



DIPLOMADO
**Desarrollo de sistemas con
tecnología Java**

Módulo 8
Persistencia con Spring Data

Dr. Omar Mendoza González

omarmendoza564@aragon.unam.mx

Relaciones

En JPA, las relaciones entre entidades permiten modelar asociaciones entre las clases de entidades de forma similar a cómo se representan en una base de datos relacional.

Tipos de relaciones

Many-to-One

Many-to-Many

One-to-Many

One-to-One

Anotaciones

@ManyToOne

@ManyToMany

@OneToMany

@OneToOne

Relaciones

- @ManyToOne
 - Existe una relación **Many-To-One** entre entidades cuando una entidad hace referencia a otra entidad mediante una **FK**, que apunta a una **PK** de la entidad referenciada.
 - Cada alumno está asociado a un estado en una relación unidireccional.

```
@ManyToOne @JoinColumn(name = "id_estado")
private Estado estado;
```

Relaciones

- **@ManyToMany**
 - La relación **Many-To-Many** permite que varias filas de una entidad estén asociadas a múltiples filas de otra entidad.
 - Este tipo de relación se implementa mediante una **tabla intermedia** que contiene las **FK** de ambas entidades, lo que permite establecer una asociación bidireccional.
 - Un **alumno** puede estar inscrito en **varios grupos**, y un **grupo** puede tener varios **alumnos**.
 - Esta relación se representa mediante una **tabla intermedia** llamada **Alumnos_Grupos** que contiene las FK de **Alumno** y **Grupo**.

Relaciones

- @ManyToMany

@ManyToMany

@JoinTable(

 name = "Alumnos_Grupos",

 joinColumns = @JoinColumn(name = "matricula",

 referencedColumnName = "matricula"),

 inverseJoinColumns = @JoinColumn(name = "id_grupo",

 referencedColumnName = "id_grupo")

)

private Set<Grupo> grupos = new HashSet<>();

Relaciones

- **@OneToMany**
 - En una relación **One-To-Many**, cada fila de una entidad hace referencia a varios registros secundarios en otra entidad.
 - Lo importante es que los registros secundarios **no pueden tener varios padres**; es decir, cada registro en la entidad secundaria está asociado únicamente a un registro en la entidad principal.
 - Un **alumno** puede tener **muchas calificaciones**. Cada calificación está asociada con un solo alumno.

```
@OneToMany(mappedBy = "alumno")  
private List<Calificacion>  
    calificaciones = new ArrayList<>();
```

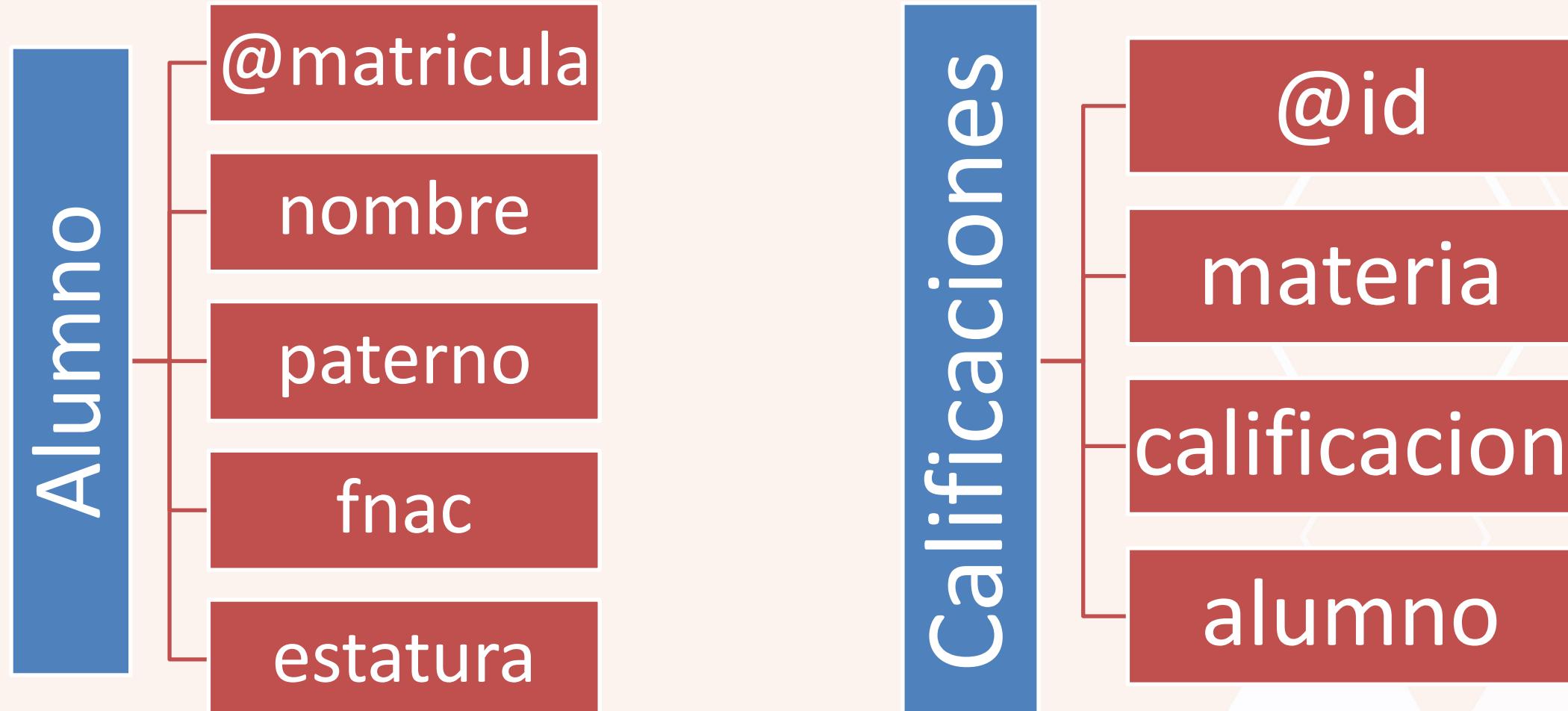
Relaciones

- **@OneToOne**
 - En una relación **One-To-One**, una entidad se asocia de forma exclusiva con una única entidad relacionada.
 - Esto significa que **cada fila** de una entidad está vinculada a **una y solo una fila** de otra entidad.
 - Ambas entidades tienen una correspondencia directa y única.

```
@OneToOne @JoinColumn(name = "perfil_id")
private Perfil perfil;
```

```
@OneToOne(mappedBy = "perfil")
private Alumno alumno;
```

