



Guía de laboratorio
Área de Programación y Algoritmia



UNIVERSIDAD DEL QUINDÍO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Información general	
ACTUALIZADO POR:	Carlos Andrés Flórez Villarraga
DURACIÓN ESTIMADA EN MINUTOS:	120
DOCENTE:	Christian Andrés Candela y Einer Zapata G.
GUÍA NO.	04
Nombre de la guía:	Entidades

Información de la Guía

OBJETIVO

Estudiar el uso de las entidades y su aplicación en el modelamiento de datos.

CONCEPTOS BÁSICOS

Manejo de Eclipse, Java, Bases de Datos, JDBC, XML, Glassfish.

CONTEXTUALIZACIÓN TEÓRICA

Una entidad es un elemento Java de persistencia, que permite modelar o representar un objeto o información del sistema de análisis. Toda entidad está compuesta por uno o más atributos, cada uno de los cuales representa una propiedad o información del objeto que se está modelando. Normalmente, una entidad representa una tabla en una base de datos relacional, y cada instancia de la entidad corresponde a una fila de esa tabla.

Las entidades surgen inicialmente como un tipo de EJB especializado en persistencia. Sin embargo, en la versión JEE 5 pasa de ser un tipo de EJB para convertirse en el corazón del api JPA (Java Persistence Api).

Para que una clase sea considerada una entidad la misma debe cumplir los siguientes requisitos:

- La clase debe ser marcada como Entidad por medio de la anotación `javax.persistence.Entity`.
- La clase debe tener un constructor público o privado sin argumentos. Se puede tener más de un constructor.
- Las entidades pueden extender tanto de otras entidades como de clases que no son entidades, De las misma forma las clases que no son entidades pueden extender de las entidades.
- Los atributos persistentes deben ser declarados como privados, protegidos o privados de paquete (por defecto), y sólo deben ser accedidos directamente por los métodos de la entidad.

Para construir una entidad se pueden usar dos enfoques. El primero de ellos encaminado a persistir los campos (variables de la entidad), en cuyo caso, se usarán anotaciones en las variables para modificar la forma en que estas son almacenadas. El segundo enfoque se basa en la persistencia de las propiedades, en este caso se debe hacer uso la convención de los componentes JavaBeans para los nombres de los métodos de la entidad, y es precisamente en estos métodos en los que se realizarán las anotaciones necesarias para especificar la forma en la que se debe persistir la entidad. Si en algún momento se desea que una de las variables de la clase no sea almacenada se debe usar la anotación `@Transient` o marcada como transitoria. De igual forma es importante tener en cuenta que los elementos bien sean estáticos, finales o transient no son persistidos.

Es importante tener en cuenta que los atributos de una entidad que pueden ser persistidos son los siguientes:

- Otras entidades
- Clases marcadas como superclases (`@MappedSuperClass`)
- Clases embebibles (`@Embeddable`).
- Tipos de datos simples (serializables), los cuales comprenden los tipos de datos primitivos, sus wrappers, `BigInteger`, `BigDecimal`, `String`
- Tipos de datos que representan una fecha (`java.util.Date`, `java.util.Calendar`, `java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`), este tipo de datos deben ser marcados como `@Temporal` y especificar si con fechas (`@Temporal(TemporalType.DATE)`), horas (`@Temporal(TemporalType.TIME)`) o fecha y hora (`@Temporal(TemporalType.TIMESTAMP)`)
- Colecciones de datos del paquete `java.util` (package `java.util`: `ArrayList`, `Vector`, `Stack`, `LinkedList`, `ArrayDeque`, `PriorityQueue`, `HashSet`, `LinkedHashSet`, `TreeSet`)
- Maps del paquete de datos `java.util` (`HashMap`, `Hashtable`, `WeakHashMap`, `IdentityHashMap`, `LinkedHashMap`, `TreeMap` y `Properties`)
- Arrays
- Enumeraciones, se debe tener en cuenta que por defecto las enumeraciones son persistidas en su representación numérica, por lo que si una enumeración se modifica debe hacerse al final para no afectar los datos previamente almacenados. Si se desea almacenar la enumeración como cadena en lugar de su representación numérica se debe indicar con `@Enumerated(EnumType.STRING)` o `@Enumerated(EnumType.ORDINAL)`
- Objetos serializables.

