

## Historia de JDO

- Breve repaso

- Versión 1.0: JSR 12, Final release 30/04/2002
- Versión 2.0: JSR 243, Final release 10/05/2006
- Versión 2.1: Final release 02/2008
- Versión 2.2: Final release 10/2008
- Versión 3.0: Final release 04/2010

## Historia de JPA

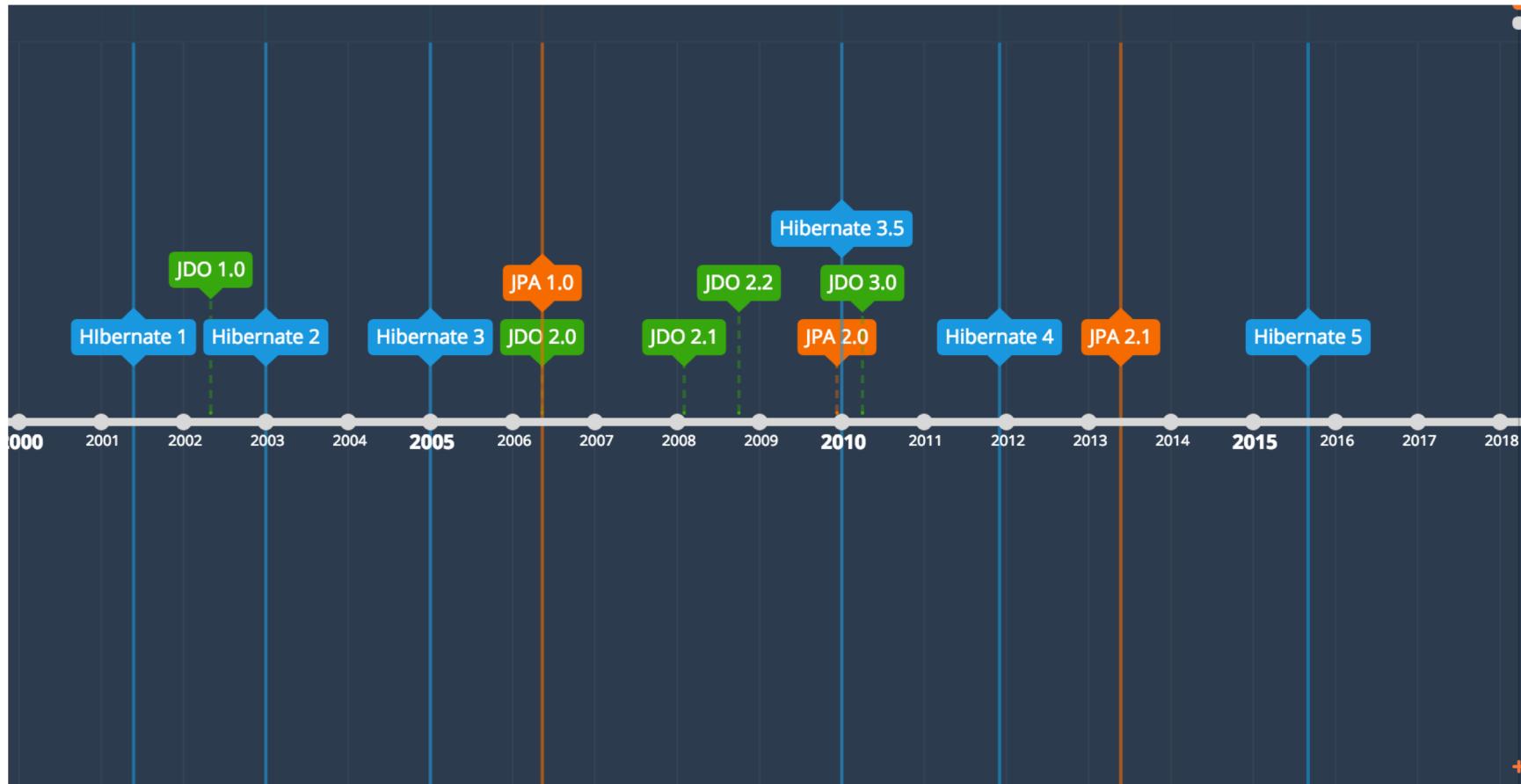
- Breve repaso

- Versión 1.0: JSR 220, Final release 11/05/2006
- Versión 2.0: JSR 317, Final release 10/12/2009
- Versión 2.1: JSR 338, Final release 22/05/2013