Entidades



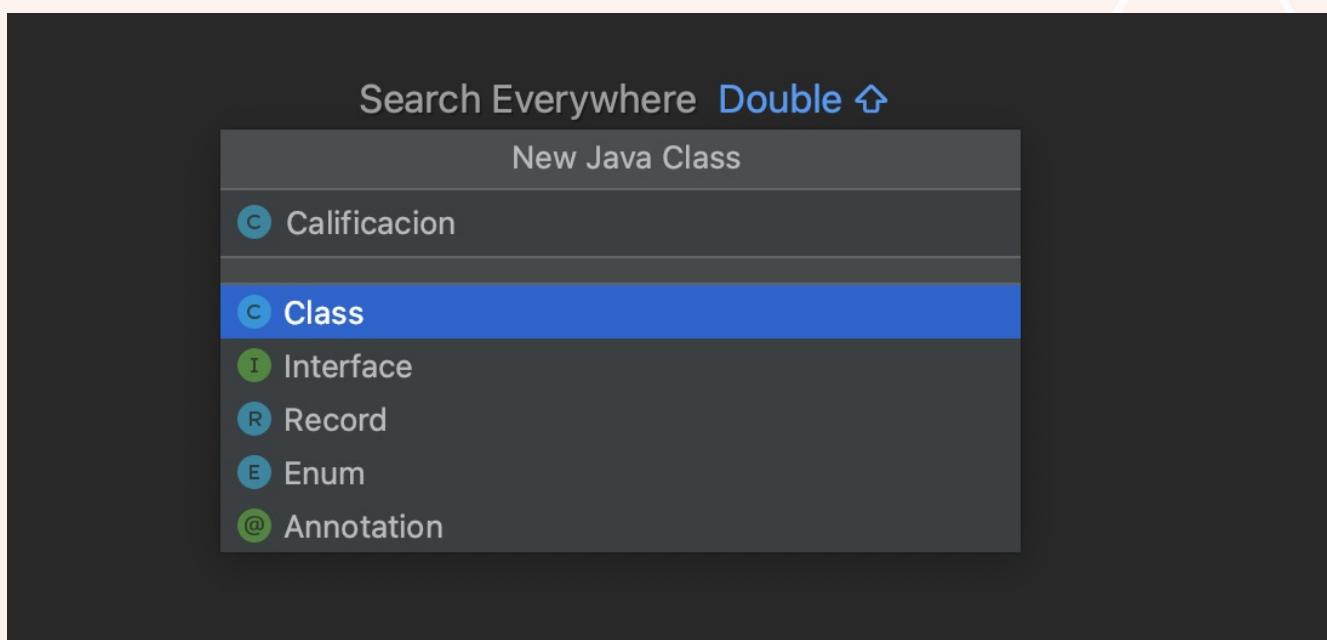
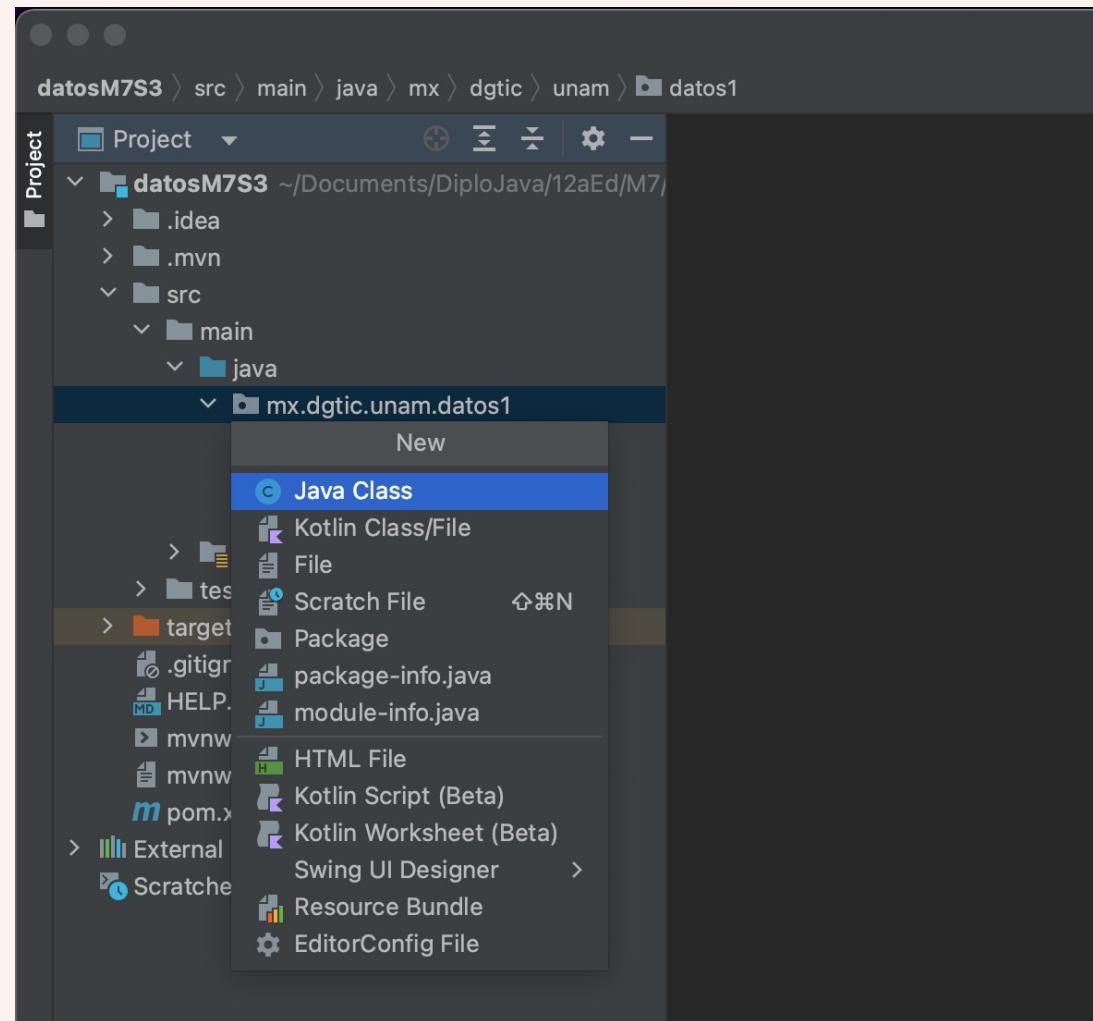
Relaciones



