

## PROGRAMACION N-CAPAS LABORATORIO #5



**CRUD JPA @Transactional, persist(), merge(),flush()  
Capa de Servicio @Service  
Relaciones entre entidades @OneToMany, @ManyToOne**

### **Catedrático:**

Lic. Juan Lozano

### **Instructores:**

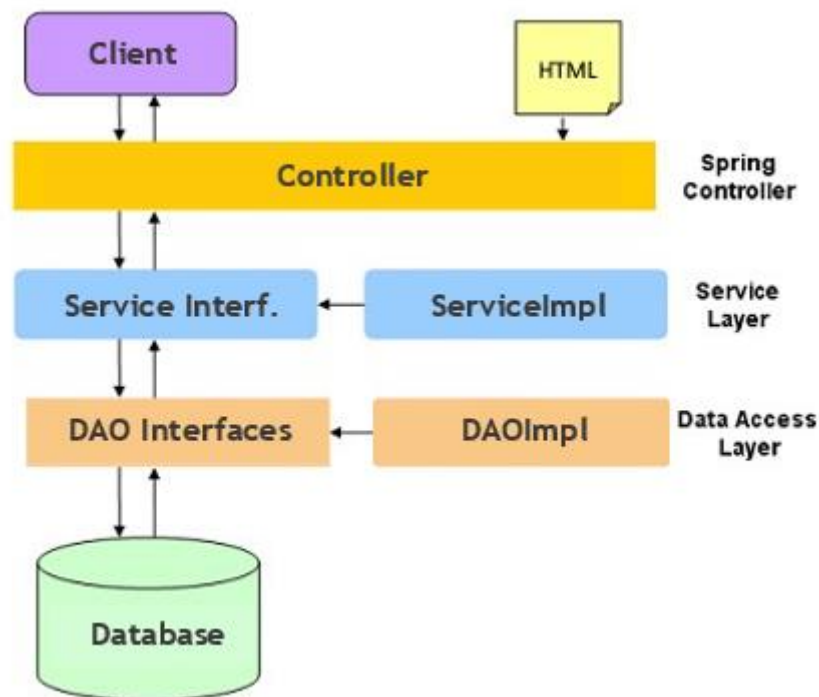
Karla Beatriz Morales Alfaro [00022516@uca.edu.sv](mailto:00022516@uca.edu.sv)  
Sara Noemy Romero Menjivar [00030716@uca.edu.sv](mailto:00030716@uca.edu.sv)  
Salvador Edgardo Campos Gómez [00117716@uca.edu.sv](mailto:00117716@uca.edu.sv)

## CRUD JPA



### Capa de Servicio @Service

La capa de servicios es una división entre la capa de acceso a datos y la propia base de datos, funciona como un intermediario para que de esta manera la lógica de negocio no esté tan arraigada a la capa DAO y así dar consistencia a la manera en la cual se programan los aplicativos.



**@Service:** Esta anotación le dicta a Spring que dicha clase es de Servicio, con lo cual se podrá manejar los métodos dentro de transacciones.

### Ejemplo de Laboratorio:

#### Observaciones:

- Se trabajará con la base de datos del laboratorio 5 **BDEscuela**, usando la base de datos se harán **insert**, **delete** y **update** de los datos de la tabla **estudiante**.

figura 1. Tabla estudiante

Query Editor   Query History

```
1 SELECT * FROM estudiante;
```

Data Output

	id_estudiante integer	nombre character varying (30)	apellido character varying (30)	edad integer	estado boolean
1	1	Karla	Morales	22	true
2	2	Ayleen	Alfaro	20	false
3	3	Wilfredo	Maqueen	20	true
4	4	Monica	Herrera	22	true
5	5	Alan	Brito	29	false

- Se trabajará sobre el proyecto del laboratorio #5.