Colecciones de datos simples

En el caso de que una entidad tenga atributos que son tratados como colecciones de tipos de datos simples como puede ser los teléfonos de una persona, se puede hacer uso de la anotación `@ElementCollection`

Tipo de llave

Cada objeto almacenado en una base de datos debe identificarse de forma única, en el caso de las entidades estas son identificadas por su tipo y por una llave. En muchos casos cuando se construye una entidad puede reconocerse claramente el atributo que lo identifica de forma única, en otros casos puede ser que la entidad sea identificada de forma única por un conjunto de atributos, o simplemente puede no existir un atributo que la identifique de forma única, en cuyo caso se debe asignar un atributo que represente la llave y que sea asignado de forma automática.

Básico: Cuando un campo o atributo simple (tipo primitivo, wrapper, `BigInteger`, `BigDecimal`, `String`, `Date`, `Time`, `Timestamp`, enum, otra entidad) identifica de forma única la entidad, en este caso dicho campo es marcado como la llave de la entidad así:

```
@Id
private String isbn;
```

Compuestas: Las llaves compuestas son aquellas conformadas por más de un campo. Para crear una llave compuesta se debe hacer una clase que agrupe los campos que compondrán la llave e identificar dicha clase en la entidad de forma específica así:

```
public class Llave implements Serializable {  
    private int atributo1;  
    private long atributo2;  
    ...  
}  
  
@Entity  
@IdClass(Llave.class)  
public class Entidad implements Serializable {  
    @Id  
    private int atributo1;  
    @Id  
    private long atributo2;  
    ...  
}
```

Embebibles: En el caso de las llaves embebibles, son llaves que son compuestas por más de un campo, pero que son declaradas dentro de la entidad como un único atributo representado a través de un tipo de dato que a su vez ha sido marcado con la anotación `@Embeddable` así:

```
@Embeddable  
public class Llave implements Serializable {  
    private int campo1;  
    private long campo2;  
    ...  
}  
  
@Entity  
public class Entidad implements Serializable {  
    @EmbeddedId  
    private Llave atributo;  
    ...  
}
```

Generadas: Las llaves generadas como se ha dicho, son aquellas que se crean para poder identificar de forma única a una entidad que por sí misma no tiene un campo que la identifique de forma única. Estas llaves son generadas de forma automática por el sistema.

```
@Entity  
public class Entidad implements Serializable {  
    @Id  
    @GeneratedValue  
    private Long id;  
}
```

Para la generación de valores de forma automática para la llave puede realizarse mediante diferentes estrategias (Auto, Identity, Sequence, Table).

- **Auto** (`@GeneratedValue(strategy=GenerationType.AUTO)`): En este caso se tiene un generador de números para cada base de datos, los cuales son usados para crear valores a ser asignados como llaves primarias. Esta es la estrategia usada por defecto.
- **Identity** (`@GeneratedValue(strategy=GenerationType.IDENTITY)`): Similar al auto, sin embargo no existe un generador por base de datos, en su lugar existe un generador de números para cada tipo.
- **Sequence** (`@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="seq")`): Se usa cuando el motor de base de datos soporta el uso de secuencias para la generación de números. En el caso de la secuencia se debe usar una anotación de clase que declare la secuencia y sus valores iniciales `@SequenceGenerator(name="seq", initialValue=1, allocationSize=100)`
- **Table** (`@GeneratedValue(strategy=GenerationType.TABLE, generator="tab")`): Similar a la secuencia, pero en este caso es el proveedor de JPA es el encargado de crear una tabla para simular una secuencia y así poder asignar los números a ser usados como llaves primarias. La tabla también debe declararse previamente con una anotación de clase `@TableGenerator(name="tab", initialValue=0, allocationSize=50)`

PRECAUCIONES Y RECOMENDACIONES

Recuerde verificar que el servidor de aplicaciones soporte el motor de base de datos que usará, de igual forma debe verificar que eclipse este haciendo uso del JDK y no del JRE y recuerde adicionar al workspace el servidor de aplicaciones Glassfish antes de crear cualquier proyecto. También puede ser importante verificar que los puertos usados por Glassfish no estén ocupados (Para ello puede hacer uso del comando **netstat -npl** o **netstat -a**).