## Historia de Hibernate

- Breve repaso
  - Versión 1.0: Final release 23/05/2001
  - Versión 2.0: Principios de 2003
  - Versión 3.X: Año 2005
    - Versión 3.5: Año 2010 implementación de referencia de JPA 2.0
  - Versión 4: Diciembre 2011
  - Versión 5: Septiembre 2015

## Línea de tiempo



# Comparación entre las alternativas

## • Implementaciones de JDO

Name	License	JDO Spec	Datastore(s)
<a href="#">DataNucleus Access Platform</a>	NonCommercial	1.0, 2.0, 2.1, 2.2, 3.0, 3.1	RDBMS, db4o, NeoDatis, LDAP, Excel XLS, Excel OOXML, ODF, XML, JSON, Google BigTable, HBase, Amazon S3, MongoDB, GoogleStorage, Cassandra, OrientDB, Salesforce.com, Neo4j
<a href="#">JDOInstruments</a>	NonCommercial	1.0	JDOInstruments
<a href="#">JPOX</a>	NonCommercial	1.0, 2.0, 2.1	RDBMS, db4o
<a href="#">Kodo</a>	Commercial	1.0, 2.0	RDBMS, XML
<a href="#">ObjectDB for Java/JDO</a>	Commercial	1.0, 2.0	ObjectDB
<a href="#">Objectivity</a>	Commercial	1.0	ObjectivityDB
<a href="#">Orient</a>	Commercial	1.0	Orient
<a href="#">hywy's PE:J</a>	Commercial	1.0	RDBMS
<a href="#">SignSoft IntelliBO</a>	Commercial	1.0	intelliBO
<a href="#">Speedo</a>	NonCommercial	1.0	RDBMS
<a href="#">TJDO</a>	NonCommercial	1.0	RDBMS
<a href="#">Versant</a>	Commercial	1.0, 2.0	Versant Object Database
<a href="#">Xcalia</a>	Commercial	1.0, 2.0	RDBMS, XML, Versant ODBMS, Jalisto, Web services, mainframe transactions and screens (CICS, IMS...), packaged applications (ERP, CRM, SFA...), components (EJB...).

## Comparación entre las alternativas

- **Implementaciones de JPA**

Hibernate	4.3.0	JPA 2.1
Eclipse Link	2.5.1	JPA 2.1
OpenJPA	2.2.2	JPA 2.0
BatooJPA	2.0.1.2	JPA 2.0
DataNucleus	3.0	JPA 2.1
ObjectDB	2.5.3	JPA 2.0
Versant JPA	1.0.2	JPA 2.0

