HIBERNATE

Sesión 2: Mapeado de clases persistentes

Puntos a tratar

- Clases persistentes
- Modelo de programación de clases persistentes
- Modelos de objetos de "grano fino"
- Colecciones de valores
- Sesiones y transacciones
- Operaciones con objetos persistentes

Clases persistentes

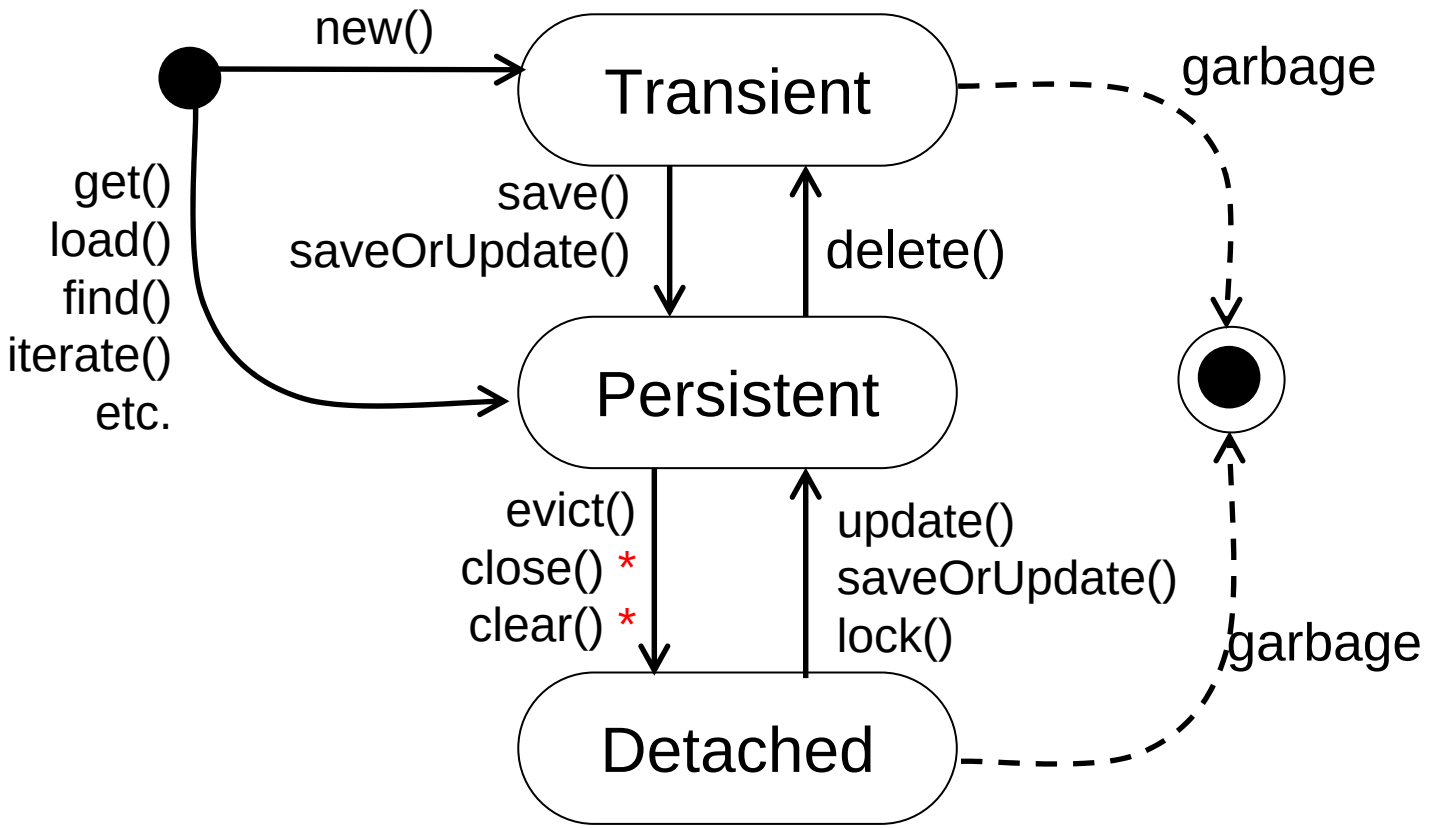
- Son clases de la capa de persistencia que implementan las entidades del negocio del problema en forma de POJOs (Plain Old Java Objects)
- Ejemplo de POJO:

```
public class Customer {
    private Long id = null;
    private CLocation customerLocation;
    public Customer() {}

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

    public CLocation getCustomerLocation() {
                        return customerLocation;}
    public void setCustomerLocation(CLocation
            customerLocation) {
          this.customerLocation = customerLocation;}
}
```