ARTEFACTOS

Se requiere tener instalado el JDK y un IDE para el desarrollo de aplicaciones (Eclipse JEE en su última versión), un servidor de aplicaciones que cumpla con las especificaciones de JEE, para esta práctica Glassfish y el motor de base de datos Mysql.

EVALUACIÓN O RESULTADO

Se espera que el alumno logre modelar entidades lógicas de una aplicación y por medio de ellas se cree tablas en la base de datos que las representen de forma adecuada.

Procedimiento

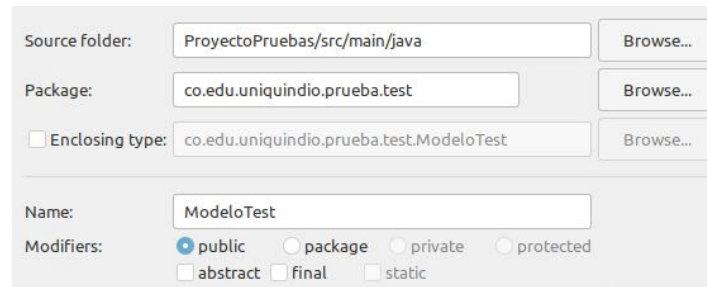
1. Para empezar, será necesario crear una base de datos en MySQL. Si ya ha creado previamente una obvie este paso.
2. Si aún no lo ha hecho, adicione al workspace de Eclipse el servidor de aplicaciones GlassFish.
3. En Eclipse cree un Proyecto de tipo Maven Project (padre) que contenga un módulo de persistencia y otro de pruebas, todo con las mismas características de la guía anterior. Puede hacer uso de los proyectos creados en la guía de Maven.

- Configure la conexión a su base de datos (Pool y Datasource). Si ya posee una, puede omitir este paso.
- Cree una entidad en el proyecto persistencia, no olvide generar un empaquetado formal. Para este caso cree la entidad `Persona` con los atributos cédula, nombre y apellido.

Para ello, de clic derecho sobre el proyecto y en el menú new seleccione `JPA Entity`. Deberá proporcionar el nombre del paquete en el que se almacenará la entidad así como el nombre dicha entidad. En la siguiente pantalla adicione los atributos de la entidad, seleccione la cédula como llave.

- Al manejar entidades es importante poder compararlas y decir cuándo una entidad es igual a otra, por lo que es aconsejable, aunque no estrictamente necesario sobrescribir el método `equals()` de las entidades creadas. Para hacer esto, teniendo el archivo de su entidad abierta y el nombre de la clase seleccionado acceda al menú `source`, opción `Generate hashCode() and equals()`. En la ventana que aparece verá todos los atributos seleccionados de su entidad, se sugiere dejar seleccionado únicamente el atributo correspondiente a la llave primaria de la entidad (es decir, la cédula).
- Para verificar la creación de las tablas se adicionará al Proyecto de pruebas una clase que al ser ejecutada forzará la creación de las tablas. Para ello, de clic derecho sobre el proyecto de pruebas y seleccione el menú `new - Class`

En la pantalla siguiente proporcione un nombre de paquete y posteriormente uno de clase así:



- Adicione a su clase los siguientes imports

```
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.jboss.arquillian.container.test.api.Deployment;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.shrinkwrap.api.Archive;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.asset.EmptyAsset;
import org.jboss.shrinkwrap.api.spec.WebArchive;
import org.junit.Test;
import org.junit.runner.RunWith;
```

- Inque a JUnit que la clase se ejecutará por medio de Arquillian, así:

```
@RunWith(Arquillian.class)
public class ModeloTest {
```

```
}
```

10. Cree un atributo de clase de tipo `EntityManager`, el cual nos permitirá la gestión de las entidades.

```
@RunWith(Arquillian.class)
public class ModeloTest {

    @PersistenceContext
    private EntityManager entityManager;

}
```

11. Programe un método estático para la creación del archivo de empaquetado de java.

```
@RunWith(Arquillian.class)
public class ModeloTest {

    @PersistenceContext
    private EntityManager entityManager;

    @Deployment
    public static Archive<?> createTestArchive() {
        return ShrinkWrap.create(WebArchive.class,
            "prueba.war").addPackage(ENTIDAD.class.getPackage())
            .addAsResource("persistenceForTest.xml",
"META-INF/persistence.xml")
            .addAsWebInfResource(EmptyAsset.INSTANCE, "beans.xml");
    }

}
```

NOTA: Como puede observar el método anterior crea un archivo de tipo war que contiene el paquete de las entidades y el archivo de persistencia para las pruebas. ENTIDAD debe ser reemplazado por el nombre de una de sus entidades existentes en el Proyecto de Persistencia, por lo tanto, si no lo ha hecho, debe agregar como dependencia del proyecto de pruebas el de persistencia.