## JPA

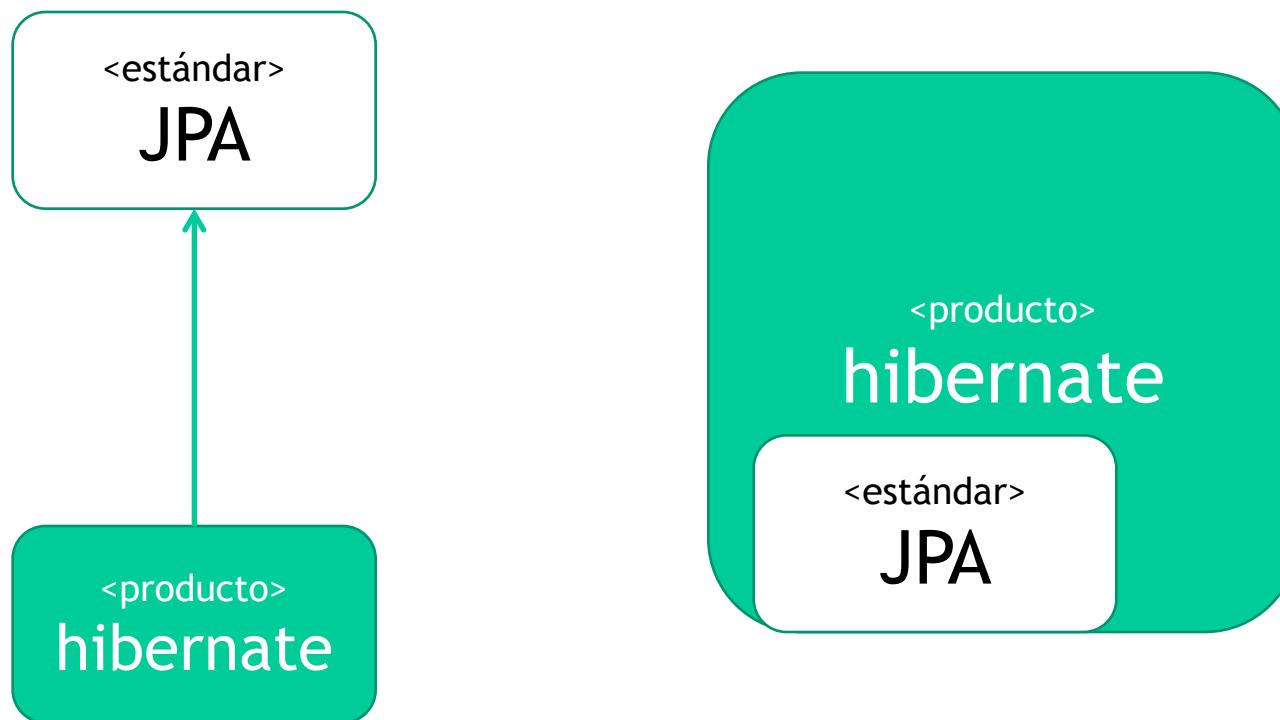
- JPA (Java Persistence API) es una especificación independiente para mapear objetos
- Originalmente estaba pensado exclusivamente para bases de datos relacionales.
- Actualmente se puede mapear a diferentes tipos de repositorios, por ejemplo a MongoDB (NoSQL).

## JPA

- JPA está compuesto de 3 diferentes partes
  - Entidades
    - Actualmente son clásicos P.O.J.O.s
    - Antes se requería extender de ciertos tipos de JPA.
  - Meta-data objeto-relacional
    - Mediante archivos de configuración (xml)
    - Mediante annotations
  - Java Persistence Query Language (JP-QL)
    - Tiene como objetivo lograr abstracción del producto particular que se utiliza para resolver la consulta

# JPA

- Relación entre JPA y Hibernate



## JPA

- Toda interacción en JPA se realiza a través de un “EntityManager”.
- Proceso de trabajo:
  - Crear un EntityManagerFactory (normalmente uno solo por “unidad de persistencia”).
  - Obtener una instancia de EntityManager a partir del factory.
  - Todos los aspectos de la configuración se realizan a través del archivo “persistence.xml” (debe estar en el directorio META-INF).

# JPA

- Persistence.xml

```
01 <?xml version="1.0" encoding="UTF-8" ?>
02 <persistence xmlns="http://java.sun.com/xml/ns/persistence"
03     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
05 http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
06
07     <persistence-unit name="PersistenceUnit" transaction-type="RESOURCE_LOCAL">
08         <provider>org.hibernate.ejb.HibernatePersistence</provider>
09         <properties>
10             <property name="connection.driver_class" value="org.h2.Driver"/>
11             <property name="hibernate.connection.url" value="jdbc:h2:~/jpa"/>
12             <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
13             <property name="hibernate.hbm2ddl.auto" value="create"/>
14             <property name="hibernate.show_sql" value="true"/>
15             <property name="hibernate.format_sql" value="true"/>
16         </properties>
17     </persistence-unit>
18 </persistence>
```

# JPA

- Creación de un EntityManager

```
01 EntityManagerFactory factory = null;
02 EntityManager entityManager = null;
03 try {
04     factory = Persistence.createEntityManagerFactory("PersistenceUnit");
05     entityManager = factory.createEntityManager();
06     persistPerson(entityManager);
07 } catch (Exception e) {
08     LOGGER.log(Level.SEVERE, e.getMessage(), e);
09     e.printStackTrace();
10 } finally {
11     if (entityManager != null) {
12         entityManager.close();
13     }
14     if (factory != null) {
15         factory.close();
16     }
17 }
```

