

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

Información general	
ACTUALIZADO POR:	Einer Zapata G.
DURACIÓN ESTIMADA EN MINUTOS:	120
DOCENTE:	Christian Andrés Candela
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 la 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 super clases (`@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;  
}
```

La generación automática de las de valores para la llave de las entidades, se puede realizar por medio de 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.

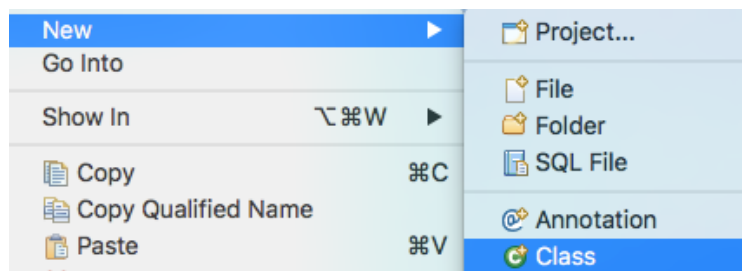
## EVALUACIÓN O RESULTADO

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.

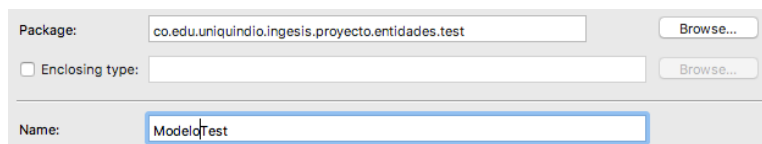
### Procedimiento

1. Para empezar, será necesario crear una base de datos en mysql. Si ya ha creado una obvie este paso.
2. Adicione al workspace el servidor de aplicaciones GlassFish.

3. En eclipse cree un Proyecto de tipo Maven Project (padre) que contenga un modulo de persistencia y otro de pruebas, todo con las mismas características del de la guía anterior. Puede hacer uso de los creados en la guía de Maven.
4. Configure la conexión a su base de datos. Si ya posee una, puede omitir este paso.
5. Cambie la perspectiva de trabajo a la de JPA.
6. Cree una entidad en el proyecto persistencia, no olvide generar un empaquetado formal. Para este caso cree la entidad niño con los atributos identificación, fecha de nacimiento, nombre, apellido. Para ello, de clic derecho sobre el proyecto y en el menú new seleccione nueva entidad. Deberá proporcionar el nombre del paquete en el que se almacenará la entidad así como el nombre de la clase a ser usada como entidad. En la siguiente pantalla adicione los atributos de la entidad, seleccione la identificación como llave. En la parte inferior podrá determinar la forma en la que desee se estructure la entidad (Fields o Properties).
7. Al manejar entidades es importante poder compararlas y decir cuando 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 solo el atributo correspondiente a la llave primaria de la entidad.
8. 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



9. 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;
```

10. Inque a JUnit que la clase se ejecutara por medio de arquilliam

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

}
```

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

```
@RunWith(Arquillian.class)
public class TestModelo {
    @PersistenceContext
    private EntityManager entityManager;

}
```

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

```
@RunWith(Arquillian.class)
public class TestModelo {
    @PersistenceContext
    private EntityManager entityManager;

    @Deployment
    public static Archive<?> createTestArchive() {
        return ShrinkWrap.create(WebArchive.class,
            "test.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 remplazado por el nombre de una de sus entidades.

13. Cree un método test, el mismo puede estar vacío, ya que solo será usado para obligar a la generación de las tablas.

```
@RunWith(Arquillian.class)
public class TestModelo {
    @PersistenceContext
    private EntityManager entityManager;

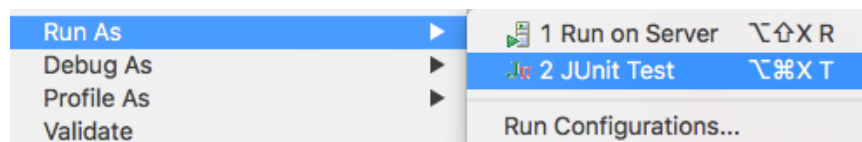
    @Deployment
```

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

@Test
public void generarTest() {
}
}
```

14. Agregue como dependencia del proyecto pruebas el de persistencia.

15. De clic derecho sobre su clase Test y ejecútela como prueba unitaria



16. Para ver las tablas generadas abra la consola de mysql en el menú ejecutar de Windows, y tras autenticarse ingrese el siguiente comando.

```
show tables from BASE_DATOS;
describe BASE_DATOS;
```

**NOTA:** BASE\_DATOS debe ser remplazado por el nombre de su base de datos.

17. Cree una enumeración que le permita representar el género de una niño (masculino o femenino).

18. Adicione a su entidad un atributo que represente el género haciendo uso de la enumeración creada previamente.

19. Ejecute nuevamente la generación de tablas y observe los resultados.

20. 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.

21. Ejecute nuevamente la generación de tablas y observe los resultados.

22. Ahora bien, si se deseara 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 agregue un nuevo atributo de tipo de dato Map.

23. Ejecute nuevamente la generación de tablas y observe los resultados.

24. 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.
25. Ejecute nuevamente para generar las tablas y observe los resultados.
26. 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.
27. Ejecute nuevamente la generación de tablas y observe los resultados.



## Proyecto Final

Basado en el siguiente problema elabore un diagrama de clases que lo represente.

El Herbario de la Universidad del Quindío (HerbarioUQ), desea manejar una plataforma que les permita manejar la información de las plantas que han recolectado y recolectarán.

**Nota:** En adelante la acción gestionar hará referencia a crear, modificar, buscar (ver información), listar y si es posible eliminar o invalidar información.

HerbarioUQ desea contar con una aplicación que maneje tres tipos de roles, administrador, empleado y recolector.

### Administrador:

- Gestionar géneros y familias de plantas.
- Gestionar empleados.
- Gestionar recolectores.
- Registrar especies vegetales (plantas). Al registrar la información de las plantas se debe tomar en cuenta el nombre de la familia, del género y la especie, además de poder cargar una imagen representativa de la especie.
- Aceptar o rechazar las especies vegetales registradas.
- Ver la información detallada de una especie vegetal.
- Listar todas las especies vegetales.
- Listar las especies vegetales aceptadas o rechazadas.
- Listar las especies vegetales por familia o genero.
- Recuperar contraseñas usando correo electrónico.

### Empleado:

- Gestionar recolectores.
- Registrar especies vegetales (plantas). Al registrar la información de las plantas se debe tomar en cuenta el nombre de la familia, del género y la especie, además de poder cargar una imagen representativa de la especie.
- Ver la información detallada de una especie vegetal.
- Listar todas las especies vegetales.
- Listar las especies vegetales aceptadas o rechazadas.
- Listar las especies vegetales por familia o genero.
- Recuperar contraseñas usando correo electrónico.

### Recolector:

- Registrarse y actualizar su información.
- Registrar especies vegetales (plantas). Al registrar la información de las plantas se debe tomar en cuenta el nombre de la familia, del género y la especie, además de poder cargar una imagen representativa de la especie.
- Ver la información detallada de una especie vegetal.
- Ver la lista de las especies vegetales aceptadas.
- Ver la lista de especies vegetales que ha enviado. Tenga en cuenta el respectivo estado del envío (aceptada o rechazada ¿Por qué?).
- Listar las especies vegetales (aceptadas) por familia o genero.
- Recuperar contraseñas usando correo electrónico.



*Guía de laboratorio*  
*Área de Programación y Algoritmia*



Para tener en cuenta:

- Piense en otras necesidades asociadas (no trivial) a este sistema para que las incluya dentro de su modelo. Esta funcionalidad debe estar presente durante las tres entregas del semestre.
- Selección de un groupid único.