figura 2. Paquetes del proyecto

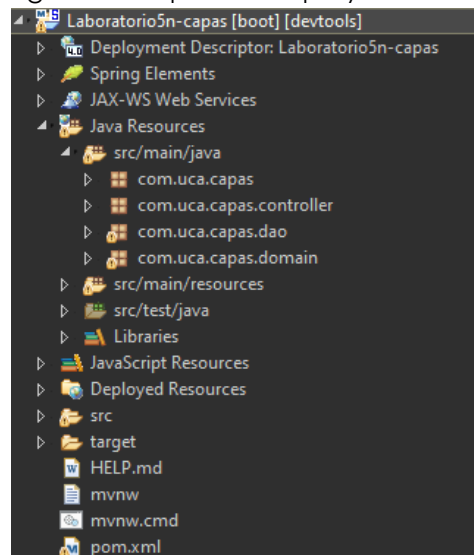


figura 3. Estudiante.java

```
Estudiante.java x
9  @Entity
10 @Table(schema="public",name="estudiante")
11 public class Estudiante {
12
13     @Id
14     @Column(name="id_estudiante")
15     private Integer códigoEstudiante;
16
17     @Column(name="nombre")
18     private String nombre;
19
20     @Column(name="apellido")
21     private String apellido;
22
23     @Column(name="edad")
24     private Integer edad;
25
26     @Column(name="estado")
27     private Boolean estado;
28 }
```

- Agregar validaciones en Estudiante.java

figura 3.1 . Validaciones

```
@Id
@GeneratedValue(generator = "estudiante_id_estudiante_seq", strategy = GenerationType.AUTO)
@SequenceGenerator(name = "estudiante_id_estudiante_seq", sequenceName = "public.estudiante_id_estudiante_seq")
@Column(name="id_estudiante")
private Integer codigoEstudiante;

@Size(message = "El campo no debe contener mas de 30 caracteres", max = 30)
@NotEmpty(message = "Este campo no puede estar vacio")
@Column(name="nombre")
private String nombre;

@Size(message = "El campo no debe contener mas de 30 caracteres", max = 30)
@NotEmpty(message = "Este campo no puede estar vacio")
@Column(name="apellido")
private String apellido;

@NotNull(message = "El campo no puede estar vacio")
@Min(value = 18, message = "No puede ser menor a 18 años")
@Column(name="edad")
private Integer edad;

@Column(name="estado")
private Boolean estado;
```

- Para utilizar la anotación @Transactional debemos de configurar la funcionalidad en la clase de configuración de JPA:
  - Agregar en **JPAConfiguration**:

figura 4. @EnableTransactionManagement

```
@Configuration
@EnableTransactionManagement
public class JPAConfiguration {
```

**@EnableTransactionManagement** habilita el soporte de transacciones por Spring

- Agregar en **JPAConfiguration**:

figura 5. JpaTransactionManager

```
@Bean
JpaTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {
    JpaTransactionManager transactionManager = new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(entityManagerFactory);
    return transactionManager;
}
```

figura 6. Configuración de la secuencia.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name="id_estudiante")
private Integer codigoEstudiante;
```

## -----CAPA DAO-----

EntityManager proporciona una API para todas las operaciones de persistencia requeridas. Estas incluyen las siguientes operaciones CRUD:

- **persist()** -----> (INSERT)
- **merge()** -----> (UPDATE)
- **remove()** -----> (DELETE)
- **find()** -----> (SELECT)
- **flush()** ----> Ejecuta, sincroniza la consulta con la base de datos.

- Agregar los métodos de **save** y **delete** en EstudianteDAO:

figura 7. Metodos en **EstudianteDAO.java**

```
public void save(Estudiante estudiante) throws DataAccessException;

public void delete(Integer codigoEstudiante) throws DataAccessException;
```

- Implementar métodos en EstudianteDAOImpl

figura 8. Metodo save en **EstudianteDAOImpl.java**