# JPA

- Transacciones

```
01 private void persistPerson(EntityManager entityManager) {  
02     EntityTransaction transaction = entityManager.getTransaction();  
03     try {  
04         transaction.begin();  
05         Person person = new Person();  
06         person.setFirstName("Homer");  
07         person.setLastName("Simpson");  
08         entityManager.persist(person);  
09         transaction.commit();  
10     } catch (Exception e) {  
11         if (transaction.isActive()) {  
12             transaction.rollback();  
13         }  
14     }  
15 }
```

# JPA

- Meta-data

```
01 @Entity
02 @Table(name = "T_PERSON")
03 public class Person {
04     private Long id;
05     private String firstName;
06     private String lastName;
07
08     @Id
09     @GeneratedValue
10     public Long getId() {
11         return id;
12     }
13
14     public void setId(Long id) {
15         this.id = id;
16     }
17
18     @Column(name = "FIRST_NAME")
19     public String getFirstName() {
20         return firstName;
21     }
22
23     public void setFirstName(String firstName) {
24         this.firstName = firstName;
25     }
26
27     @Column(name = "LAST_NAME")
28     public String getLastName() {
29         return lastName;
30     }
31
32     public void setLastName(String lastName) {
33         this.lastName = lastName;
34     }
35 }
```

# JPA

- Meta-data

```
/**
 * Get the users responsible for this story item.
 * @return Set of the responsible users
 */
@ManyToMany(
    targetEntity = fi.hut.soberit.agilefant.model.User.class
)
@JoinTable(
    name = "story_user",
    joinColumns={@JoinColumn(name = "Story_id")},
    inverseJoinColumns={@JoinColumn(name = "User_id")}
)
@BatchSize(size=20)
@Fetch(FetchMode.SUBSELECT)
@XmlElementWrapper
@XmlElement(name = "user")
public Set<User> getResponsibles() {
    return responsibles;
}
```

## JPA

- Tipos de relaciones soportadas
  - One to one
  - One to many / Many to one
  - Many to many
  - Embedded
  - ElementCollection

# JPA

- Consultas

- JPQL

```
1 entityManager.createQuery("from Project p where p.projectPeriod.startDate = :startDate",
  Project.class).setParameter("startDate", createDate(1, 1, 2015));
```

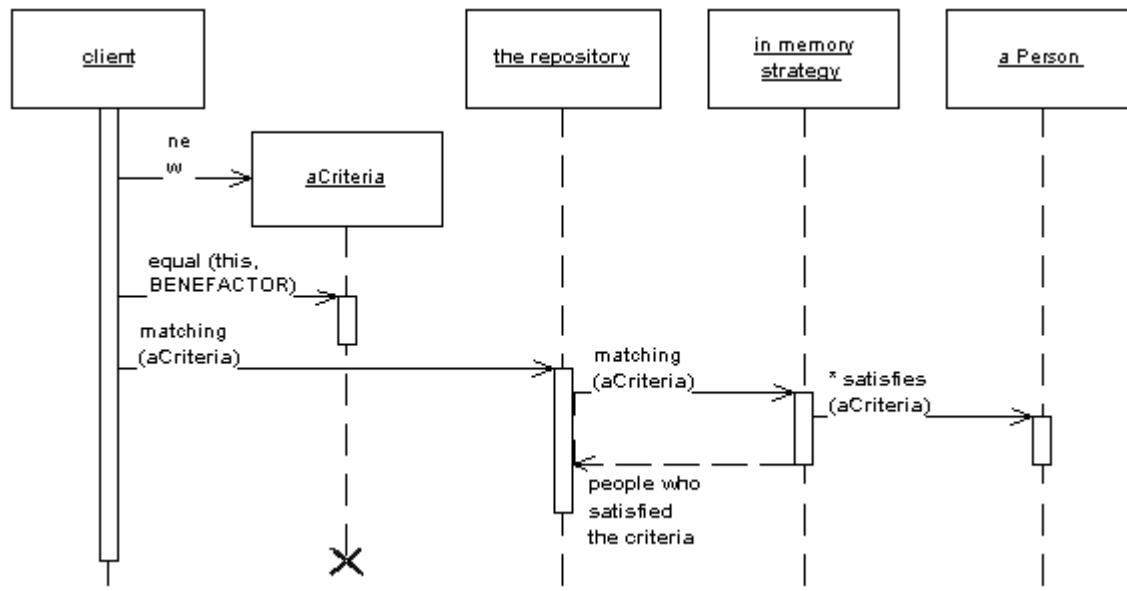
- Criteria API

```
1 CriteriaBuilder builder = entityManager.getCriteriaBuilder();
2 CriteriaQuery<Person> query = builder.createQuery(Person.class);
3 Root<Person> personRoot = query.from(Person.class);
4 query.where(builder.equal(personRoot.get("firstName"), "Homer"));
5 List<Person> resultList = entityManager.createQuery(query).getResultList();
```

