PERSISTENCIA ORIENTADA A OBJETOS

CONCEPTOS FUNDAMENTALES

JDBC

```
publc class Concurso {
        private String id;
        private String nombre;
        ...
}
public void ejecutarInsert(Concurso concurso) {
  try (Connection conn = obtenerConexionBD();
    PreparedStatement statement = conn.prepareStatement("insert into
                              concurso(id, nombre) values ('ID1', ?)");) {
    statement.setString(1, concurso.nombre());
    statement.executeUpdate();
  } catch (SQLException ex) {
    throw new RuntimeException(ex);
  }
}
```

Speaker notes

De esta forma vemos los objetos como estructura de datos, obteniendo su estado interno via getters para generar sentencias SQL. Uno tiende a poner lógica de negocios en los métodos DAO de esta forma y no debería ser así.

PERSISTENCIA ORIENTADA A OBJETOS

¿No hay una forma mejor?

Transparencia: El desarrollador tendría que preocuparse lo menos posible por la persistencia para centrarse en el modelo de objetos. Tenemos que trabajar con los objetos en memoria pensando que la persistencia se maneja "mágicamente".

PERSISTENCIA ORIENTADA A OBJETOS

¿Hay magia?

PERSISTENCIA POR ALCANCE

- Objeto al cual se puede llegar a partir de un objeto ya persistente, debe ser necesariamente persistente.
- Esto nos regala una forma elegante de diseñar, ya que para persistir un objeto solo tenemos que vincularlo a otro ya persistente.

PERSISTENCIA ORIENTADA A OBJETOS

PERSISTENCIA POR ALCANCE

A ver la magia...

```
public class Universidad {
  private Set<Concurso> cs = new HashSet<>();
  public void nuevoConcurso(Concurso c) {
    cs.add(c);
  }
  ...
}
public void nuevoConcurso(String nombreConcurso) {
  Concurso c = new Concurso(nombreConcurso);
  Universidad unrn = //contexto.recuperar(...) unrn de la base de datos
  unrn.nuevoConcurso(c);
```

Speaker notes

Notar que hacer la instancia de Concurso c persistente es totalmente transparente. No es tan transparente el traer de la BD a unrn.


MAKE GIFS AT GIFSOUP.COM

PERSISTENCIA ORIENTADA A OBJETOS

¿QUÉ MÁS TENEMOS QUE SABER?

Ciclo de Vida de los Objetos Persistentes

CICLO DE VIDA DE LOS OBJETOS PERSISTENTES

¿Cual es el ciclo de vida de los Objetos? (olvidemos la persistencia por un ratito)

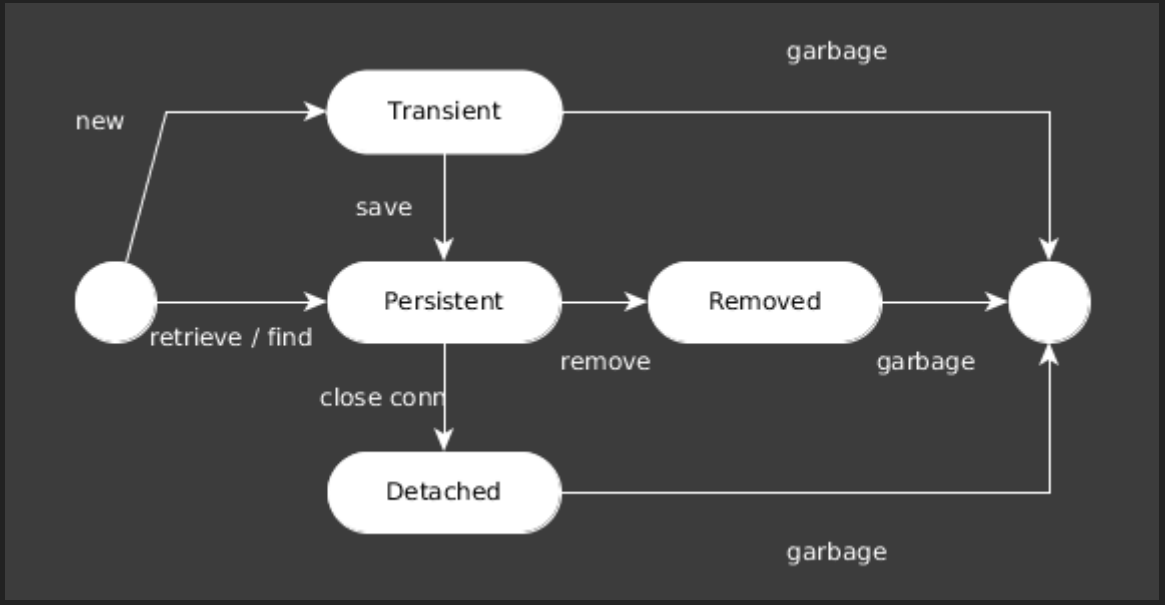
new: Se crea la instancia y se aloja en memoria

delete: El Garbage Colector se encarga

Al incorporar persistencia se agregan nuevos estados.

