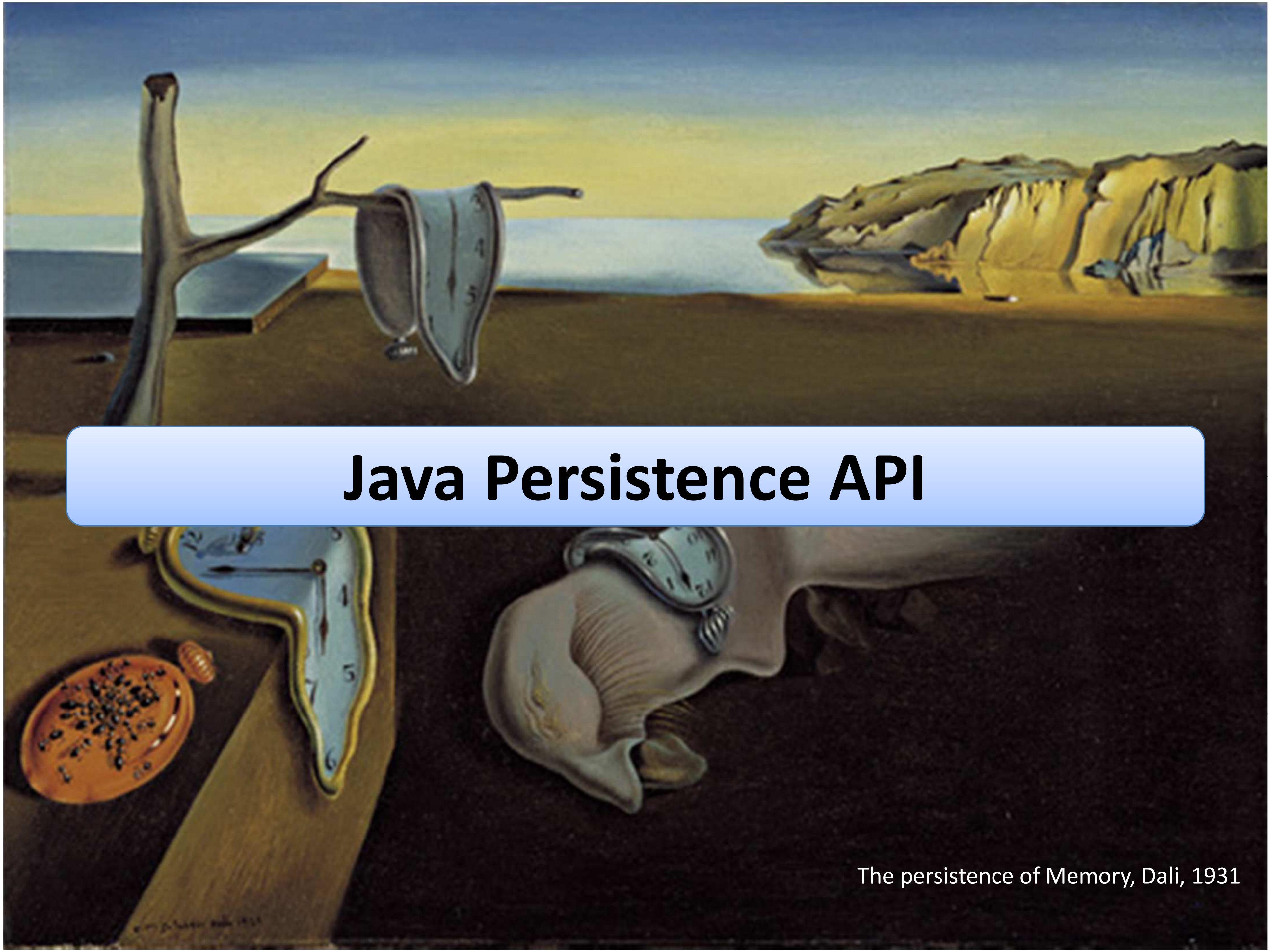


Java Persistence API

The persistence of Memory, Dali, 1931

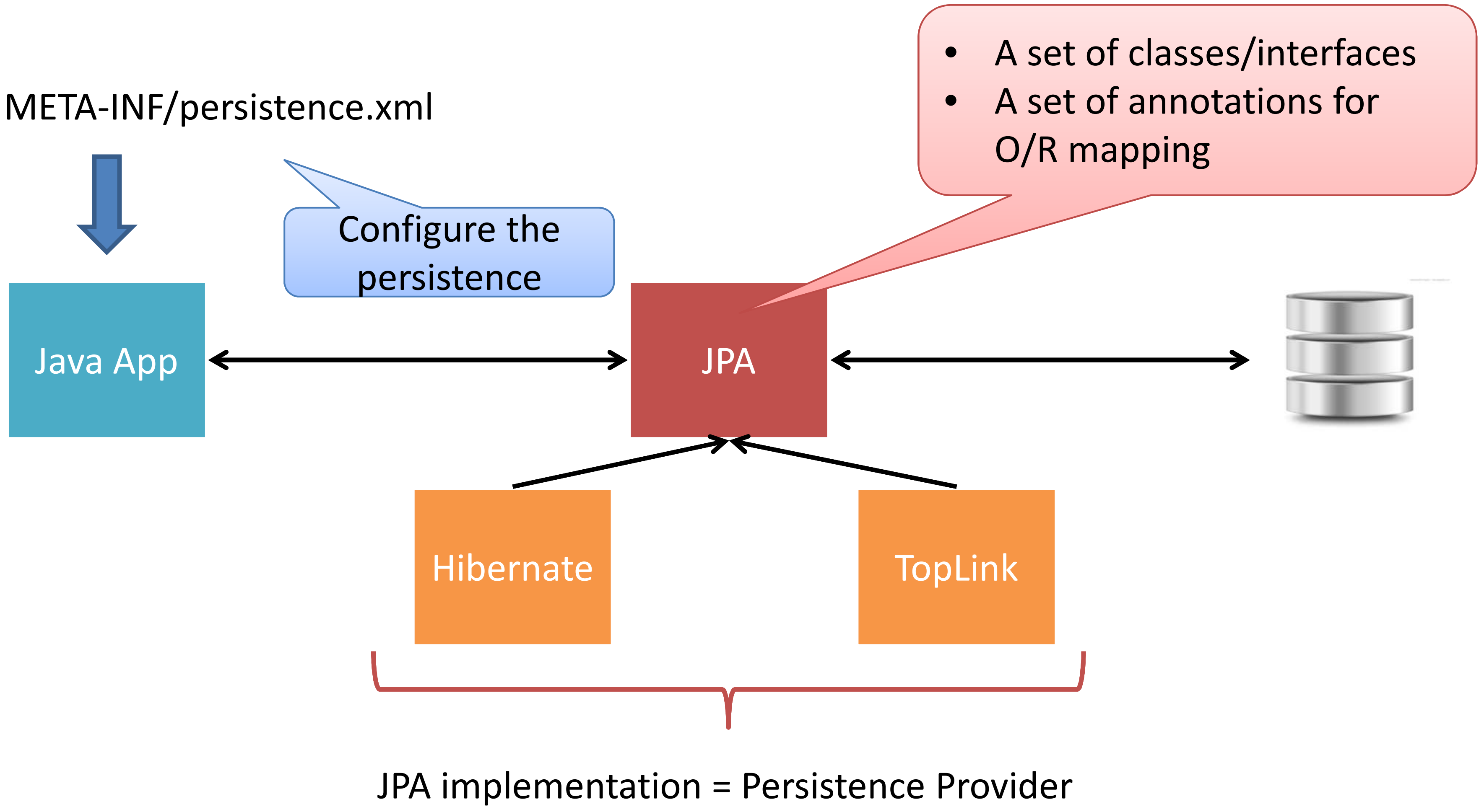
JPA – Presentation (1/2)