```
@Override
@Transactional
public void save(Estudiante estudiante) throws DataAccessException {
    // TODO Auto-generated method stub
    try {
        if(estudiante.getCodigoEstudiante() == null) {
            entityManager.persist(estudiante);
        }
        else {
            entityManager.merge(estudiante);
            entityManager.flush();
        }
    } catch (Throwable e) {
        e.printStackTrace();
    }
}
```

figura 9. Método delete en **EstudianteDAOImpl.java**

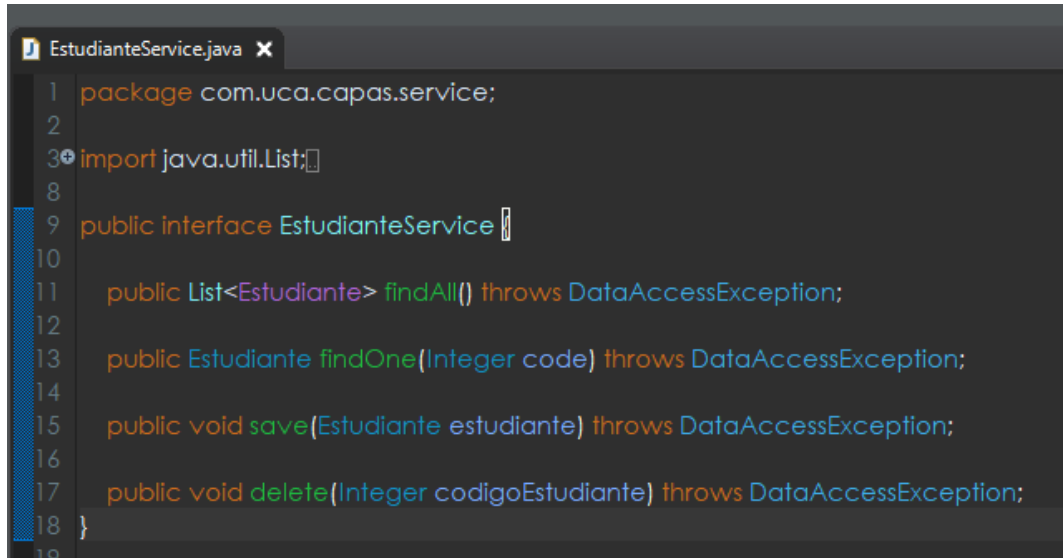
```
@Override
@Transactional
public void delete(Integer codigoEstudiante) throws DataAccessException {
    // TODO Auto-generated method stub
    Estudiante estudiante = entityManager.find(Estudiante.class, codigoEstudiante);
    entityManager.remove(estudiante);
}
```

## -----CAPA DE SERVICIO-----

### @Service

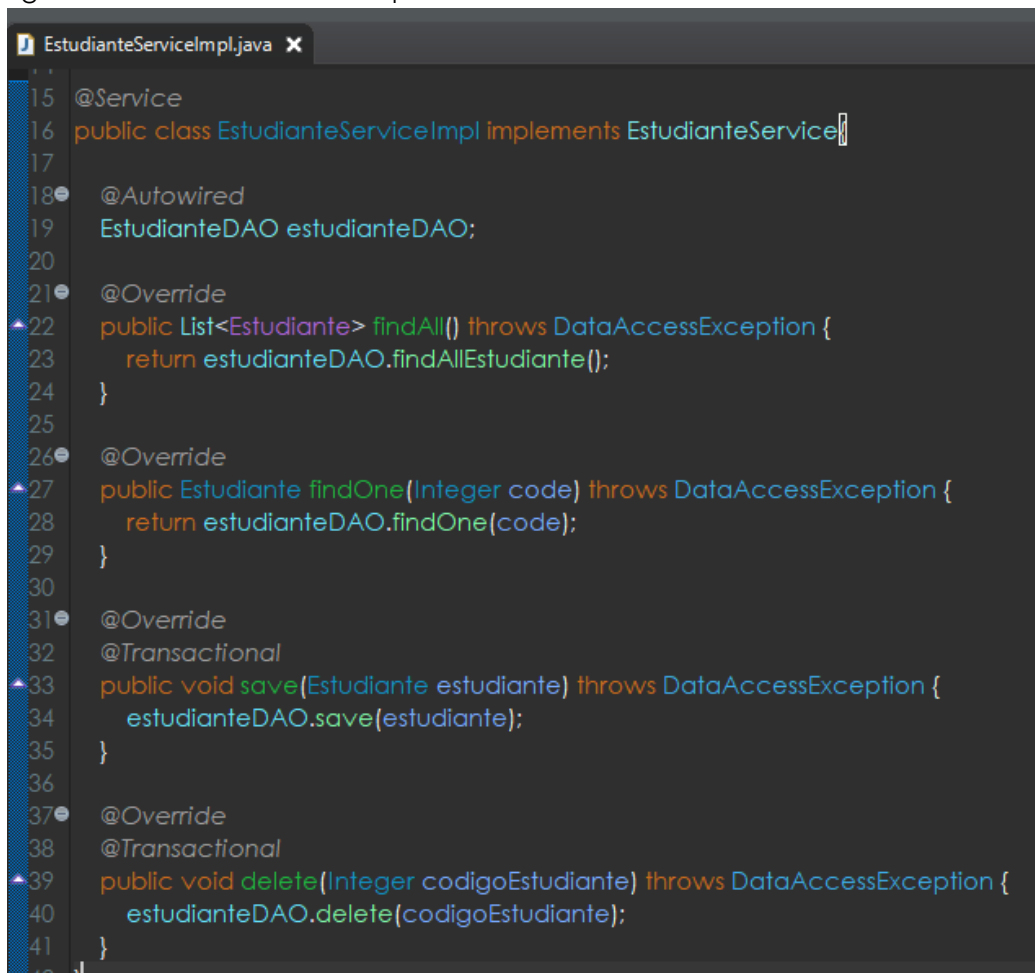
- Crear un paquete llamado **com.uca.capas.service**.
  - Dentro de com.uca.capas.service crear:
    - **EstudianteService Interface**
    - **EstudianteService Implementation.**

figura 10. EstudianteService interface