PERSISTENCIA ORIENTADA A OBJETOS

Ciclo de Vida de los Objetos Persistentes



PERSISTENCIA ORIENTADA A OBJETOS

¿Quién lleva registro en memoria del estado de los objetos persistentes?

CONTEXTO DE PERSISTENCIA

- Maneja y monitorea en memoria todas las instancias de las clases persistentes.
- Reconoce cuando una instancia fue modificada para persistirla.
- Básicamente... es quien se encarga de la magia!

PERSISTENCIA ORIENTADA A OBJETOS

Veamos un ejemplo en JPA (Java Persistent API)

```
public void nuevoConcurso(String nombreConcurso) {
 EntityManager manager = EntityManagerFactory.createEntityManager();
 Concurso concurso = new Concurso(nombreConcurso);
 Universidad unrn = manager.find(idUnrn);
 unrn.nuevoConcurso(concurso);
 manager.close(); //Aca se persiste lo modificado...
}
```

¿Cuál es el estado de *unrn* y *concurso*?

¿Queda *unrn* en estado detached?

PERSISTENCIA ORIENTADA A OBJETOS

¿Y QUÉ MÁS TENEMOS QUE SABER?

Cascade & *Lazy* vs *Early*

PERSISTENCIA ORIENTADA A OBJETOS

Cascade

```
public void nuevoProducto() {
 EntityManager manager = EntityManagerFactory.createEntityManager();
 Categoria elec = new Categoria("electrodoméstico");
 Producto tv = new Producto("TV", elec);
 manager.persist(elec);
 manager.persist(tv);
 manager.close();
}
```

¿Podría solamente persistir el producto y que se persista su categoria mágicamente?

Sí. Debería configurar los colaboradores de *Producto* como *cascade* = true

EARLY VS LAZY

Supongamos el siguiente modelo:

```
public class Universidad {
 private Set<Concurso> concursos = new HashSet<Concurso>();
 private Categoria categoria;
 private String domicilio;
 public void nuevoConcurso(Concurso c) {
  cs.add(c);
 }
 ...
}
public class Concurso {
 private List<Personas> inscriptos = new ArrayList<>();
 ...
 ...
}
```

EARLY VS LAZY

Segun el modelo anterior, si traemos de la BD la instancia *unrn*:

```
...
EntityManager manager = EntityManagerFactory.createEntityManager();
Universidad unrn = manager.find(idUnrn);
manager.close();
...
```

¿Qué trae a memoria junto con *unrn*? ¿su instancia de categoria? ¿sus concursos?

EARLY VS LAZY

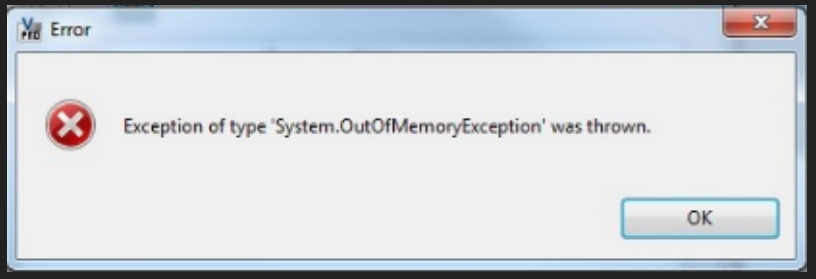
unrn

```
public class Universidad {
 private Set<Concurso> concursos = new HashSet<Concurso>();
 private Categoria categoria;
 private String domicilio;
 public void nuevoConcurso(Concurso c) {
  cs.add(c);
 }
 ...
}
public class Concurso {
 private List<Personas> inscriptos = new ArrayList<>();
 ...
 ...
}
```

Speaker notes

Ir mostrando de a poco el grafo de objetos para ir preguntandole a los alumnos hasta donde se trae...


BOOM


Error
Exception of type 'System.OutOfMemoryException' was thrown.
OK

EARLY VS LAZY

Por colaborador se configura si deseo *early* o *lazy*

Early (temprano): Al recuperar una instancia, recupero su colaborador también.

Lazy (tardío): Al recuperar una instancia, inyecta en su colaborador un Proxy, quién sabe recuperar el colaborador real solo cuando sea necesario.

¿Cuando se vuelve necesario?

LAZY

¿Cuándo se dispara la recuperación de la BD del colaborador?

```
public class Universidad {
  private Set<Concurso> concursos = new HashSet<Concurso>();
  private Categoria categoria;
  private String domicilio;
  public void nuevoConcurso(Concurso c) {
   cs.add(c);
  }
  /*...Demostrar cuando se dispara...*/
 }
public class Concurso {
  private List<Personas> inscriptos = new ArrayList<>();
  ...
  ...
}
```