- Bundled with the EJB 3.0 specification
  - JPA 1.0 – 2004 JSR 220
  - JPA 2.0 – 2009 JSR 317
- Released with the Java EE 5 platform **BUT** JPA can be used in a Java SE context!
- Standardization of the best existing persistence technologies (Hibernate, TopLinnk, etc.)

JPA – Presentation (3/3)

- It's composed by:
  - An API defined in the `javax.persistence` package
  - The **J**ava **P**ersistence **Q**uery **L**anguage (JPQL)
  - Object/relational metadata

JPA - Architecture


META-INF/persistence.xml
Configure the persistence
• A set of classes/interfaces
• A set of annotations for O/R mapping
Java App
JPA
Hibernate
TopLink
JPA implementation = Persistence Provider

JPA - Principles

- Annotations vs Configuration files
  - XML always overrides values
- Convention Over Configuration (Configuration by exception)
  - configure only to override default

→ Autotmatic binding

JPA – Essential ORM

What’s an Entity (1/3)

- It’s the thing you persist!

- It’s a POJO (**P**lain **O**ld **J**ava **O**bject)

* POJO – n.m. [pôdjô]

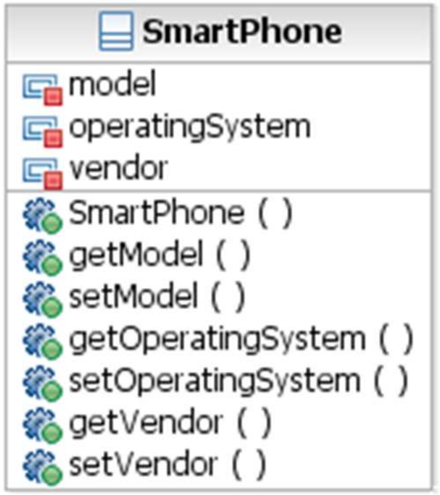
POJO is an acronym for **Plain Old Java Object**, and is favoured by advocates of the idea that the simpler the design, the better. - Wikipedia.

"We wondered why people were so against using regular objects in their systems and concluded that it was because simple objects lacked a fancy name. So we gave them one, and it's caught on very nicely."

- Martin Fowler

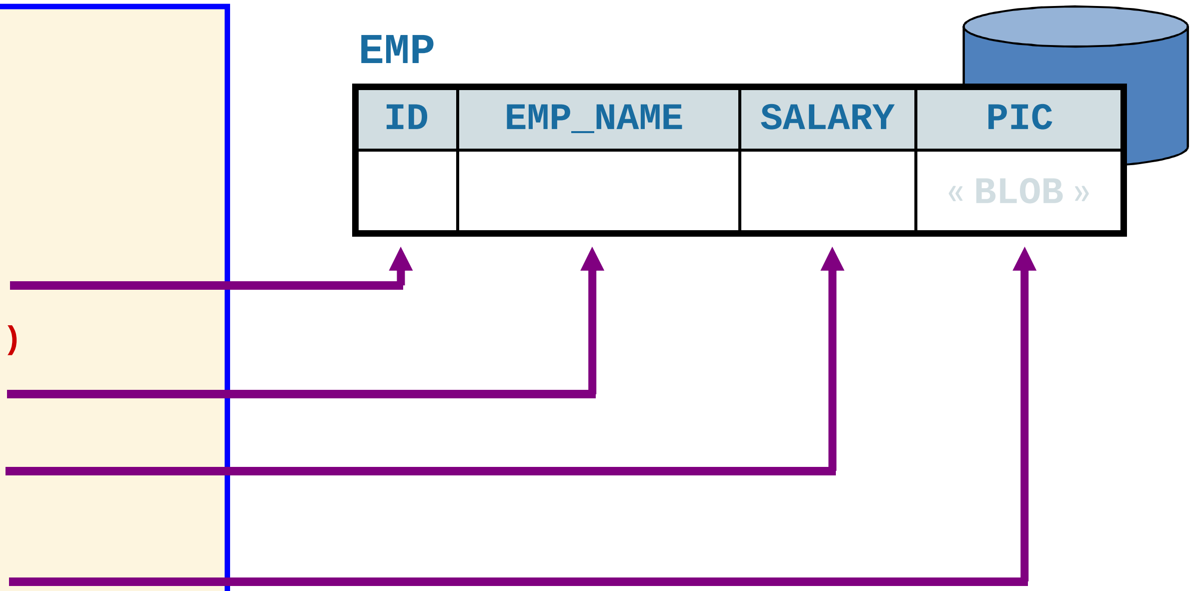
What’s an Entity (2/3)