```
1 package com.uca.capas.service;
2
3 import java.util.List;
4
5
6
7
8
9 public interface EstudianteService {
10
11     public List<Estudiante> findAll() throws DataAccessException;
12
13     public Estudiante findOne(Integer code) throws DataAccessException;
14
15     public void save(Estudiante estudiante) throws DataAccessException;
16
17     public void delete(Integer codigoEstudiante) throws DataAccessException;
18 }
19
```

figura 11. EstudianteServiceImpl



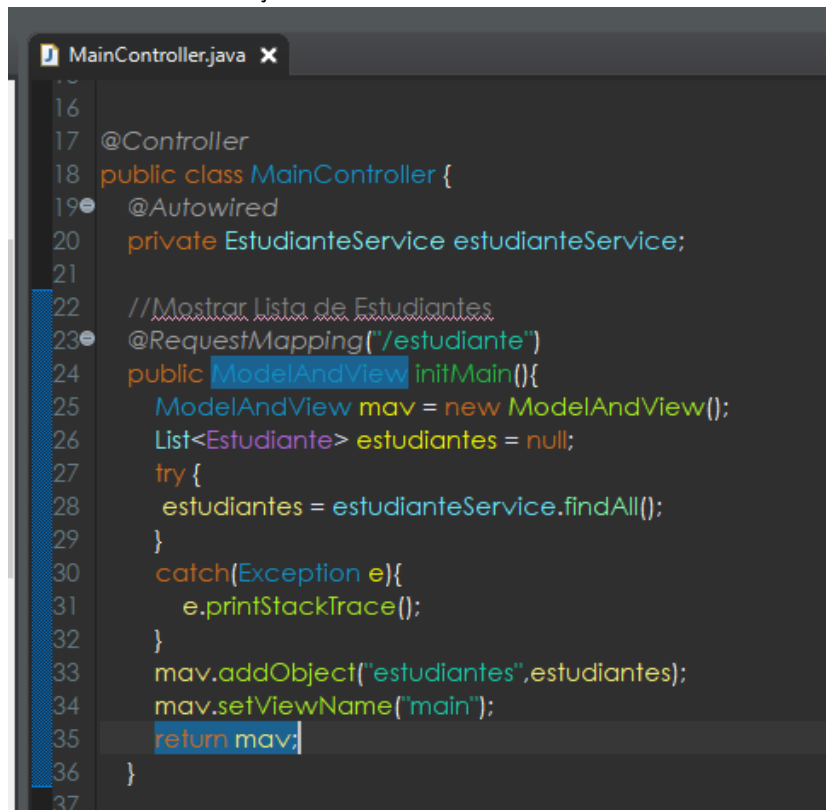
```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15 @Service
16 public class EstudianteServiceImpl implements EstudianteService {
17
18     @Autowired
19     EstudianteDAO estudianteDAO;
20
21     @Override
22     public List<Estudiante> findAll() throws DataAccessException {
23         return estudianteDAO.findAllEstudiante();
24     }
25
26     @Override
27     public Estudiante findOne(Integer code) throws DataAccessException {
28         return estudianteDAO.findOne(code);
29     }
30
31     @Override
32     @Transactional
33     public void save(Estudiante estudiante) throws DataAccessException {
34         estudianteDAO.save(estudiante);
35     }
36
37     @Override
38     @Transactional
39     public void delete(Integer codigoEstudiante) throws DataAccessException {
40         estudianteDAO.delete(codigoEstudiante);
41     }
42 }

```

## -----CAPA CONTROLADOR-----

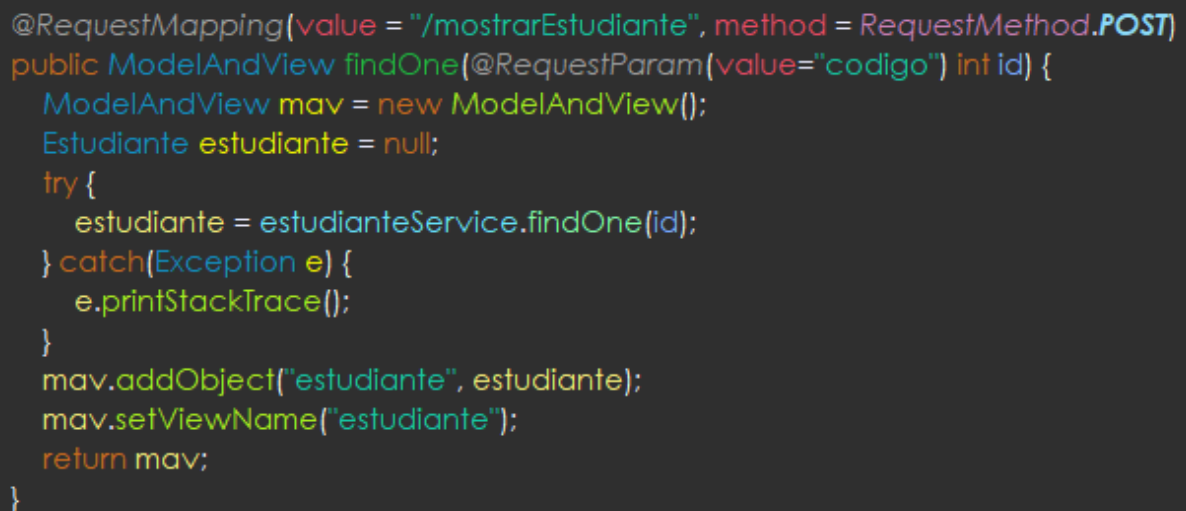
- Ahora la capa controlador hace contacto con la capa de servicio y no con el patrón DAO. Modificar en MainController.java:

figura 12. MainController.java