Ciclo de vida de las clases persistentes



* afecta a todas las instancias de una Session

Modelo de programación de clases persistentes

- Identificar las entidades de negocio del dominio del problema
- Crear el modelo del dominio (modelo conceptual de las entidades y sus atributos)
- Implementar el modelo del domino utilizando POJOs (una clase persistente para cada entidad)
- Definir los metadatos de la correspondencia (mapeado) entre clases Java y las tablas de la BD

Mapeado de clases persistentes

CABECERA

```
<hibernate-mapping package="nombre">
   <class name="Nombre_clase"
          table="nombre_tabla">
     <id name="nombre_clave_primaria">
       <generator class="nombre_clase_generadora_ids"/>
     </id>
     <property name="nombrePropiedad"
       type="date"
       not-null="true"
       update="false"/>
      <property name=... />
     ...
     <sql-query name=myQuery>
       SELECT * FROM PERSON/>
     ...
   </class>
```

Modelo de objetos de "grano fino"

- "Grano fino" = Más clases que tablas
- Una fila de una tabla puede representar múltiples objetos
- En Java, todas las clases tienen el mismo "status"
- Hibernate distingue entre:
  - Tipo **entity**: tiene su identidad en la BD (`<class/>`)
  - Tipo **value**: No tiene identidad en la BD (`<property type=.../>`, `<component type=.../>`,...

Hibernate *mapping* type

- Hibernate actúa de "puente" entre el sistema de tipos Java y los tipos de base de datos SQL
- El valor del atributo *type* es el nombre de un *Hibernate mapping type*
- Ejemplos de tipos de mapeado Hibernate básicos:
  - integer, long, short, float, double, string, date
  - Cada uno de ellos tiene su correspondiente tipo constante definido en *org.hibernate.Hibernate*: `Hibernate.INTEGER, Hibernate.LONG, ...`

Colecciones de valores

```
public class Usuario {
...
   private Set telefonos = new HashSet();
   public Set getTelefonos() { return telefonos; }
   public void setTelefonos (Set telefonos) { this.telefonos = telefonos; }
}
```

value type

- Mapeado Hibernate:

```
<class name="Usuario>
    <set name="telefonos" table="USU_TELEFONOS"
                          lazy="false">
        <key column="USUARIO_ID"/>
        <element type="string" column="TELEFONOS"/>
    </set>
</class>
```

clave ajena

tipo Hibernate

Sesiones y transacciones

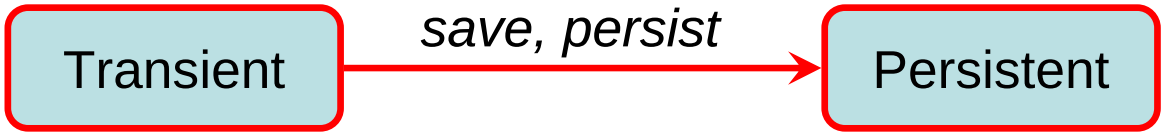
- **Sesión**: unidad de trabajo que engloba un conjunto de operaciones con la BD.
- **Transaccion**: conjunto de operaciones que deben realizarse todas, o ninguna.

```
...
Session se = factory.openSession();
Transacion tx = null;
try { tx = se.beginTransaction();
  // hacer algo ...
     tx.commit();
} catch (RuntimeException e) {
    if (tx!= null) tx.rollback();
    throw e;
} finally { sess.close(); }
...
```

propiedad *(Hibernate.cfg.xml)*

```
//current_session_context_class=thread
try {
    sessionFactory.getCurrentSession()
                  .beginTransaction();
    // hacer algo ...
    sessionFactory.getCurrentSession()
          .getTransaction().commit();
} catch(RuntimeException e) {
    sessionFactory.getCurrentSession()
    .getTransaction().rollback();
    throw e;} ...
```