- Simply annotate your class with « @Entity »
- Must have an @Id annotated identity field
- Follow the JavaBean convention
  - Camel casing
  - Public no-args constructor
  - Getter/setter
  - Not final



What's an Entity (3/3)

```
@Entty
@Table(name="EMP")
public class Employee {

  @Id
  private int id;

  @Column(name="EMP_NAME")
  private String name;

  private double salary;

  @Lob
  private byte[] pic;

  // getters & setters
  ...

}
```



JPA – ORM Basics (1/3)

- **@Transient**
  - This annotation specifies that the property or field is not persistent
- **@GeneratedValue**
  - Primary Key generation strategy.
  - GenererationType values: (AUTO | IDENTITY | SEQUENCE | TABLE)
- **@Basic**
  - The Basic annotation is the simplest type of mapping to a database column. Optional, it can be used to basics persistent field.
- **@Column**
  - Is used to specify a mapped column for a persistent property or field. If no Column annotation is specified, the default values are applied.

Have a look at the Javadoc!!

JPA – ORM Basics (2/3)

- **@Enumerated**
  - Specifies that a persistent property or field should be persisted as a enumerated type.
  - EnumType values: (ORDINAL| STRING)

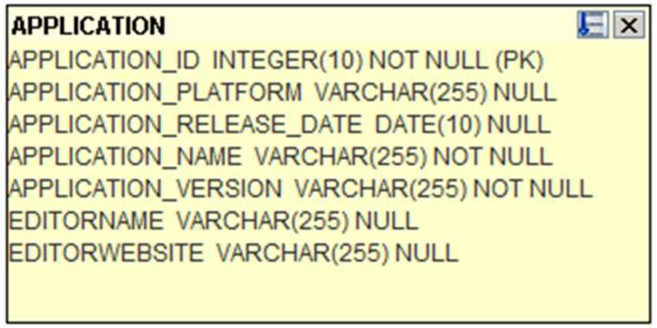
- **@Temporal**
  - This annotation must be specified for persistent fields or properties of type *Date* and *Calendar*.
  - TemporralType values: (DATE, TIME, TIMESTAMP)

- **@Embeddable**
  - Defines a class whose instances are stored as an intrinsic part of an owning entity and share the identity of the entity.

- **@Embedded**
  - Defines a persistent field or property of an entity whose value is an instance of an embeddable class.

JPA – ORM Basics (3/3)

```java
@Entity
@Table(name = "APPLICATION", catalog = "", schema = "APP")
public class Application implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Basic(optional = false)
    @Column(name = "APPLICATION_ID", nullable = false)
    private Integer id;

    @Basic(optional = false)
    @Column(name = "APPLICATION_NAME", nullable = false, length = 255)
    private String name;

    @Basic(optional = false)
    @Column(name = "APPLICATION_VERSION", nullable = false, length = 255)
    private String version;


    @Column(name = "APPLICATION_RELEASE_DATE")
    @Temporal(TemporalType.DATE)
    private Date releaseDate;

    @Column(name = "APPLICATION_PLATFORM")
    @Enumerated(EnumType.STRING)
    private Platform platform;

    @Embedded
    private Editor editor;

    //Getter + Setter
}
```



APPLICATION

```
APPLICATION_ID  INTEGER(10) NOT NULL (PK)
APPLICATION_PLATFORM  VARCHAR(255) NULL
APPLICATION_RELEASE_DATE  DATE(10) NULL
APPLICATION_NAME  VARCHAR(255) NOT NULL
APPLICATION_VERSION  VARCHAR(255) NOT NULL
EDITORNAME  VARCHAR(255) NULL
EDITORWEBSITE  VARCHAR(255) NULL
```

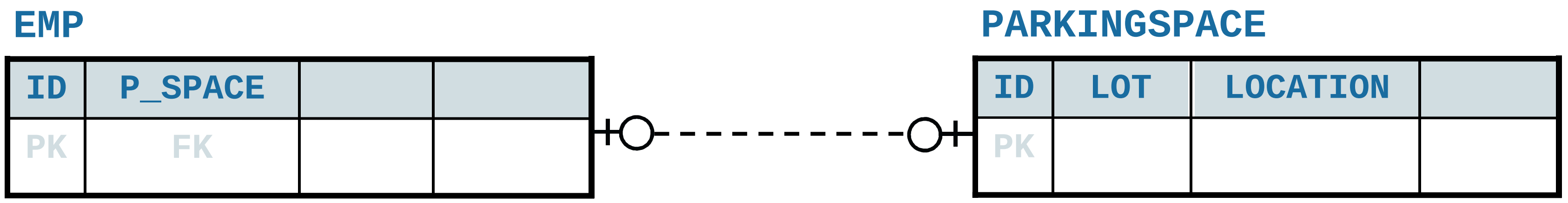
Relationships

- @OneToOne
- @OneToMany
- @ManyToMany
- @ManyToOne

@OneToOne

```java
@Entity
@Table(name="EMP")
public class Employee {

  @Id
  private int id;


  @OneToOne
  @JoinColumn(name="P_SPACE")
  private ParkingSpace space;


  // getters & setters
  ...

}
```

```java
@Entity
public class ParkingSpace {

  @Id
  private int id;

  private int lot;

  private String location;

  @OneToOne(mappedBy="space")
  private Employee emp;

  // getters & setters
  ...

}
```