```
16
17 @Controller
18 public class MainController {
19     @Autowired
20     private EstudianteService estudianteService;
21
22     //Mostrar lista de Estudiantes
23     @RequestMapping("/estudiante")
24     public ModelAndView initMain(){
25         ModelAndView mav = new ModelAndView();
26         List<Estudiante> estudiantes = null;
27         try {
28             estudiantes = estudianteService.findAll();
29         }
30         catch(Exception e){
31             e.printStackTrace();
32         }
33         mav.addObject("estudiantes",estudiantes);
34         mav.setViewName("main");
35         return mav;
36     }
37 }
```

figura 12.1. MainController.java



```
@RequestMapping(value = "/mostrarEstudiante", method = RequestMethod.POST)
public ModelAndView findOne(@RequestParam(value="codigo") int id) {
    ModelAndView mav = new ModelAndView();
    Estudiante estudiante = null;
    try {
        estudiante = estudianteService.findOne(id);
    } catch(Exception e) {
        e.printStackTrace();
    }
    mav.addObject("estudiante", estudiante);
    mav.setViewName("estudiante");
    return mav;
}
```

- Agregar controladores para hacer **save** y **delete**.

figura 12.2. MainController.java

```
@PostMapping("/save")
public ModelAndView guardar(@Valid @ModelAttribute Estudiante estudiante, BindingResult result){
    ModelAndView mav = new ModelAndView();
    if(result.hasErrors()) {
        mav.setViewName("agregarEstudiante");
    }
    else{
        estudianteService.save(estudiante);
        List<Estudiante> estudiantes = null;
        try {
            estudiantes = estudianteService.findAll();
        }
        catch(Exception e){
            e.printStackTrace();
        }
        mav.addObject("estudiantes",estudiantes);
        mav.setViewName("listaEstudiantes");
    }

    return mav;
}
```

figura 12.3. MainController.java

```
@RequestMapping(value = "/borrarEstudiante", method = RequestMethod.POST)
public ModelAndView delete(@RequestParam(value="codigo") int id) {
    ModelAndView mav = new ModelAndView();
    List<Estudiante> estudiantes = null;
    try {
        estudianteService.delete(id);
        estudiantes = estudianteService.findAll();
    }
    catch(Exception e){
        e.printStackTrace();
    }
    mav.addObject("estudiantes",estudiantes);
    mav.setViewName("main");
    return mav;
}
```

figura 12.4. MainController.java