12. Cree un método para test que se llame `generarTest()`, puede estar vacío, ya que solo será usado para obligar a la generación de las tablas.

```
@RunWith(Arquillian.class)
public class ModeloTest {

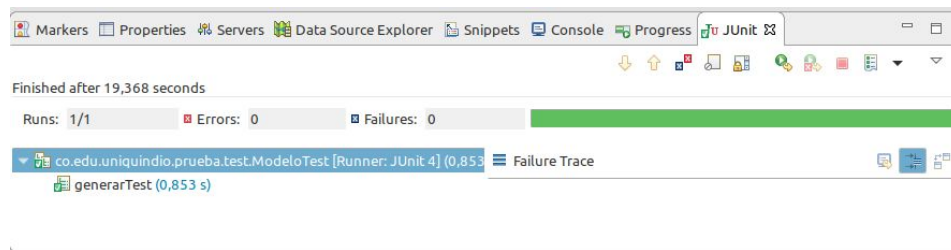
    @PersistenceContext
    private EntityManager entityManager;

    @Deployment
    public static Archive<?> createTestArchive() {
        return ShrinkWrap.create(WebArchive.class,
            "prueba.war").addPackage(ENTIDAD.class.getPackage())
            .addAsResource("persistenceForTest.xml",
"META-INF/persistence.xml")
    }

}
```

```
        .addAsWebInfResource (EmptyAsset.INSTANCE, "beans.xml" );  
  
    }  
  
    @Test  
    public void generarTest() {  
  
    }  
  
}
```

13. De clic derecho sobre su clase `ModeloTest` y ejecútela como prueba unitaria (JUnit).



Verifique que la prueba se ejecutó correctamente, debe estar en color verde.

14. Para ver las tablas generadas, abra la consola de MySQL y tras autenticarse ingrese los siguientes comandos.

```
show tables from BASE_DATOS;  
use BASE_DATOS;  
describe TABLA;
```

NOTA: `BASE_DATOS` debe ser reemplazado por el nombre de su base de datos y `TABLA` debe ser reemplazada por el nombre de la tabla a la cual desea observar sus campos.

15. Cree una enumeración que le permita representar el género de una persona (masculino o femenino).
16. Adicione a su entidad un atributo que represente el género haciendo uso de la enumeración creada previamente.
17. Elimine la tabla previamente creada y luego ejecute nuevamente la generación de tablas a través de la clase `ModeloTest` y observe los resultados.

Puede usar el siguiente comando para borrar una tabla:

```
drop table TABLA;
```

18. Adicione a su entidad un atributo que represente los números de teléfono de la persona, tenga en cuenta que una persona puede tener más de un número de teléfono. Para ello use inicialmente un `List` y la anotación `@ElementCollection`.



Guía de laboratorio
Área de Programación y Algoritmia



19. Ejecute nuevamente la generación de tablas y observe los resultados.
20. Ahora bien, si se desea asociar una clave o texto que permita identificar o más bien diferenciar el número de teléfono de la casa del celular y del trabajo, se podría usar un `Map` en lugar de un `List`. Para observar la diferencia cambie el tipo de dato de `List` a `Map`.
21. Ejecute nuevamente la generación de tablas (recuerde borrar las previamente generadas) y observe los resultados.
22. Cree una segunda entidad que represente un programa, con atributos nombre, descripción y versión. Tenga en cuenta que al no tener un atributo que lo pueda identificar de forma única deberá adicionar un atributo que lo identifique y sea generado de forma automática por el sistema.
23. Ejecute nuevamente para generar las tablas y observe los resultados.
24. Cree una tercera entidad que represente un punto en el planeta, el cual tiene como atributos la longitud, la latitud y un nombre. Tenga en cuenta que para esta entidad deberá usar una llave compuesta.
25. Ejecute nuevamente la generación de tablas y observe los resultados.

Para la próxima clase

Leer sobre los diferentes tipos de restricciones que se le pueden asignar a los campos de una tabla de una base de datos