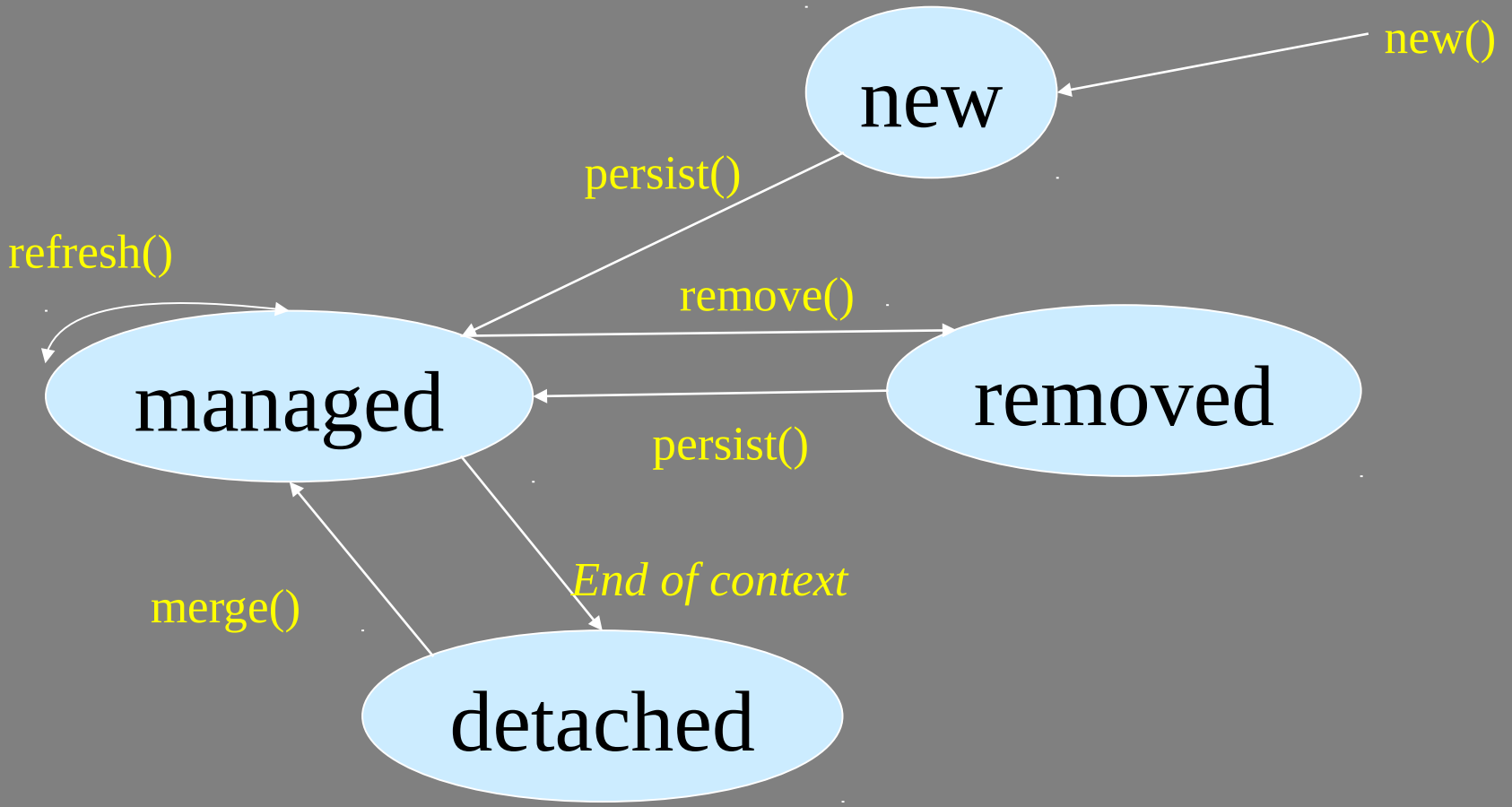
Persistence Context



Entity Manager

- Un istanza di **`EntityManager`** è usata per gestite lo stato ed il ciclo divita delle entities con un persistence context
- Le entities possono essere in uno dei seguenti stati:
  1. New
  2. Managed
  3. Detached
  4. Removed

Entity Lifecycle



Entity Lifecycle

- New – Entity è istanziata ma non associata al persistence context. Non collegata al database.
- Managed – associata al persistence context. Le modifiche sono sincronizzate con il database
- Detached – ha un id, ma non collegata al database
- Removed – associate con un persistence context, ma l'eliminazione dal database è già stata schedulata.

Entity Manager

- EntityManager API:
  - crea e rimuove istanze di entity
  - cercare entities tramite la chiave primaria
  - permette l’esecuzione di queries sulle entities

Operationi sugli oggetti Entity

- EntityManager API operations:
  - **`persist()`-** Memorizza l'entity nel db
  - **`remove()`-** Elimina l'entity dal db
  - **`refresh()`-** Ricarica lo stato dell'entity dal db
  - **`merge()`-** Sincronizza un entity detached con il p/c
  - **`find()`-** Cerca un entity tramite primary key
  - **`createQuery()`-** Crea una query usando JPQL
  - **`createNamedQuery()`-** Crea una query predefinita
  - **`createNativeQuery()`-** Crea una query SQL nativa "
  - **`contains()`-** se un entity è gestita da p/c
  - **`flush()`-** Forza la sincronizzazione del p/c con il database