EMP

ID	P_SPACE		
PK	FK		

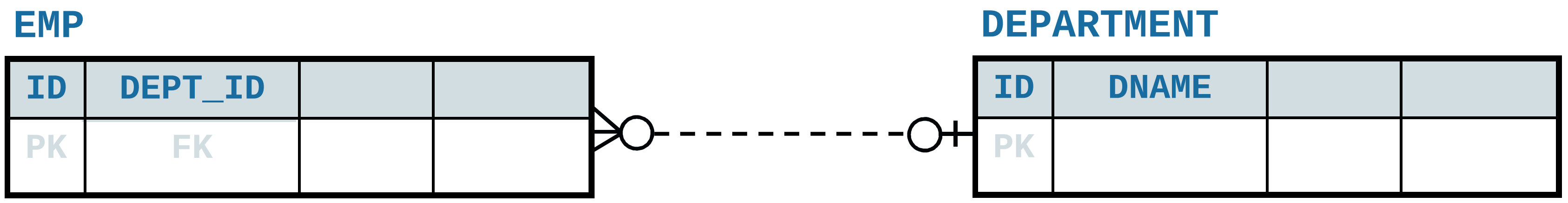
PARKINGSPACE

ID	LOT	LOCATION	
PK			

@ManyToOne

```java
@Entity
@Table(name="EMP")
public class Employee {

  @Id
  private int id;


  @ManyToOne
  @JoinColumn(name="DEPT_ID")
  private Department d;


  // getters & setters
  ...

}
```

```java
@Entity
public class Department {

  @Id
  private int id;

  private String dname;


  // getters & setters
  ...

}
```



EMP

ID	DEPT_ID		
PK	FK		

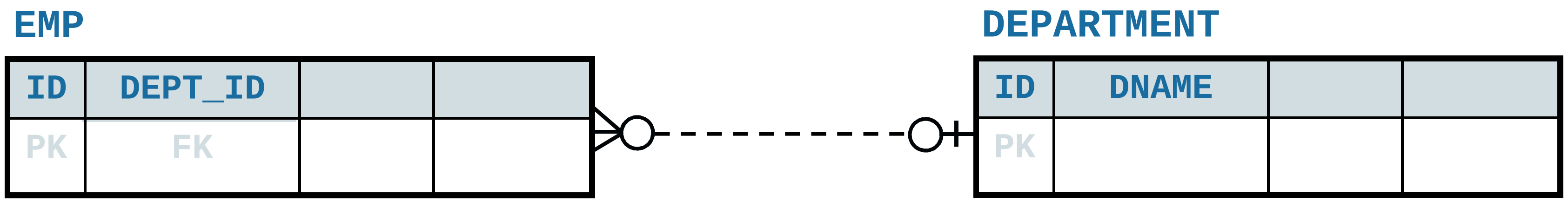
DEPARTMENT

ID	DNAME		
PK			

@OneToMany

```java
@Entity
@Table(name="EMP")
public class Employee {

  @Id
  private int id;

  @ManyToOne
  @JoinColumn(name="DEPT_ID")
  private Department d;


  // getters & setters
  ...

}
```

```java
@Entity
public class Department {

  @Id
  private int id;

  private String dname;

  @OneToMany(mappedBy="d")
  private Collection<Employee> emps;

  // getters & setters
  ...

}
```



EMP

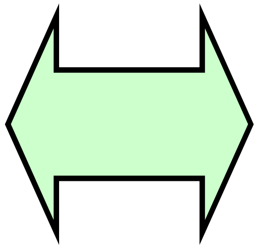
ID	DEPT_ID		
PK	FK		

DEPARTMENT

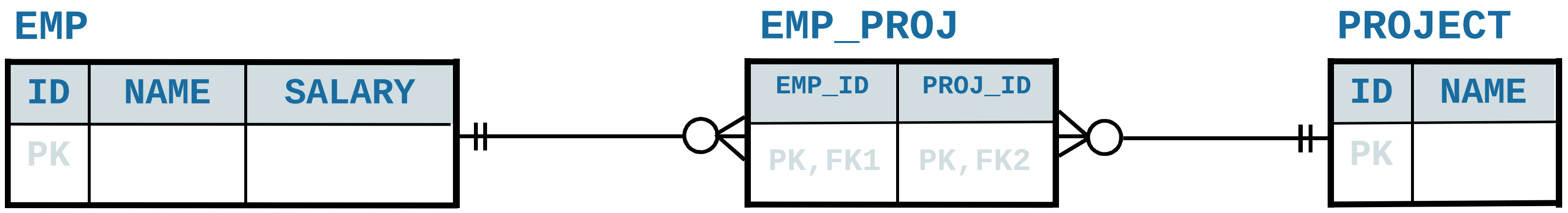
ID	DNAME		
PK			

@ManyToMany

```
@Entity
@Table(name="EMP")
public class Employee {

  @Id
  private int id;


  @JoinTable(name="EMP_PROJ",
      joinColumns=
      @JoinColumn(name="EMP_ID"),
      inverseJoinColumns=
      @JoinColumn(name="PROJ_ID"))
  @ManyToMany
  private Collection<Project> p;
}
```



```
@Entity
public class Project {

  @Id
  private int id;

  private String name;

  @ManyToMany(mappedBy="p")
  private Collection<Employee> e;

  // getters & setters
  ...

}
```