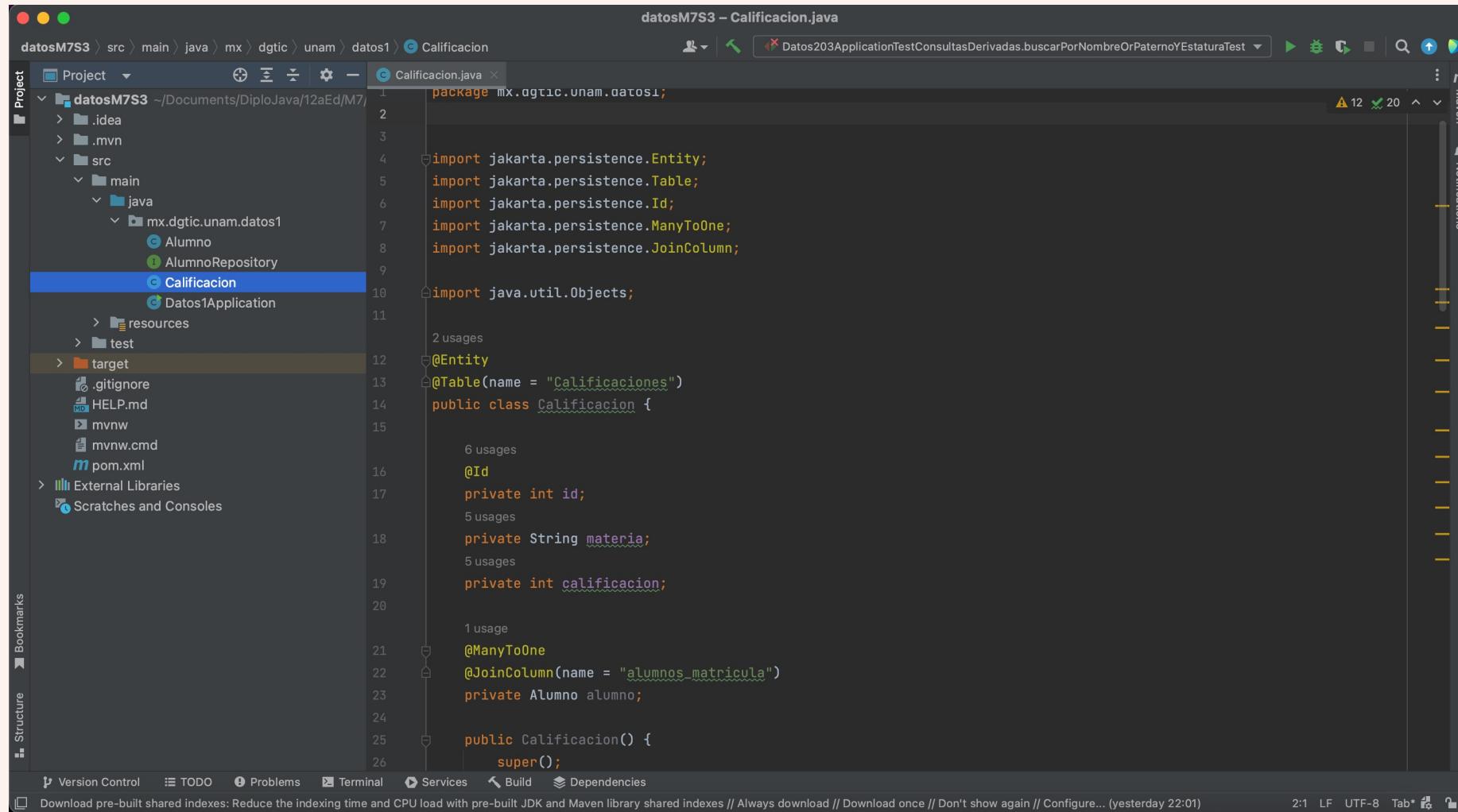
Relaciones



Entidad Calificacion



Entidad Calificacion



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "datosM7S3". It contains a "src" directory with "main" and "test" sub-directories. "main" has "java" and "resources" sub-directories. "java" contains "mx.dgtic.unam.datos1" which includes "Alumno", "AlumnoRepository", "Calificacion", and "Datos1Application".
- Editor:** The code editor displays "Calificacion.java".

```
package mx.dgtic.unam.datos1;
import jakarta.persistence.Entity;
import jakarta.persistence.Table;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.JoinColumn;
import java.util.Objects;

@Entity
@Table(name = "Calificaciones")
public class Calificacion {

    @Id
    private int id;
    private String materia;
    private int calificacion;

    @ManyToOne
    @JoinColumn(name = "alumnos_matricula")
    private Alumno alumno;

    public Calificacion() {
        super();
    }
}
```
- Toolbars and Status Bar:** The top bar shows tabs for "Calificacion.java", "Datas203ApplicationTestConsultasDerivadas.buscarPorNombreOrPaternoYEstaturaTest", and various icons. The bottom status bar shows "Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK and Maven library shared indexes // Always download // Download once // Don't show again // Configure... (yesterday 22:01)" and "2:1 LF UTF-8 Tab*".

Entidad Calificacion

Generar la relación con Alumno

```
1 usage  
@ManyToOne  
@JoinColumn(name = "alumnos_matricula")  
private Alumno alumno;
```