Nota: p/c == current persistence context

Persistence Units

- Un persistence unit definisce l'insieme di tutte le classi Entity gestite da EntityManager in una applicazione
- Ogni persistence unit può avere differenti providers e database drivers
- Le Persistence units sono definite dal file di configurazione **`persistence.xml`**

JPQL

- JPA ha un query language basato su SQL
- JPQL è un estenzione del EJB QL
- Più robusto, flessibile e object-oriented rispetto a SQL
- Il persistence engine analizza la query, trasforma il JPQL in SQL nativo e lo esegue

Creazione delle queries

- Le istance di Query sono ottenute usando:
  - EntityManager.createNamedQuery (static query)
  - EntityManager.createQuery (dynamic query)
  - EntityManager.createNativeQuery (native query)

- Query API (le principali):
  - getResultList() – esegue la query e ritorna risultati multipli
  - getSingleResult() – esegue la query e ritorna un singolo risultato
  - setFirstResult() – imposta il primo risultare da recuperare
  - setMaxResults() – imposta il numero massimo di risultati da recuperare
  - setParameter() – collega il valore a un parametro posizionale o con nome

Static (Named) Queries

- Definite staticamente con l'annotazione @NamedQuery insieme con la entity class
- elements:
  - name – il nome della query che sarà usato con il metodo createNamedQuery
  - query – query string

```
@NamedQuery(name="findAllCustomers",
            query="SELECT c FROM Customer")
```

```
Query findAllQuery =
    entityManager.createNamedQuery("findAllCustomers");
List customers = findAllQuery.getResultList();
```

Multiple Named Queries

Named queries multiple possono essere definite tramite l'annotazione **@NamedQueries**

```
@NamedQueries( {
    @NamedQuery(name = "Mobile.selectAllQuery"
            query = "SELECT M FROM MOBILEENTITY"),
    @NamedQuery(name = "Mobile.deleteAllQuery"
            query = "DELETE M FROM MOBILEENTITY")
} )
```

Queries dinamiche

- Le query dinamiche sono queries che sono definite direttamente all'interno della logica di business
- !Non efficienti & lente. Il Persistence engine deve analizzare, validare & mappare JPQL in SQL a run-time

```
public List findAll(String entityName){
    return entityManager.createQuery(
        "select e from " + entityName + " e")
    .getResultList();
}
```

Parametri con nome

- I parametri con nome sono parametri in una query che hanno il prefisso formato da **due punti** (**:**)
- Per collegare il parametro ad un argomento viene usato il metodo:
  - **Query.setParameter(String name, Object value)**

```
public List findWithName(String name) {
     return em.createQuery(
"SELECT c FROM Customer c WHERE c.name LIKE :custName")
    .setParameter("custName", name)
    .getResultList();
}
```

Parametri posizionali

- I parametri posizionali sono individuati dal punto interrogativo (**?**) seguito dal numero del paramentro nella query
- Per collegare il parametro ad un argomento viene usato il metodo:
  - **Query.setParameter(integer position, Object value)**

```
public List findWithName(String name) {
    return em.createQuery(
    “SELECT c FROM Customer c WHERE c.name LIKE ?1”)
    .setParameter(1, name)
    .getResultList();
}
```

Queries native

- Queries possono essere scritte in SQL nativo
- Vengono usate quando si necessita di usare il SQL nativo del database target

```
Query q = em.createNativeQuery(
  "SELECT o.id, o.quantity, o.item " +
  "FROM Order o, Item i " +
  "WHERE (o.item = i.id) AND (i.name = 'widget')",
  com.acme.Order.class);
```

Operazioni – Resulti multipli

- **`Query.getResultList()`** esegue una query e ritorna una List di object contenente istanze multiple di entity

```
Query query = entityManager.createQuery("SELECT C FROM CUSTOMER");
List<MobileEntity> mobiles = (List<MobileEntity>)query.getResultList();
```

- Può eseguire solo statement di select
  - Per uno statement diverso dalla select a run-time viene lanciata una **`IllegalStateException`**

Operazioni– risultato singolo

- Una query che ritorna un singolo oggetto entity

```
Query singleSelectQuery = entityManager.createQuery(
“SELECT C FROM CUSTOMER WHERE C.ID = ‘ABC-123’”);
Customer custObj = singleSelectQuery.getSingleResult();
```

- Se il match non ha successo viene lanciata una EntityNotFoundException

- Se c’è più di un match viene lanciata una NonUniqueResultException

Paginazione del risultato

```
int maxRecords = 10; int startPosition = 0;
String queryString = “SELECT M FROM MOBILEENTITY”;
while(true){
   Query selectQuery = entityManager.createQuery(queryString);
   selectQuery.setMaxResults(maxRecords);
   selectQuery.setFirstResult(startPosition);
   List<MobileEntity> mobiles =
   entityManager.getResultList(queryString);
   if (mobiles.isEmpty()){ break; }
   process(mobiles);       // process the mobile entities
   entityManager.clear(); // detach the mobile objects
   startPosition = startPosition + mobiles.size();
}
```

JPQL Statement Language

- JPQL statement types:
  - SELECT, UPDATE, DELETE
- Supported clauses:
  - FROM
  - WHERE
  - GROUP_BY
  - HAVING
  - ORDER BY
  - ...
- Conditional expressions, aggregate functions, ...