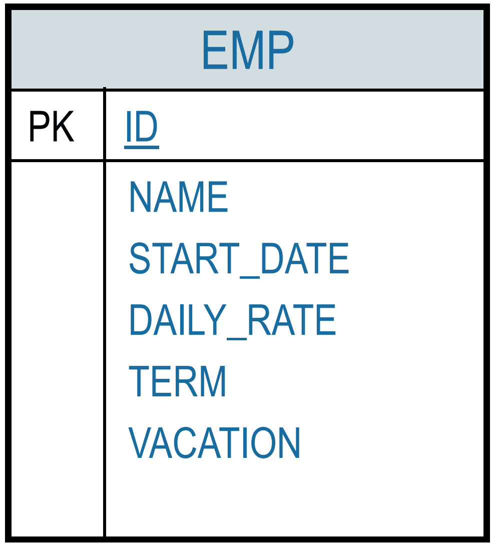


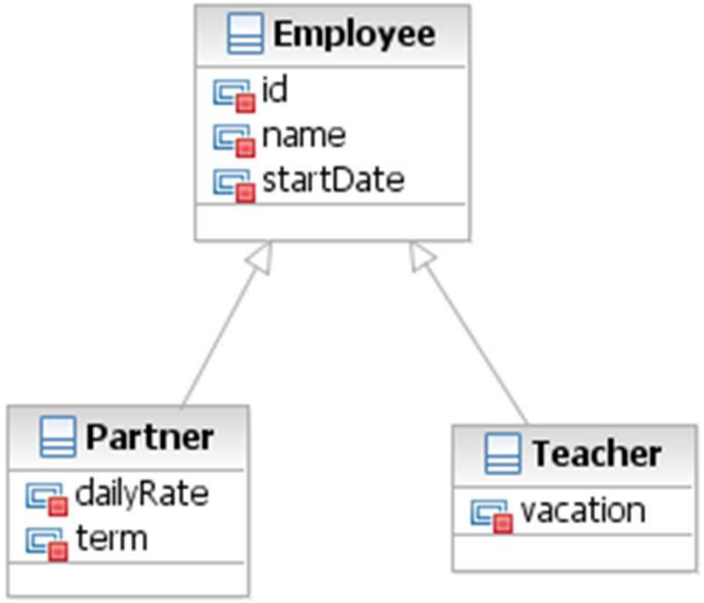
Inheritance

- With JPA you have 3 inheritance strategies:
  - Single table + discriminator
  - Joined tables
  - Table per class

Single table + discriminator (1/2)

- All the hierarchy is in one table. A discrimanator is used to make the distinction.



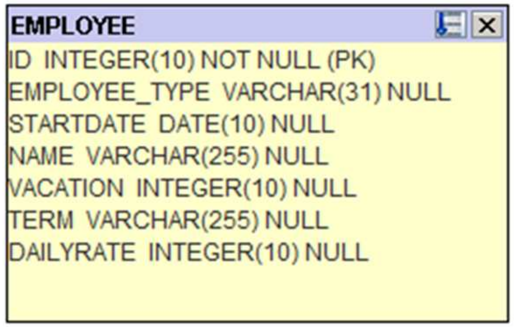


Single table + discriminator (2/2)

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "employee_type")
public class Employee {
   @Id
   protected int id;

   protected String name;

   @Temporal(TemporalType.DATE)
   protected Date startDate;
   //...
}
```

```
@Entity
@DiscriminatorValue(value = "partner")
public class Partner extends Employee{

   private int dailyRate;

   private String term;
   //...
}
```

```
@Entity
@DiscriminatorValue(value = "teacher")
public class Teacher extends Employee{

   private int vacation;
   //...
}
```

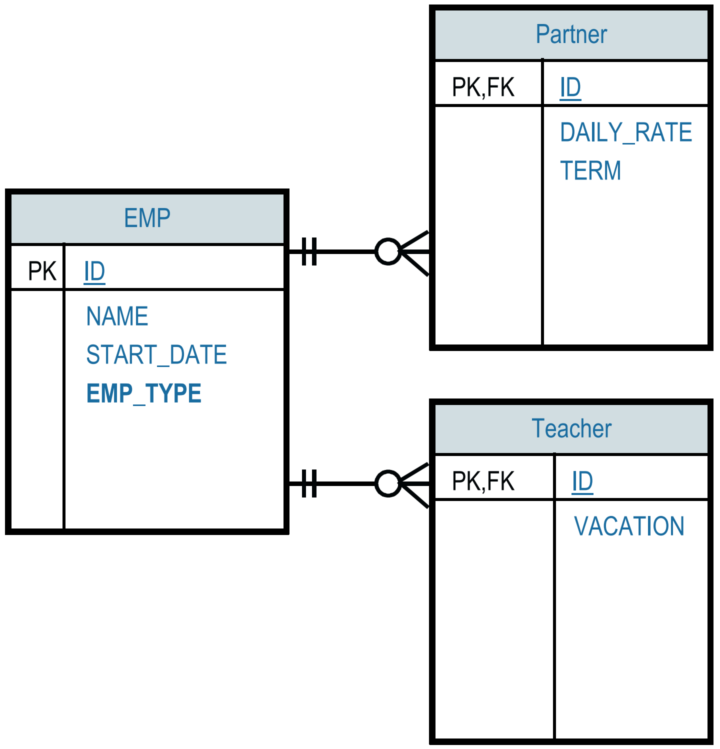


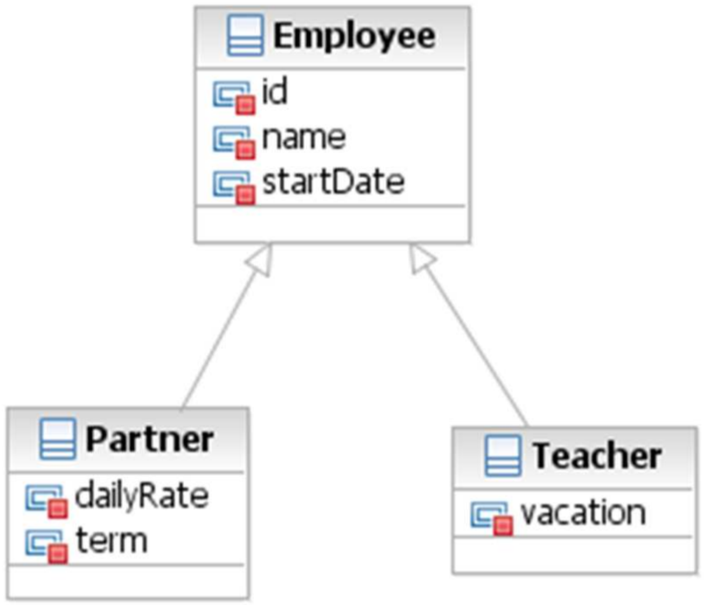
EMPLOYEE