```
@GetMapping("/insertarEstudiante")
public ModelAndView inicio(){
    ModelAndView mav = new ModelAndView();
    mav.addObject("estudiante",new Estudiante());
    mav.setViewName("agregarEstudiante");
    return mav;
}
```



## -----CAPA DE VISTA-----

- En **main.html** agregar botones para **insertar un nuevo estudiante** y para **borrar un estudiante por código**.

figura 13. main.html

```
<form th:action= "@{/insertarEstudiante}">
  <input type="submit" value="Insertar nuevo estudiante">
</form><br>
<form th:action= "@{/borrarEstudiante}" method = "post">
  <label>Borrar estudiante por codigo: </label><input type="number" name="codigo"><br>
  <input type="submit" value="Borrar">
</form>
```

- Crear un formulario para insertar un nuevo estudiante, para ello se creará un nuevo html **"agregarEstudiante.html"**.

figura 14. agregarEstudiante.html

```
agregarEstudiante.html x
1 <!-- nuevo -->
2
3 <meta charset="ISO-8859-1">
4 <title>Nuevo estudiante</title>
5 </head>
6 <body>
7
8
9 <h1>Ingresar nuevo estudiante</h1>
10 <form th:action="@{/save}" th:object="{estudiante}" method="post">
11
12   <label>Nombre: </label>
13   <input th:field="{nombre}" type="text"/>
14   <span th:errors="{nombre}" style="color: #E81505"></span><br>
15
16   <label>Apellido: </label>
17   <input th:field="{apellido}" type="text"/>
18   <span th:errors="{apellido}" style="color: #E81505"></span><br>
19
20   <label>Edad (No menor a 18 años):</label>
21   <input type = "number" th:field = "{edad}"><br>
22   <span th:errors = "{edad}" style = "color: #E81505"></span><br>
23
24   <label>Activo</label>
25   <input type = "radio" th:field = "{estado}" th:value = "{true}">
26   <label>Inactivo</label>
27   <input type = "radio" th:field = "{estado}" th:value = "{false}"><br>
28
29   <input type="submit" value="Guardar">
30 </form>
31 </body>
```

- Crear un html llamado **"listaEstudiantes.html"** donde se mostrará la lista de estudiantes actualizada con un botón de **"Regresar"**.

figura 15. listaEstudiantes.html