Entidad Alumno

Generar la relación con Calificación

```
@OneToMany(mappedBy = "alumno")
private List<Calificacion> calificaciones = new ArrayList<Calificacion>();
```

```
public List<Calificacion> getCalificaciones() {
    return calificaciones;
}

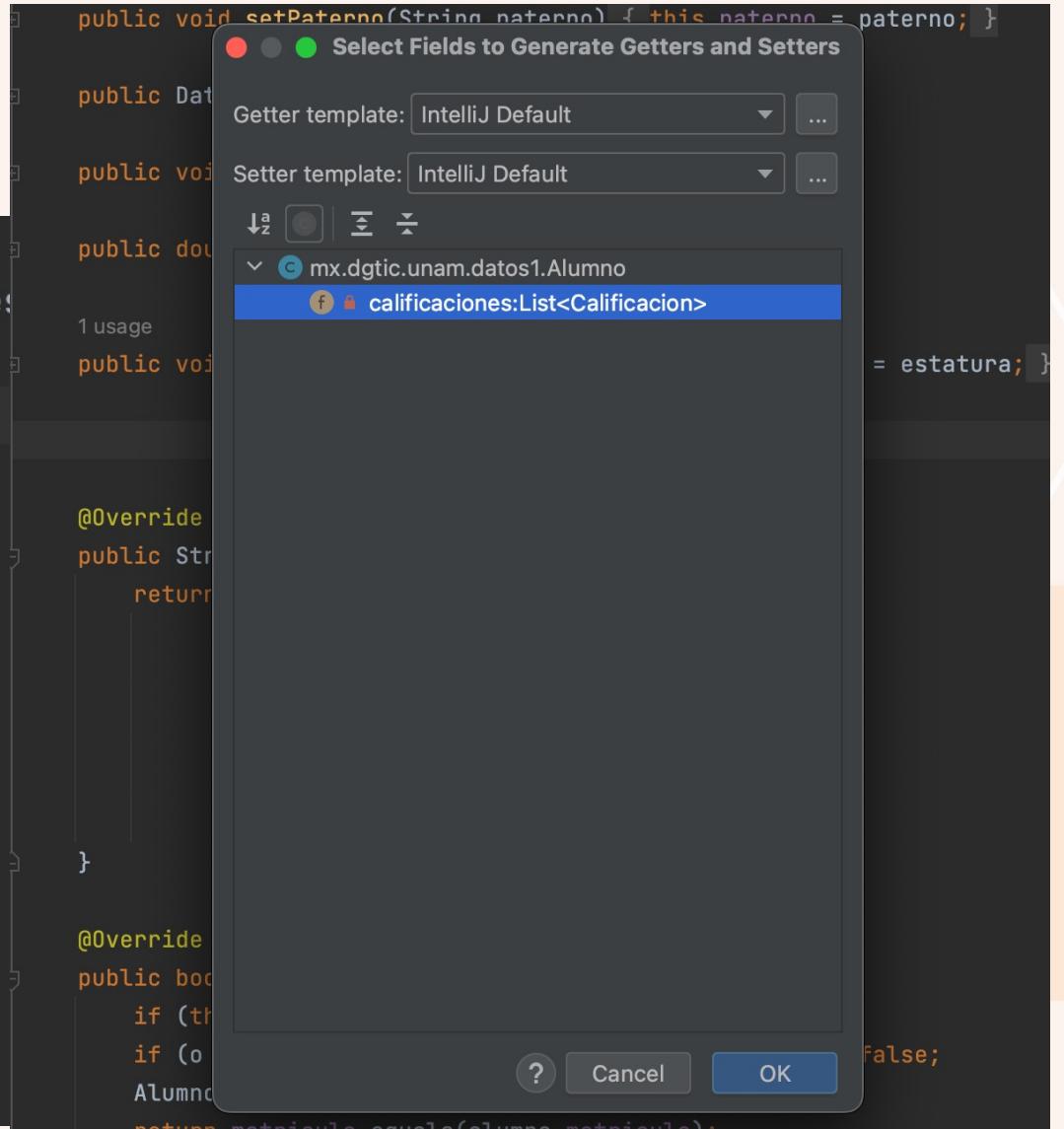
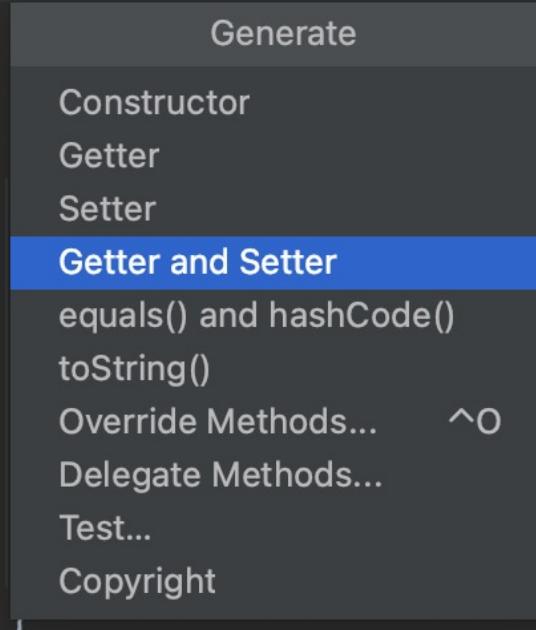
public void setCalificaciones(List<Calificacion> calificaciones) {
    this.calificaciones = calificaciones;
}
```

Entidad Alumno

Agregar métodos Set y Get de calificaciones

1 usage

```
public void setEstatura(double estatura) { this.estatura = e:
```



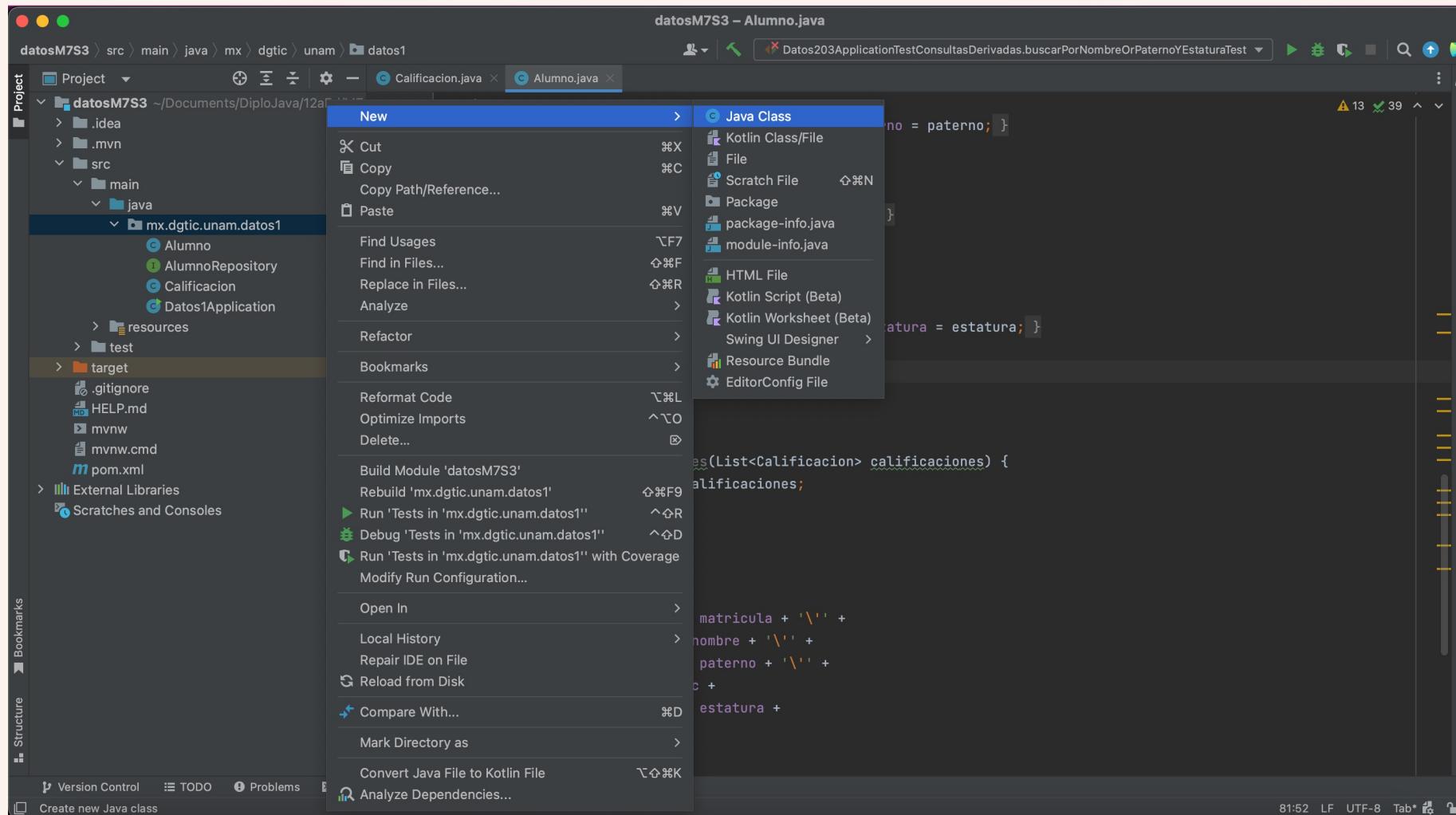
CalificacionRepository

CalificacionRepository

findByMateria(String materia)

findByCalificacion(Int calificacion)

CalificacionRepository

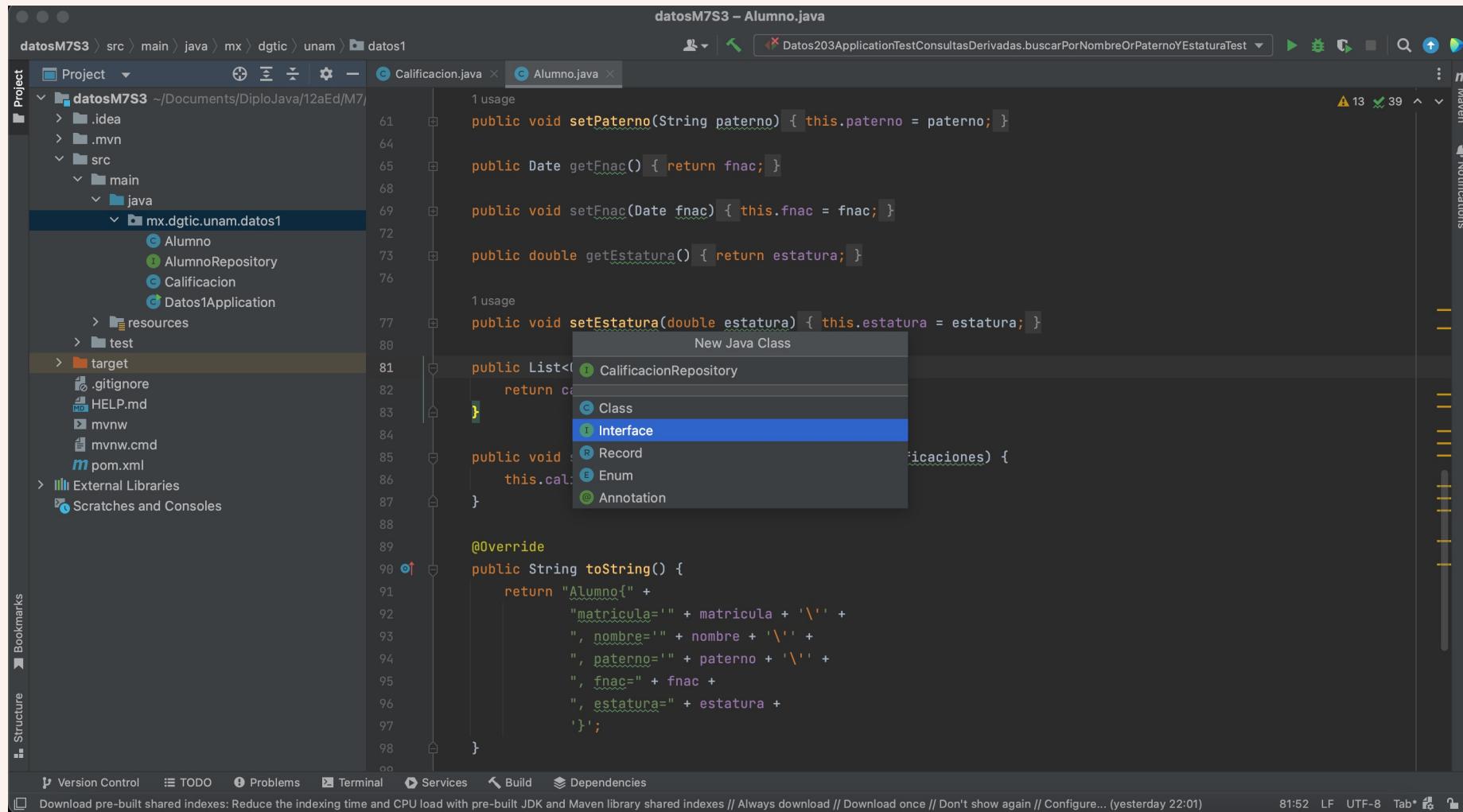


The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "datosM7S3". It contains a "src" directory with "main" and "java" sub-directories. Inside "java", there is a package "mx.dgtic.unam.datos1" which contains classes "Alumno", "AlumnoRepository", "Calificacion", and "Datos1Application".
- Code Editor:** The file "Alumno.java" is open, showing code related to student data. A context menu is open over the code, with "New" selected.
- New Context Menu:** The "New" submenu is open, showing options like "Java Class", "Kotlin Class/File", "File", etc. "Java Class" is highlighted.
- Code Snippet:** The code editor shows a snippet of Java code being typed:

```
no = paterno; }  
atura = estatura; }  
  
List<Calificacion> calificaciones) {  
alificaciones;  
  
matricula + '\n' +  
nombre + '\n' +  
paterno + '\n' +  
+  
estatura +
```
- Status Bar:** The status bar at the bottom right shows "81:52 LF UTF-8 Tab*".

CalificacionRepository



The screenshot shows a Java code editor in an IDE (IntelliJ IDEA) displaying the file `Alumno.java`. The code defines a class `Alumno` with various methods like `setPaterno`, `getFnac`, etc. A code completion dropdown is open at line 81, showing suggestions for the method `return c:`. The suggestion `CalificacionRepository` is highlighted in blue, indicating it is the most relevant or frequently used option. Other suggestions include `Class`, `Interface`, `Record`, `Enum`, and `Annotation`.

```
datosM7S3 – Alumno.java
1 usage
61     public void setPaterno(String paterno) { this.paterno = paterno; }
64
65     public Date getFnac() { return fnac; }
68
69     public void setFnac(Date fnac) { this.fnac = fnac; }
72
73     public double getEstatura() { return estatura; }
76
77     public void setEstatura(double estatura) { this.estatura = estatura; }
78
80     public List<Calificacion> getCalificaciones() {
81         return c: New Java Class
82     }
83
84     public void addCalificacion(Calificacion calificacion) {
85         this.calificaciones.add(calificacion);
86     }
87
88
89     @Override
90     public String toString() {
91         return "Alumno{" +
92             "matricula='" + matricula + '\'' +
93             ", nombre='" + nombre + '\'' +
94             ", paterno='" + paterno + '\'' +
95             ", fnac=" + fnac +
96             ", estatura=" + estatura +
97             '}';
98 }
```

Project Tree (Left):

- datosM7S3 (~/Documents/DiploJava/12aEd/M7)
- .idea
- .mvn
- src
 - main
 - java
 - mx.dgtic.unam.datos1
 - Alumno
 - AlumnoRepository
 - Calificacion
 - Datos1Application
 - resources
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
- External Libraries
- Scratches and Consoles

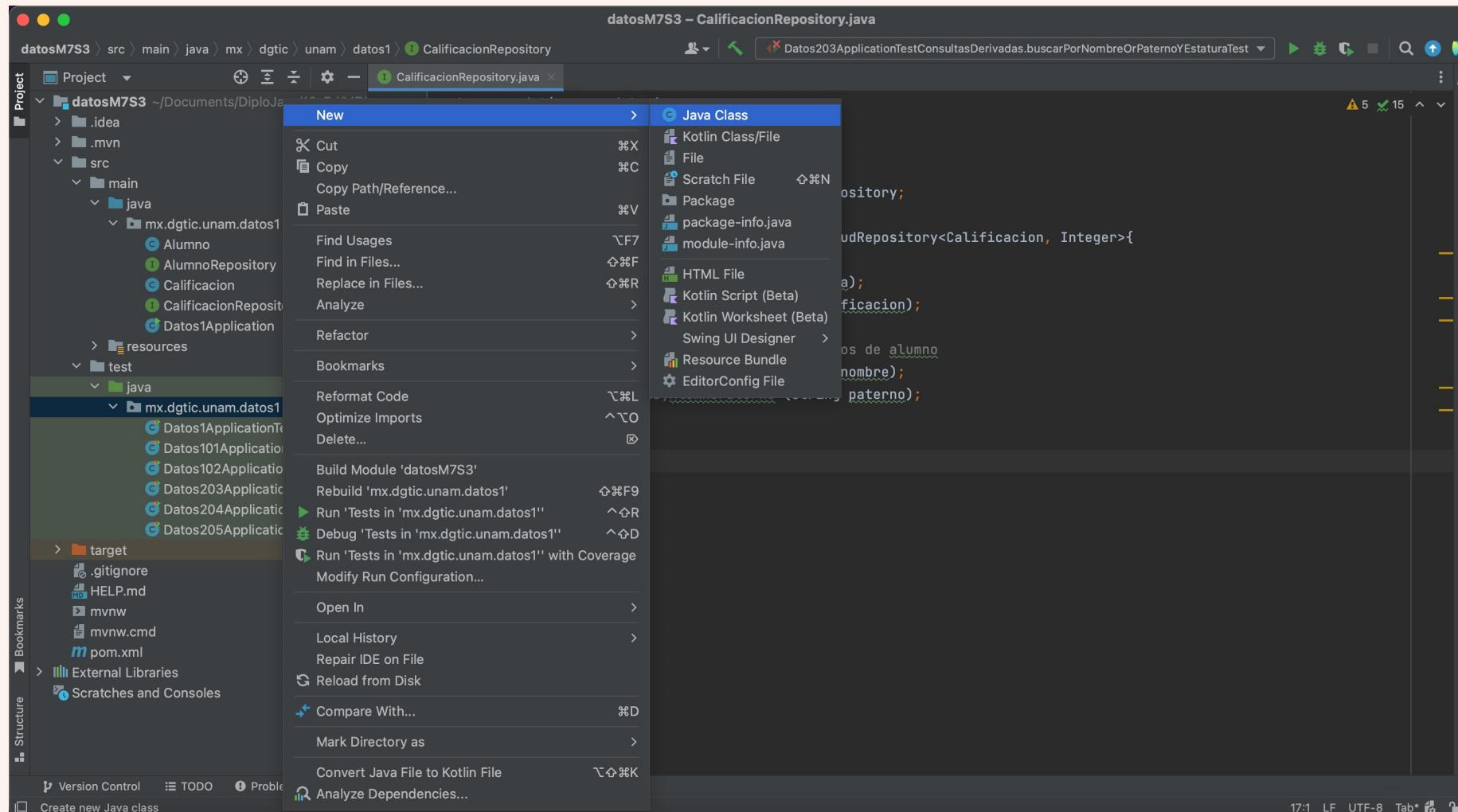
Toolbars and Status Bar (Bottom):

- Version Control
- TODO
- Problems
- Terminal
- Services
- Build
- Dependencies
- Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK and Maven library shared indexes // Always download // Download once // Don't show again // Configure... (yesterday 22:01)
- 81:52 LF UTF-8 Tab* 81:52

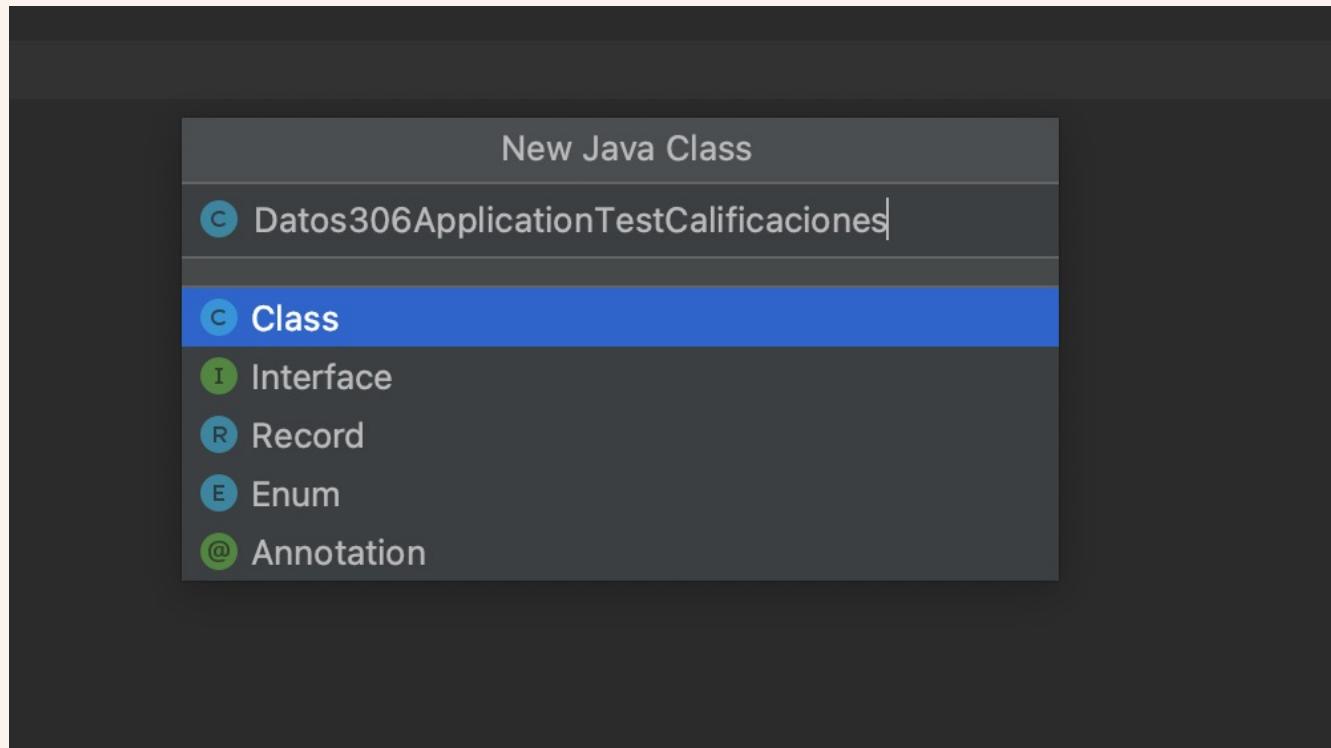
CalificacionRepository

```
I CalificacionRepository.java ×  
1 package mx.dgtic.unam.datos1;  
2  
3 import java.util.List;  
4  
5 import org.springframework.data.repository.CrudRepository;  
6  
7 public interface CalificacionRepository extends CrudRepository<Calificacion, Integer>{  
8  
9     List<Calificacion> findByMateria(String materia);  
10    List<Calificacion> findByCalificacion(int calificacion);  
11  
12    //Busqueda de calificaciones por medio de campos de alumno  
13    List<Calificacion> findByAlumnoNombre (String nombre);  
14    List<Calificacion> findByAlumnoPaterno (String paterno);  
15 }  
16 |
```

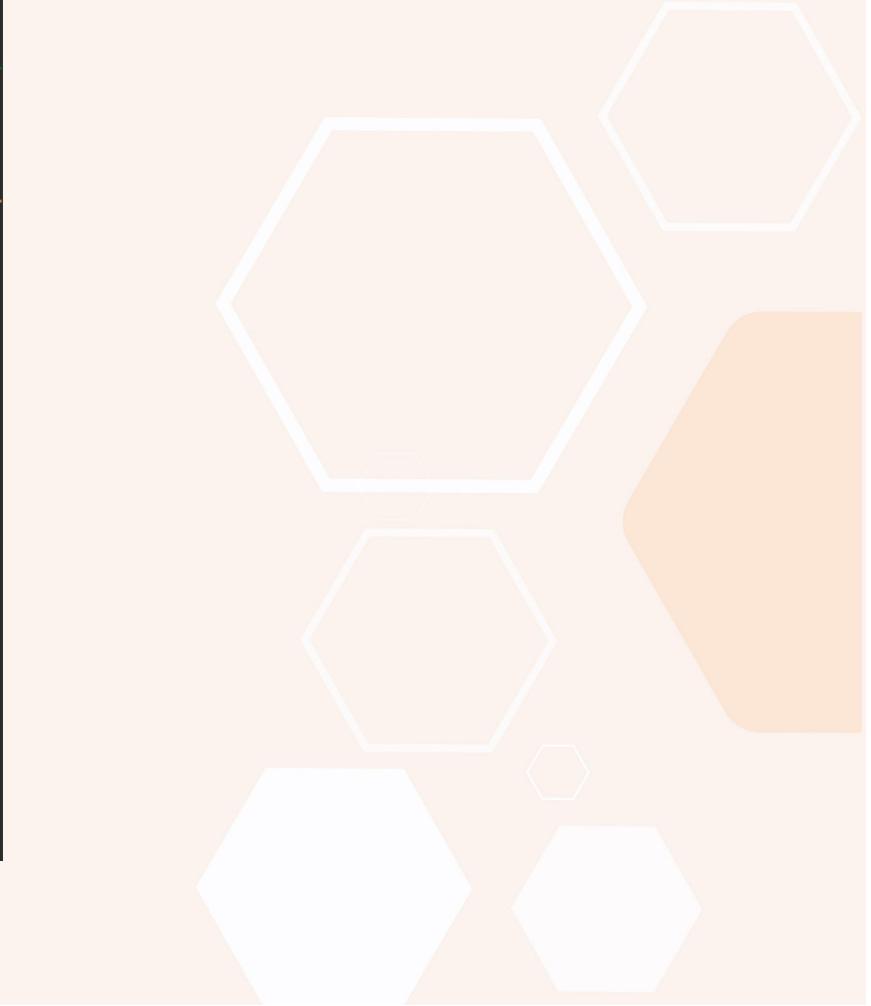
Nueva Prueba Unitaria



Nueva Prueba Unitaria



Nueva Prueba Unitaria



```
CalificacionRepository.java × Datos306ApplicationTestCalificaciones.java × : 2 23 ▲ ▼ ⋮  
9  
10 @SpringBootTest  
11 public class Datos306ApplicationTestCalificaciones {  
12     1 usage  
13     private static final String MATERIA = "BD";  
14     1 usage  
15     private static final int CALIFICACION = 8;  
16     @Autowired  
17     AlumnoRepository repositorioAlumno;  
18     4 usages  
19     @Autowired  
20     CalificacionRepository repositorioCalificacion;  
21     @Test  
22     void buscarTodosCalificacionTest() {  
23         Iterable<Calificacion> iterable = repositorioCalificacion.findAll();  
24         System.out.println("findAll Calificacion");  
25         iterable.forEach(System.out::println);  
26     }  
27     @Test  
28     void buscarPorMateriaTest() {  
29         List<Calificacion> lista = repositorioCalificacion.findByMateria(MATERIA);  
30         System.out.println("findByMateria");  
31         lista.forEach(System.out::println);  
32     }  
33     @Test  
34     void buscarPorAlumnoTest() {  
35     }
```

CalificacionRepository

CalificacionRepository

findByMateria(String materia)

findByCalificacion(Int calificacion)

findByAlumnoNombre(String nombre)