```
ID  INTEGER(10) NOT NULL (PK)
EMPLOYEE_TYPE  VARCHAR(31) NULL
STARTDATE  DATE(10) NULL
NAME  VARCHAR(255) NULL
VACATION  INTEGER(10) NULL
TERM  VARCHAR(255) NULL
DAILYRATE  INTEGER(10) NULL
```

Joined tables (1/2)

- Simulate the hierarchy using distincts, but joined, tables



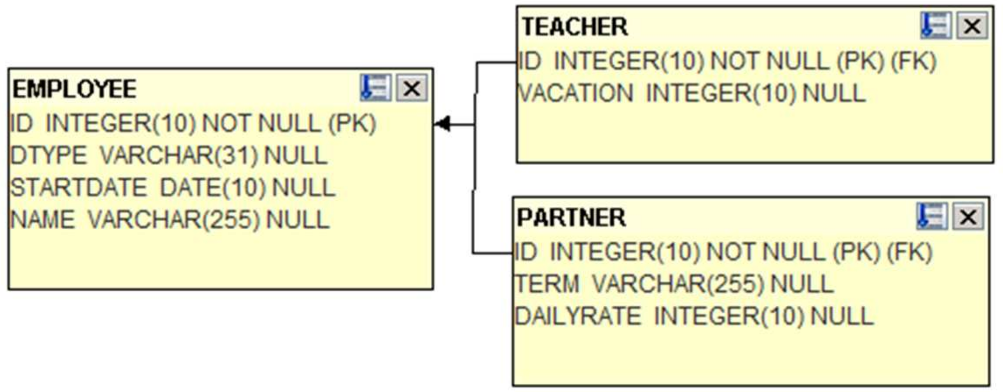


Joined tables (2/2)

```java
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Employee {
   @Id
   protected int id;

   protected String name;

   @Temporal(TemporalType.DATE)
   protected Date startDate;
   //...
}
```

```java
@Entity
@PrimaryKeyJoinColumn(name = "id")
public class Partner extends Employee{

   private int dailyRate;

   private String term;
  //...
}
```

```java
@Entity
@PrimaryKeyJoinColumn(name = "id")
public class Teacher extends Employee{

   private int vacation;

   //...
}
```



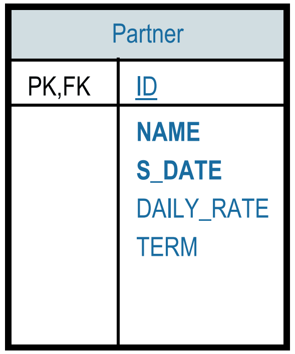
EMPLOYEE
ID INTEGER(10) NOT NULL (PK)
DTYPE VARCHAR(31) NULL
STARTDATE DATE(10) NULL
NAME VARCHAR(255) NULL

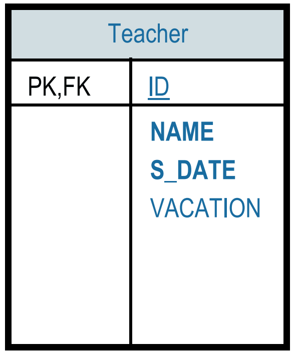
TEACHER
ID INTEGER(10) NOT NULL (PK) (FK)
VACATION INTEGER(10) NULL

PARTNER
ID INTEGER(10) NOT NULL (PK) (FK)
TERM VARCHAR(255) NULL
DAILYRATE INTEGER(10) NULL

Table per class (1/2)

- The leaf classes is represented by a table with all the inherited fields. No super classes here.





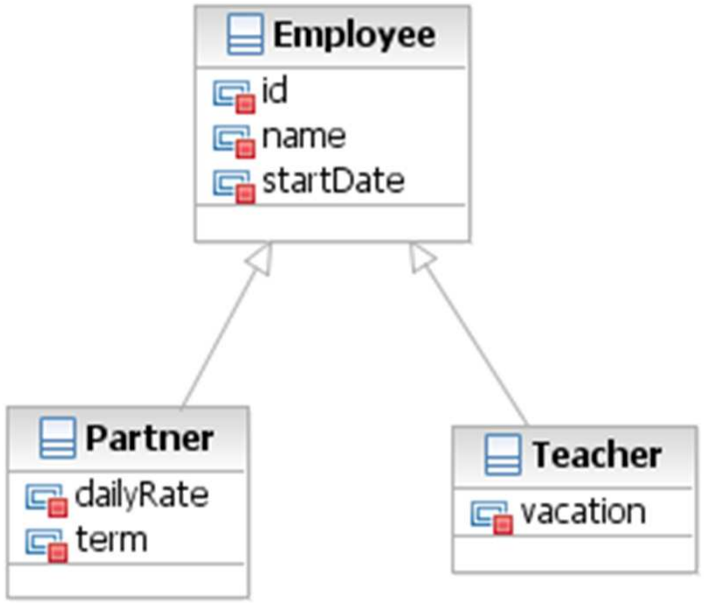
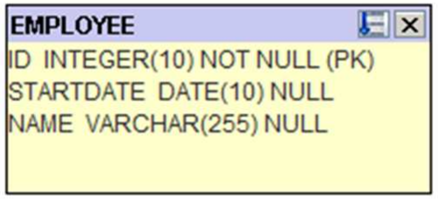


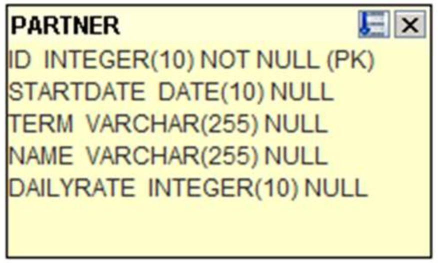
Table per class (2/2)

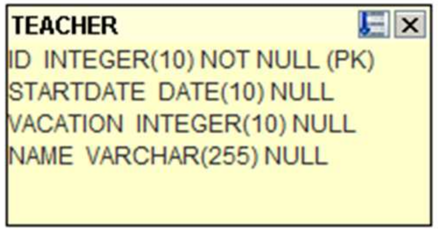
```java
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
@MappedSuperclass
public class Employee {
   @Id
   protected int id;

   protected String name;