```

1 listaEstudiantes.html x
2 <!-- HTML IS IT? http://www.htmleat.org -->
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Insert title here</title>
6 </head>
7 <body>
8
9 <table>
10 <thead>
11 <th>Nombre</th>
12 <th>Apellido</th>
13 <th>Edad</th>
14 <th>Estado</th>
15 </thead>
16 <th:block th:each="estudiante, row: ${estudiantes}">
17 <tr>
18 <th th:text="${estudiante.codigoEstudiante}"/>
19 <th th:text="${estudiante.nombre}"/>
20 <th th:text="${estudiante.apellido}"/>
21 <th th:text="${estudiante.edad}"/>
22 <th th:text="${estudiante.estadoDelegado}"/>
23 </tr>
24 </th:block>
25 </table>
26 <br>
27 <form th:action="@{/estudiante}">
28 <input type="submit" value="Regresar">
29 </form>
30

```

## FUNCIONAMIENTO:

←
→
↻
ⓘ localhost:8080/estudiante

Nombre	Apellido	Edad	Estado
1	Karla	Morales	22 Activo
2	Ayleen	Alfaro	20 Inactivo
3	Wilfredo	Maqueen	20 Activo
4	Monica	Herrera	22 Activo
5	Alan	Brito	29 Inactivo

Buscar por codigo de estudiante:

Borrar estudiante por codigo:

## Ingresar nuevo estudiante

Nombre:

Apellido:

Edad (No menor a 18 años):

Activo ☐ Inactivo ☐

## Ingresar nuevo estudiante

Nombre:  El campo no debe contener mas de 30 caracteres

Apellido:  Este campo no puede estar vacio

Edad (No menor a 18 años):  No puede ser menor a 18 años

Activo ☒ Inactivo ☐

	Nombre	Apellido	Edad	Estado
1	Karla	Morales	22	Activo
2	Ayleen	Alfaro	20	Inactivo
3	Wilfredo	Maqueen	20	Activo
4	Monica	Herrera	22	Activo
5	Alan	Brito	29	Inactivo
6	Manuez	Mendoza	32	Activo

### Data Output

	id_estudiante integer	nombre character varying (30)	apellido character varying (30)	edad integer	estado boolean
1	1	Karla	Morales	22	true
2	2	Ayleen	Alfaro	20	false
3	3	Wilfredo	Maqueen	20	true
4	4	Monica	Herrera	22	true
5	5	Alan	Brito	29	false
6	6	Manuez	Mendoza	32	true

← → ↻ ⓘ localhost:8080/estudiante?

	Nombre	Apellido	Edad	Estado
1	Karla	Morales	22	Activo
2	Ayleen	Alfaro	20	Inactivo
3	Wilfredo	Maqueen	20	Activo
4	Monica	Herrera	22	Activo
5	Alan	Brito	29	Inactivo
6	Manuez	Mendoza	32	Activo

Buscar por codigo de estudiante:

Buscar

Insertar nuevo estudiante

Borrar estudiante por codigo: 6

Borrar

← → ↻ ⓘ localhost:8080/borrarEstudiante

	Nombre	Apellido	Edad	Estado
1	Karla	Morales	22	Activo
2	Ayleen	Alfaro	20	Inactivo
3	Wilfredo	Maqueen	20	Activo
4	Monica	Herrera	22	Activo
5	Alan	Brito	29	Inactivo

Regresar

#### Data Output

	id_estudiante integer	nombre character varying (30)	apellido character varying (30)	edad integer	estado boolean
1	1	Karla	Morales	22	true
2	2	Ayleen	Alfaro	20	false
3	3	Wilfredo	Maqueen	20	true
4	4	Monica	Herrera	22	true
5	5	Alan	Brito	29	false

### Relaciones entre entidades

@OneToMany, @ManyToOne

- Crear tabla “**computadora**”, un estudiante puede tener distintas marcas de computadoras (**HP, DELL, Mac, Lenovo, Acer,Toshiba,Samsung,Asus**)

figura 16. Tabla computadora

```
CREATE TABLE computadora(  
    id_computadora serial NOT NULL PRIMARY KEY,  
    marca character varying (30),  
    id_estudiante integer  
);
```

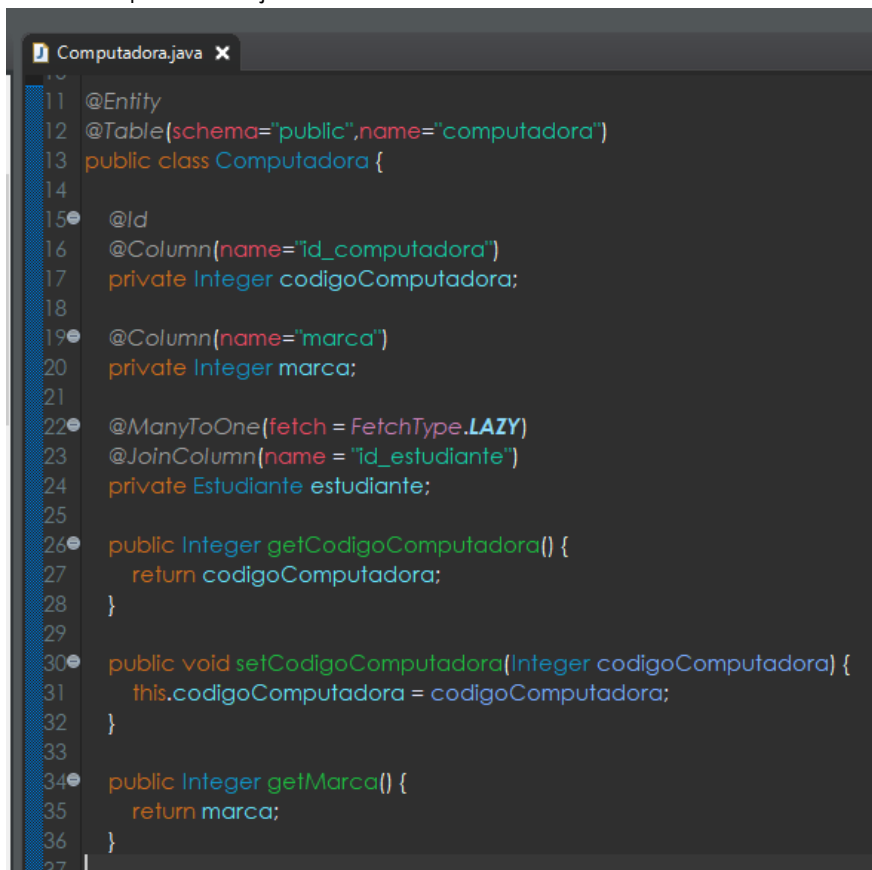
- Relación: **estudiante 1:N computadora**
- Agregar la llave foránea **id\_estudiante** en la tabla **computadora**.

figura 17. Llave foránea id\_estudiante