# Patrones

- Repository

- Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.



# Patrones

- **Repository**

- La implementación en Java puede ser considerable

- C** **HibernateIndividualsRepository**

- $\Delta$  `findIndividualByEmail(String) : Individual`
  - $\Delta$  `findIndividuals(int, long, String, String, boolean, String) : Collection<Individual>`
  - $\Delta$  `getIndividualsCount(boolean, String) : long`

- C** **HibernateUsersRepository**

- $\Delta$  `findUserByEmail(String) : User`
  - $\Delta$  `findUserByName(String) : User`
  - $\Delta$  `findUsers(int, long, String, String, boolean) : Collection<User>`
  - $\Delta$  `getUsersCount(boolean, String) : long`

- ▼ C** **HibernateZoigenRepository**

- $\Delta$  `findExtractionCenter(String) : ExtractionCenter`
  - $\Delta$  `findServiceDefinitionByName(String) : ServiceDefinition`
  - $\Delta$  `findServiceDefinitions(int, long, String, String) : Collection<ServiceDefinition>`
  - $\Delta$  `findServiceRequestById(String) : ServiceRequest`
  - $\Delta$  `findServiceRequests(int, long, String, String, String) : Collection<ServiceRequest>`
  - $\Delta$  `findServiceRequestsByIndividual(String) : Collection<ServiceRequest>`
  - $\Delta$  `findZoigen() : Zoigen`
  - $\Delta$  `getNextId() : long`
  - $\Delta$  `getServiceDefinitionsCount() : long`
  - $\Delta$  `getServiceRequestsCount(String) : long`

## Spring Data

- Es un proyecto que engloba muchos subproyectos de Spring y de la comunidad.
- Su objetivo es proveer una interface consistente para manejar datos abstrayendo las particularidades de cada motor de persistencia.
  - Spring Data JPA
  - Spring Data JDBC
  - Spring Data Cassandra
  - Spring Data Mongodb
  - Spring Data LDAP
  - Spring Data Redis
  - ...

## Spring Data

- **Repository**

- El principal objetivo de este módulo es reducir el código requerido para crear las capas de acceso a los datos.
- El concepto central de este módulo es la interface “Repository” que se configura con la clase de dominio a manejar y el tipo del identificador.

## Patrones

- Spring Data Repository

```
public interface CrudRepository<T, ID extends Serializable>
    extends Repository<T, ID> {

    <S extends T> S save(S entity);

    T findOne(ID primaryKey);

    Iterable<T> findAll();

    Long count();

    void delete(T entity);

    boolean exists(ID primaryKey);

    // ... more functionality omitted.
}
```

# Patrones

- Spring Data Repository

```
public interface PersonRepository extends Repository<User, Long> { ... }
```

```
List<Person> findByLastname(String lastname);
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">

    <repositories base-package="com.acme.repositories" />

</beans>
```

```
public class SomeClient {

    @Autowired
    private PersonRepository repository;

    public void doSomething() {
        List<Person> persons = repository.findByLastname("Matthews");
    }
}
```