   @Temporal(TemporalType.DATE)
   protected Date startDate;
   //...
}
```

```java
@Entity
public class Partner extends Employee {

   private int dailyRate;

   private String term;
   //...
}
```

```java
@Entity
public class Teacher extends Employee {

   private int vacation;
   //...
}
```


EMPLOYEE
ID INTEGER(10) NOT NULL (PK)
STARTDATE DATE(10) NULL
NAME VARCHAR(255) NULL


PARTNER
ID INTEGER(10) NOT NULL (PK)
STARTDATE DATE(10) NULL
TERM VARCHAR(255) NULL
NAME VARCHAR(255) NULL
DAILYRATE INTEGER(10) NULL


TEACHER
ID INTEGER(10) NOT NULL (PK)
STARTDATE DATE(10) NULL
VACATION INTEGER(10) NULL
NAME VARCHAR(255) NULL

orm.xml

- You can define all the O/R mapping in the META-INF/orm.xml

- The XML file will override the annotations values

```xml
<entity-mappings>
<entity class="Customer">
   <id name="id">
      <generated-value/>
   </id>
   <basic name="c_rating">
      <column name="ratings"/>
   </basic>
   ...
   <one-to-many name="orders" mapped-by="cust"/>
</entity>
...
</entity-mappings>
```

Persistence – Key concepts

Persistence Unit (1/2)

- Used to define application persistence properties
  - DB connection information
  - DB Dialect
  - Schema creation parameters
- **Persistence.xml** defines one or more persistence units
- The JAR file that contains **persistence.xml** will be scanned for any classes annotated with @Entity
- A set of managed entities, belonging to a single persistence unit is called **Persistence Context**

Persistence Unit (2/2) - SimpleAppShop

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
              http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

   <persistence-unit name="simpleAppShopPU" transaction-type="RESOURCE_LOCAL">

      <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

      <class>fr.danielpetisme.javaee.jpa_example.Download</class>
      <class>fr.danielpetisme.javaee.jpa_example.Application</class>
      <class>fr.danielpetisme.javaee.jpa_example.Member</class>
      <class>fr.danielpetisme.javaee.jpa_example.Platform</class>
      <class>fr.danielpetisme.javaee.jpa_example.Administrator</class>

      <properties>
         <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.ClientDriver"/>
         <property name="javax.persistence.jdbc.url"
                   value="jdbc:derby://localhost:1527/SimpleAppShop.db;create=true"/>
         <property name="javax.persistence.jdbc.user" value="app"/>
         <property name="javax.persistence.jdbc.password" value="app"/>
      </properties>

   </persistence-unit>
</persistence>
```

1 Persistence Unit

Persistence provider

The entities

The DB connection properties

EntityManager (1/2)

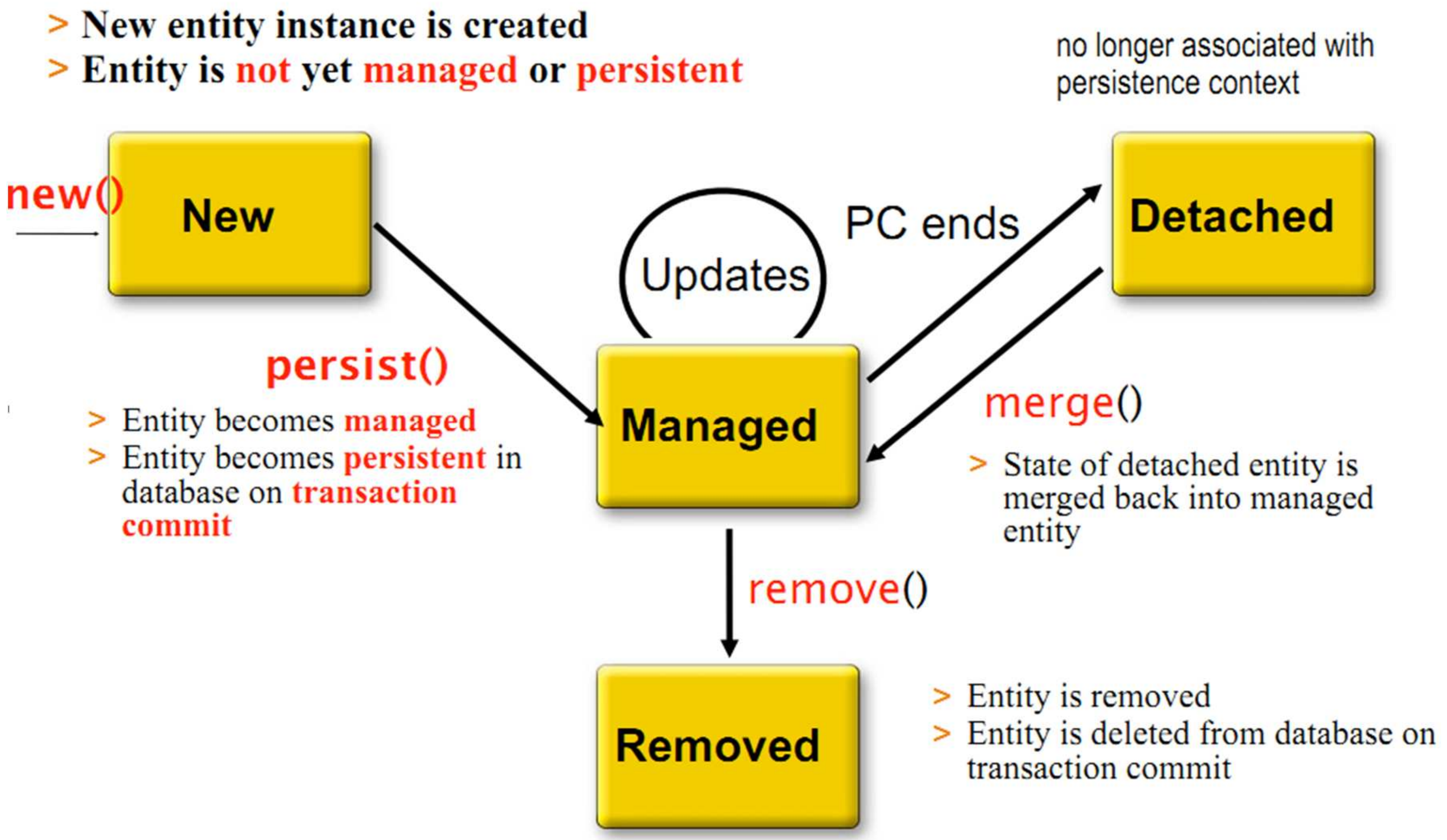
- API to manage entity lifecycle. Is the gateway to persistence functions