```
ALTER TABLE computadora ADD CONSTRAINT id_estudiante FOREIGN KEY (id_estudiante)  
REFERENCES estudiante (id_estudiante);
```

- En **com.uca.capas.domain** hacer una clase llamada **Computadora.java**, así mismo agregar **entidad** y **columnas** a la clase:
  - Hacer relación **@ManyToOne: estudiante 1:N computadora**.

figura 19. Computadora.java.



```
11 @Entity  
12 @Table(schema="public",name="computadora")  
13 public class Computadora {  
14  
15     @Id  
16     @Column(name="id_computadora")  
17     private Integer codigoComputadora;  
18  
19     @Column(name="marca")  
20     private Integer marca;  
21  
22     @ManyToOne(fetch = FetchType.LAZY)  
23     @JoinColumn(name = "id_estudiante")  
24     private Estudiante estudiante;  
25  
26     public Integer getCodigoComputadora() {  
27         return codigoComputadora;  
28     }  
29  
30     public void setCodigoComputadora(Integer codigoComputadora) {  
31         this.codigoComputadora = codigoComputadora;  
32     }  
33  
34     public Integer getMarca() {  
35         return marca;  
36     }  
37 }
```

- Relación: **estudiante 1:N computadora**
- Relación **@OneToMany** en **Estudiante.java**

figura 20. Estudiante.java.

```
@OneToMany(mappedBy = "estudiante", fetch = FetchType.EAGER)  
private List<Computadora> computadoras;
```

## ¿Cómo manejar entidades relacionas?

### -----CAPA CONTROLADOR-----

figura 21. MainController.java.

```
@RequestMapping("/")
public ModelAndView Main(){
    ModelAndView mav = new ModelAndView();
    mav.setViewName("main");
    return mav;
}

@RequestMapping(value = "/mostrarEstudiante", method = RequestMethod.POST)
public ModelAndView findOne(@RequestParam(value="codigo") int id) {
    ModelAndView mav = new ModelAndView();
    Estudiante estudiante = estudianteService.findOne(id);
    mav.addObject("estudiante", estudiante);
    mav.setViewName("estudiante");
    return mav;
}
```

### -----CAPA DE VISTA-----

figura 22. main.html.

```
main.html x
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Lista de estudiantes</title>
6 </head>
7 <body>
8
9 <form th:action="@{/mostrarEstudiante}" method="post">
10 <label>Buscar por código de estudiante: </label><input type="number" name="codigo"><br>
11 <input type="submit" value="Buscar">
12 </form>
13 <br>
14
15 </body>
16 </html>
```

figura 23. estudiante.html.

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Mostrar estudiante por llave primaria</title>
6 </head>
7 <body>
8 <label>Computadoras del estudiante</label> <label th:text="${estudiante.nombre}"/></label>
9 <table>
10 <thead>
11 <tr>
12 <th>Id Computadora</th>
13 <th>Marca</th>
14 </tr>
15 </thead>
16 <tbody>
17 <tr th:each="comp : ${estudiante.computadoras}">
18 <td th:text="${comp.codigoComputadora}"/></td>
19 <td th:text="${comp.marca}"/></td>
20 </tr>
21 </tbody>
22 </table>
23 <br>
24 <a href = "/estudiante">Regresar</a>
25
26
27 </body>
28 </html>
```

## FUNCIONAMIENTO:

