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

Información general	
Fecha :	2015-02-26
Duración estimada en minutos:	240
Docente:	Christian Andrés Candela
Guía no.	6
Nombre de la guía:	DAOs

Información de la Guía
Objetivos: Aprender a usar el patrón de diseño DAO en el desarrollo de aplicaciones empresariales.
Conceptos Básicos: Manejo de Eclipse, Java, Bases de Datos, JDBC, DataSource, Entidades y Glassfish.
Contextualización Teórica: Los DAOs (Data Access Object) son un patrón de diseño usado para proporcionar a las aplicaciones un medio común de acceso a los datos almacenados (bien sea en una base de datos, en un archivo o en algún otro medio). Uno de los principales objetivos de los DAOs es ocultar a la aplicación la forma especifica en la que se esta almacenando la información. De esta forma será posible (de ser necesario) cambiar la forma en que se almacena la información sin que esto implique la modificación de otras partes de la aplicación. Una de las principales desventajas a la hora de usar los DAOs se encuentra en la creación de una capa más dentro de la aplicación que invariablemente implica más código ejecutándose y más trabajo a la hora del desarrollo. Por lo general, los DAOs ofrecen funciones que permiten la inserción, borrado, actualización y consulta de datos.
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 asiendo 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 puestos 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 Luna), 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 pueda comprender los DAOs y su uso; y sea capaz de emplearlos correctamente en el desarrollo de aplicaciones.

Procedimiento

1. Para empezar será necesario crear una base de datos en mysql.
2. En eclipse cree un Proyecto de tipo Maven Project. No olvide antes de crear el proyecto adicionar al workspace el servidor de aplicaciones GlassFish.
3. Configure adecuadamente la persistencia del proyecto para que haga uso de la base de datos creada.
4. Configure el archivo persistence.xml para que se creen automáticamente las Tablas.
5. Cree al menos una entidad, o use entidades creadas previamente. Ejemplo.

```
/**
 * Clase que representa la entidad Persona
 *
 * @author cacandela
 *
 */
@Entity
public class Persona implements Serializable {

	/**
	 * Atributo de serializacion
	 */
	private static final long serialVersionUID = 1L;
	/**
	 * Atributo que representa el/la cedula
	 */
	@Id
	private String cedula;
	/**
	 * Atributo que representa el/la nombre
	 */
	private String nombre;
	/**
	 * Atributo que representa el/la apellido
	 */
	private String apellido;

	/**
	 * Permite obtener le valor del atributo cedula
	 *
	 * @return El valor del atributo cedula
	 */
	public String getCedula() {
		return cedula;
	}
```

```
	/**
	 * Permite cambiar el valor del atributo cedula
	 *
	 * @param cedula
	 *            El nuevo valor a ser asignado al atributo cedula
	 */
	public void setCedula(String cedula) {
		this.cedula = cedula;
	}

	/**
	 * Permite obtener le valor del atributo nombre
	 *
	 * @return El valor del atributo nombre
	 */
	public String getNombre() {
		return nombre;
	}

	/**
	 * Permite cambiar el valor del atributo nombre
	 *
	 * @param nombre
	 *            El nuevo valor a ser asignado al atributo nombre
	 */
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	/**
	 * Permite obtener le valor del atributo apellido
	 *
	 * @return El valor del atributo apellido
	 */
	public String getApellido() {
		return apellido;
	}

	/**
	 * Permite cambiar el valor del atributo apellido
	 *
	 * @param apellido
	 *            El nuevo valor a ser asignado al atributo apellido
	 */
	public void setApellido(String apellido) {
		this.apellido = apellido;
	}

}
```

6. Así como ha creado paquetes para almacenar sus entidades, cree un paquete para el almacenamiento de sus daos. Ejemplo co.edu.uniquindio.dao.

7. Usaremos Generics para la construcción de los DAOs. Cree una clase genérica y abstracta con 1 tributo genérico, que defina la entidad a ser manipulada así:

```
/**
 * Clase generica que brinda un modelo para la construccion de los DAOs que
 * permitiran la manipulacion de las entidades
 *
 *
 * @param <Entidad>
 *            Entidad que sera manipulada
 *
 * @author Christian A. Candela
 * @author Grupo de Investigacion en Redes Informacion y Distribucion - GRID
 * @version 2.0
 * @since 2013-04-12
 */
public abstract class Dao<Entidad extends Serializable> {
...
}
```

8. Adicione el atributo de tipo EntityManager. Este atributo nos permitirá la interacción con la base de datos. El atributo deberá crearse así:

```
/**
 * Entity Manager que permitira la ejecucion de las sentencias JPQL
 */
@Inject
protected EntityManager entityManager;
```

9. Cree un atributo que represente la clase de la entidad que será manipulada por el DAO.

```
/**
 * Representa la clase a la que pertenece la entidad que esta manipulando el DAO
 */
private Class<Entidad> entityClass;
```

10. Cree un método constructor para el DAO el cual deberá recibir por parámetro la clase de la entidad que manipulará.

```
/**
 * Costructor del DAO que permite inicializar y determinar la clase a la que
 * pertenece la Entidad que se manipulara
 *
 * @param entityClass
 */
public Dao(Class<Entidad> entityClass) {
        this.entityClass = entityClass;
}
```

11. Cree un método con el nombre insert, este método nos permitirá insertar entidades en la base de datos.

```
/**
 * Metodo que permite la insercion de una entidad en la base de datos
 *
 * @param entidad
 *          Entidad a ser almacenada
 * @return Retorna la entidad almacenada
 */
public Entidad insert(Entidad entidad){
        entityManager.persist(entidad);
        return entidad;
}
```

El método persist almacena la entidad dada en la base de datos. Como puede ver el método insert es el mismo método persist que encuentra en los EJB que creo previamente, en este caso ha sido modificado para que funcione de forma genérica para cualquier entidad y no para una específica.

12. Cree un método con el nombre update, este método nos permitirá actualizar los datos de una entidad dada.

```
/**
 * Metodo que permite la actualizacion de una entidad
 *
 * @param entidad
 *          Entidad a ser actualizada
 * @return Retorna la entidad actualizada
 */
public Entidad update(Entidad entidad){
        return entityManager.merge(entidad);
}
```

El método merge actualiza el estado de la entidad en la base de datos. Al igual que en el caso anterior este método es el método merge de los EJB modificado para generalizarlo.

13. Cree un método delete, el cual permita remover una entidad de la base de datos.

```
/**
 * Metodo que permite remover una entidad de la base de datos
 *
 * @param entidad
 *          Entidad a ser removida
 */
public void remove(Entidad entidad) {
    entityManager.remove(entityManager.merge(entidad));
}
```

El método remove permite remover una entidad de la base de datos, pero dicha entidad debe estar asociada a la base de datos, por lo que se debe invocar inicialmente el método merge. En este caso se ha realizado una pequeña modificación al método remove, esto debido a que no se puede saber en este punto cual es el campo que representa la llave primaria de la entidad genérica.

14. Cree un método con el nombre getEntityClass, este método retornará la clase de la entidad con la que se esta trabajando. Este método es necesario debido al uso de los generics y a que en entity manager requiere el uso de la clase de la entidad para la realización de algunos métodos.

```
/**
 * Metodo que permite obtener la clase a la que pertenece la entidad
 * panipulada
 *
 * @return La clase a la que pertenece la entidad manipulada
 */
public Class<Entidad> getEntityClass() {
        return entityClass;
}
```

15. Cree el método findByKey, el cual permite obtener una entidad de la base de datos por medio de su llave primaria.

```
/**
 * Permite buscar una entidad por medio de se llave
 *
 * @param llave
 *          Llave de la entidad que se desea buscar
 * @return Retorna la entidad que corresponda a la llave dada, en caso de no
 *       encontrar un resultado se retorna null
 */
public Entidad findByKey(Serializable llave){
        return entityManager.find(getEntityClass(), llave);
}
```

16. Cree un método que permita ejecutar unas sentencia de búsqueda dentro de un rango dado.

```
/**
 * Metodo que permite realizar una consulta JPQL sobre la base de datos
 *
 * @param jpqlStmt
 *          Sentencia JPQL a ser ejecutada
 * @param firstResult
 *          Posicion de inicio del primer resultado de la consulta
 * @param maxResults
 *          Maximo numero de resultados a ser obtenidos a partir de la
 *          primera posicion
 * @return Un listado con los registros obtenidos de la consulta ejecutada
 */
public List<?> queryByRange(String jpqlStmt, int firstResult,
                        int maxResults) {
   Query query = entityManager.createQuery(jpqlStmt);
   if (firstResult > 0) {
      query = query.setFirstResult(firstResult);
   }
   if (maxResults > 0) {
      query = query.setMaxResults(maxResults);
```

```
    }

    return query.getResultList();
}
```

Como puede ver este es el mismo método que se usa en los EJB, solo se modificó el tipo de retorno para hacerlo más preciso.

17. Cree un método getAll que permita obtener un listado con todos los registros de una determinada Entidad y que permita establecer un rango.

```
/**
 * Permite obtener todos los registros almancenados de una determinada
 * entidad partiendo de la posicion indicada hasta el maximo numero de
 * registros dados.
 *
 * @param firstResult
 *          Posicion de inicio del primer resultado de la consulta
 * @param maxResults
 *          Maximo numero de resultados a ser obtenidos a partir de la
 *          primera posicion
 * @return El listado de entidades encontrado
 */
public List<Entidad> getAll(int firstResult, int maxResults) {
        TypedQuery<Entidad> query = entityManager.createQuery(
                        "select entidad from " + getEntityClass().getSimpleName()
                        + " entidad ", getEntityClass());
        if (firstResult > 0) {
                query = query.setFirstResult(firstResult);
        }
        if (maxResults > 0) {
                query = query.setMaxResults(maxResults);
        }

        return query.getResultList();
}
```

El método createQuery del entity manager nos permite ejecutar consultas sobre las entidades existentes. Por ejemplo “select persona from Persona persona” retornaría todas las entidades de tipo persona almacenadas en la base de datos. El segundo parámetro que recibe el método createQuery corresponde a la clase de la entidad que se esta consultando, esto permite definir el tipo de lista que se retornará. Por último se invoca al método getResultList que es quien finalmente retorna el listado de entidades encontradas.

18. Cree un método getAll que permita obtener un listado con todos los registros de una determinada Entidad y que retorne todos los elementos almacenados.

```
/**
 * Permite obtener todos los registros almancenados de una determinada
 * entidad.
 *
 * @return El listado de entidades encontrado
```

```
 */
public List<Entidad> getAll() {
        return getAll(-1, -1);
}
```

19. Ahora cree un método findRange, el cual permitirá obtener los registros almacenados de una entidad dentro de un determinado rango.

```
/**
 * Permite obtener todos los registros almancenados de una determinada
 * entidad que se encuentren dentro del rango especificado por el parametro
 * rango.
 *
 * @param rango
 *            Arreglo que contiene el rango de resultados que se desea
 *            obtener, en la posicion 0 se envía el indice del primer
 *            resultado que se desea obtener. En el indice 1 se envía el
 *            indice del último registro que se desea obtener
 * @return El listado de entidades encontrado
 */
public List<Entidad> findRange(int[] range) {
      return getAll(range[0], range[1] - range[0]);
}
```

20. Cree un método count, el cual le permita establecer el número total de registros de una determinada entidad.

```
/**
 * Permite obtener el número de registros de una determinada entidad
 *
 * @return El número de registros de una determinada entidad
 */
public int count() {
      TypedQuery<Long> query = entityManager.createQuery(
                  "select COUNT(entidad) from "
+ getEntityClass().getSimpleName() + " entidad ",
                  Long.class);
      return query.getSingleResult().intValue();

}
```

21. Cree un método queryByRange que le permita crear una sentencia JPQL y obtener los registros que se encuentren dentro del rango especificado.

```
/**
 * Metodo que permite realizar una consulta JPQL sobre la base de datos,
 * obtener todos los registros que se encuentren dentro del rango
 * especificado por el parametro rango.
 *
 * @param jpqlStmt
 *            Sentencia JPQL a ser ejecutada
 * @param rango
 *            Arreglo que contiene el rango de resultados que se desea
 *            obtener, en la posicion 0 se envía el indice del primer
 *            resultado que se desea obtener. En el indice 1 se envía el
 *            indice del último registro que se desea obtener
 * @return Un listado con los registros obtenidos de la consulta ejecutada y
 *         dentro del rango establecido
 */
public List<?> queryByRange(String jpqlStmt, int[] rango) {
      return queryByRange(jpqlStmt, rango[0], rango[1] - rango[0]);
}
```

22. Ahora por cada entidad que tenga su proyecto cree una clase que extienda de la clase Dao. Ejemplo:

```
/**
 * Clase Dao (Data Access Object) de la entidad Persona
 *
 * @author cacandela
 */
public class PersonaDao extends Dao<Persona> {

      /**
       * Metodo que permite inicializar los elementos de la clase PersonaDao
       */
      public PersonaDao() {
            super(Persona.class);
      }

}
```

Note que en el ejemplo al extender de Dao se especifica cual de las entidades será la manejada por este dao. Adicionalmente se crea un constructor que remplaza el extendido enviando por parámetro la clase de la entidad.

23. 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 sobreescribir 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 hascode and equals**. En la ventana que aparece verá todos los atributos de su entidad seleccionados, se sugiere dejar seleccionado solo el atributo correspondiente a la llave primaria de la entidad.

24. Cree un DAO para cada una de sus entidades.