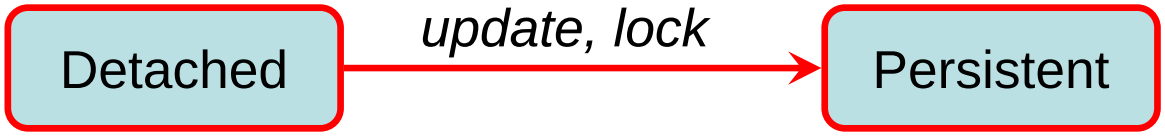
Operaciones con objetos persistentes




```
...
Usuario user = new Usuario(); //user aquí es transient (no persistente)
user.getNombre().setNombre("Juan");
user.getNombre().setApellidos("Penalva");
Session session= sessions.openSession(); //sessions es de tipo SessionFactory
Transaction tx = session.beginTransaction();
session.save(user);
tx.commit(); //sincronización con la BD
session.close(); //liberamos la conexión con la BD
...
```

Los cambios realizados sobre los objetos persistentes se sincronizan cuando hacemos *commit()*

Operaciones con objetos persistentes



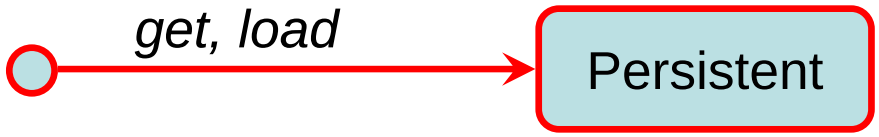

```
user.setClave("secret");
Session seDos = sf.openSession();
Transaction tx =
    seDos.beginTransaction();
seDos.update(user);
user.setLogin("Juanito");
tx.commit();
seDos.close();
```

```
Session seDos = sf.openSession();
Transaction tx =
   seDos.beginTransaction();
seDos.lock(user, LockMode.NONE);
user.setClave("secret");
user.setLogin("Juanito");tx.commit();
seDos.close();
```

No importa si los cambios se hacen antes o después de llamar a ***update***

Los cambios hechos antes de la llamada a ***lock*** no se propagan a la base de datos

Operaciones con objetos persistentes




```
Session se = sf.openSession();
Transaction tx = se.beginTransaction();
int userID = 1234;
Usuario user = (Usuario) se.get(Usuario.class, new Long(userID);
tx.commit();
session.close();
```

El método *load* nunca devuelve null

El método *get* devuelve *null* si el objeto no puede encontrarse.

Automatic dirty checking

- Hibernate sigue la pista y se anota los cambios realizados sobre un objeto persistente mientras siga estando en dicho estado, es decir, mientras siga dentro de una sesión.

```
Session session = sessions.openSession();
Transaction tx = session.beginTransaction();
int userID = 1234;
Usuario user = (Usuario) session.get(Usuario.class, new Long(userID);
user.setClave("secret");
tx.commit();
session.close();
```

no es necesario llamar a "*save*" de forma explícita

Operaciones con objetos persistentes


```
Session session = sessions.openSession();
Transaction tx = session.beginTransaction();
int userID = 1234;
Usuario user = (Usuario) session.get(Usuario.class, new Long(userID);
session.delete(user);
tx.commit();
session.close();
```

Después de cerrar la *Session*, el objeto *user* se considera como una instancia *transient* ordinaria

Operaciones con objetos persistentes


```
Session session = sessions.openSession(); T
ransaction tx = session.beginTransaction();
session.delete(user);
tx.commit();
session.close();
```

La llamada a *delete()* hace dos cosas:

- asocia el objeto con la *Session*
- y a continuación incluye dicho objeto para ser borrado cuando realicemos la operación *tx.commit()*

¿Preguntas...?