- **persist**(Obect o)
  - Saves or updates the specified object tree
- **remove**(Object o)
  - Deletes the specified object
- **find**(Class type, Serializable id)
  - Retrieves the item of the specified type by id
- **merge**(Object o)
  - Attaches a detached instance to the manager (required for update on item retrieved from a different manager)
- **getTransaction**()
  - Provides a transaction object to perform commit and rollback functions

EntityManager (2/2)

```java
//EntityManager bootstrap
EntityManagerFactory factory = Persistence.createEntityManagerFactory("myPersistenceUnitName");
EntityManager em = factory.createEntityManager();

 //Begin a transaction
em.getTransaction().begin();

 //Create a POJO, it's out of the persistence context
Computer c = new Computer();
c.setModel("Lenovo L420");
c.setOperatingSystem("Windows XP");

try {
   em.persist(c);
   //Now the object is in the persistence context
   LOGGER.log(Level.INFO, "The computer id is now: {0}", c.getId());
} catch (EntityExistsException e) {
   LOGGER.log(Level.INFO, "The computer already exists", e);
   //We can manually rollback the transaction if we have a problem
   em.getTransaction().rollback();
}

c.setOperatingSystem("Windows 7");
//Merge an object means synchronize the object instance with its DB representation
c = em.merge(c);
LOGGER.log(Level.INFO, "The computer OS is now: {0}", c.getOperatingSystem());

//Remove the object from the persistence context, it doesn't mean the object is deleted from the memory
em.remove(c);
LOGGER.log(Level.INFO, "The computer no longer exists in the DB");

//Since we don't use a Java EE container, we have to manually handle the transaction
em.getTransaction().commit();

//Never ever forget to close the resources!!
em.close();
factory.close();

//The object isn't in the persistence context, but it still live
LOGGER.log(Level.INFO, "Hi, I am the computer {0} and I am detached of the persistence context", c.getModel());
```

Entity Lifecyle



> New entity instance is created
> Entity is not yet managed or persistent
no longer associated with persistence context
new()
New
Updates
PC ends
Detached
persist()
> Entity becomes managed
> Entity becomes persistent in database on transaction commit
Managed
merge()
> State of detached entity is merged back into managed entity
remove()
Removed
> Entity is removed
> Entity is deleted from database on transaction commit

Cascading operations

- How to propagate the operation to the related objects

- CascadeType values:
  (ALL|REMOVE|DETACHE|MERGE|PERSIST|REFRESH)

```java
@Entity
public class Computer {
   @Id
   private int id;
   @OneToMany(mappedBy = "computer", cascade = CascadeType.PERSIST)
   private List<Account> accounts;

   //...
}
```

Queries – Native queries

- JPA allows you to write SQL native queries.

```java
Query query = em.createNativeQuery("SELECT * FROM PLATFORM", Platform.class);
List<Platform> values = query.getResultList();

for (Platform p : values) {
   LOGGER.log(Level.INFO, p.getPlatformName());
}
```

Queries – JPQL

- JPA defines its own object-oriented query language.

```java
//Simple Query + Object Navigation
Query query1 = em.createQuery("SELECT a FROM Application a");
List<Application> values1 = query1.getResultList();
for (Application a : values1) {
   LOGGER.log(Level.INFO, a.getPlatform().getPlatformName());
}

//Parametered Query
Query query2 = em.createQuery("SELECT a FROM Application a"
      + " WHERE a.platform.platformName = :paramName"
      + " AND a.platform.platformVersion = ?1");
query2.setParameter("paramName", "Windows");
query2.setParameter(1, "XP");
List<Application> values2 = query2.getResultList();

for (Application a : values2) {
   LOGGER.log(Level.INFO, a.getApplicationName());
}
```

Queries – Named queries

- You can define some basics queries in your entities and use them in your client app.

```java
@Entity
@Table(name = "MEMBER", schema = "APP")
@NamedQuery(name = "MemberUser.findAll", query = "SELECT m FROM MemberUser m")
public class MemberUser {

   //...
}
```

```java
//Named query
Query query3 = em.createNamedQuery("MemberUser.findAll");
List<MemberUser> values3 = query3.getResultList();

for (MemberUser m : values3) {
   LOGGER.log(Level.INFO, m.getMemberUsername());
}
```

Queries - Criteria

- You can use the Criteria API to build
  - Pro: Syntaxic verification during the compilation
  - Con: Much more « touchy » to masterize

```java
//JPQL
Query query = em.createQuery("SELECT m FROM MemberUser m"
                  + " WHERE m.memberUsername = 'danielpetisme'");

///Criteria API
CriteriaBuilder criteriaBuilder = em.getCriteriaBuilder();
CriteriaQuery criteriaQuery = criteriaBuilder.createQuery(MemberUser.class);
Root member = criteriaQuery.from(MemberUser.class);
criteriaQuery.select(member).where(criteriaBuilder.equal(
    member.get(MemberUser_.memberUsername), "danielpetisme"));
```

The entity metamodel

In a nutshell

- JPA is the Java Standard ORM based on POJOs
- Provide a large set of annotation to ease the O/R mapping (Table, Column, Association)
- Convention over configuration principle
- The EntityManager API allows you to manage entity lifecyle in a persistence context defined by a persistence unit (persistence.xml)
- Many ways to retrieve data (native, JPQL, Criteria)

Questions?
Carte
PERMANENT

Credits

- http://www.moma.org/collection_images/resized/051/w500h420/CRI_151051.jpg
- http://upload.wikimedia.org/wikipedia/en/thumb/8/80/Wikipedia-logo-v2.svg/200px-Wikipedia-logo-v2.svg.png
- http://icons.iconarchive.com/icons/untergunter/leaf-mimes/512/text-xml-icon.png
- http://blog.kadeal.com/wp-content/uploads/2011/12/silver-database-icon.jpg
- http://www.slideshare.net/jsbournival/presentation-jpa