CalificacionRepository

```
//Busqueda de calificaciones por medio de campos de alumno  
1 usage  
List<Calificacion> findByAlumnoNombre (String nombre);
```

```
@Test  
void buscarCalificacionPorAlumnoNombreTest() {  
    Iterable<Calificacion> iterable = repositorioCalificacion.findByAlumnoNombre("Nadia");  
  
    System.out.println("findByAlumnoNombre Calificacion");  
    iterable.forEach(System.out::println);  
}
```

```
findByAlumnoNombre Calificacion  
Calificacion{id=1, materia='BD', calificacion=10, alumno=Alumno{matricula='2A', nombre='Nadia', paterno='Perez', fnac=2001-01-10}  
Calificacion{id=2, materia='POO', calificacion=9, alumno=Alumno{matricula='2A', nombre='Nadia', paterno='Perez', fnac=2001-01-10}  
Calificacion{id=3, materia='SI', calificacion=10, alumno=Alumno{matricula='2A', nombre='Nadia', paterno='Perez', fnac=2001-01-10}  
Calificacion{id=4, materia='BDII', calificacion=8, alumno=Alumno{matricula='2A', nombre='Nadia', paterno='Perez', fnac=2001-01-10}
```

Join Fetch

- **Join Fetch** es una de las opciones de las que dispone el estándar de JPA a la hora de **reducir el número de consultas** que se generan contra la base de datos, Algo que es bastante habitual y que **degrada el rendimiento**.
- Permite inicializar asociaciones o colecciones de valores junto con sus objetos principales mediante una única selección
- `select distinct a from Alumno a join a.calificaciones`
- `select distinct a from Alumno a join fetch a.calificaciones`

Join Fetch

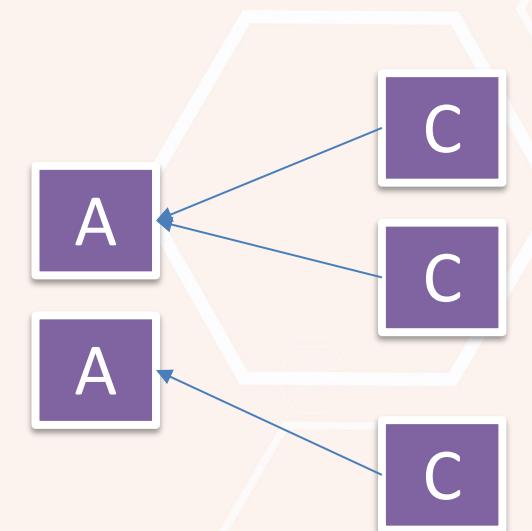
- Las dos consultas, está utilizando JOIN para consultar a todos los alumnos que tienen al menos una calificación asociado.
- La diferencia es
 - En la primera consulta, solo devuelve los Alumnos.
 - En la segunda consulta, está devolviendo los Alumnos **y** todas las Calificaciones asociadas.
- Si usa la segunda consulta, no se necesitará hacer una nueva consulta para acceder a la BD nuevamente para ver las Calificaciones de cada Alumno.

Join Fetch

```
@NamedQuery select distinct a from Alumno a join fetch a.calificaciones
```



findAllWithCalificaciones()



Join Fetch

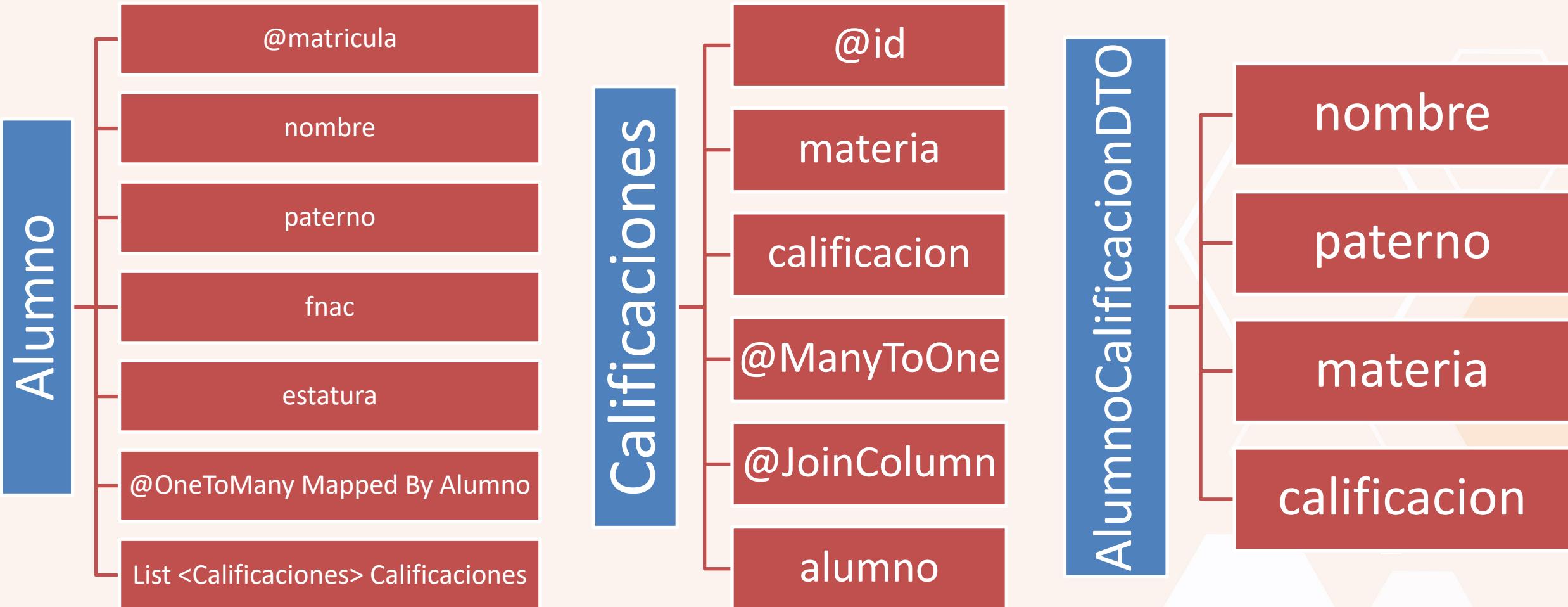
```
@Test
void buscarTodosConCalificacionesTest() {
    System.out.println(ALUMNO);
    System.out.println("buscarTodosConCalificaciones ");
    Iterable<Alumno> iterable =
        repositorioAlumno.buscarTodosConCalificaciones();

    for(Alumno a:iterable){
        System.out.println(a.getMatricula() + " " + a.getNombre() + " " + a.getPaterno());
        List<Calificacion> calificaciones = a.getCalificaciones();
        for(Calificacion c:calificaciones){
            System.out.println(c.getMateria() + " " + c.getCalificacion());
        }
    }
}
```

DTO

- DTOs (Data Transfer Objects)
- Ayuda a agrupar información de varias entidades generando una entidad propia
- El patrón DTO tiene como finalidad de crear un objeto plano (POJO) con una serie de atributos que puedan ser enviados o recuperados del servidor en **una sola invocación**
- De tal forma que un DTO puede contener información de **múltiples fuentes** o tablas y concentrarlas en una única clase simple.

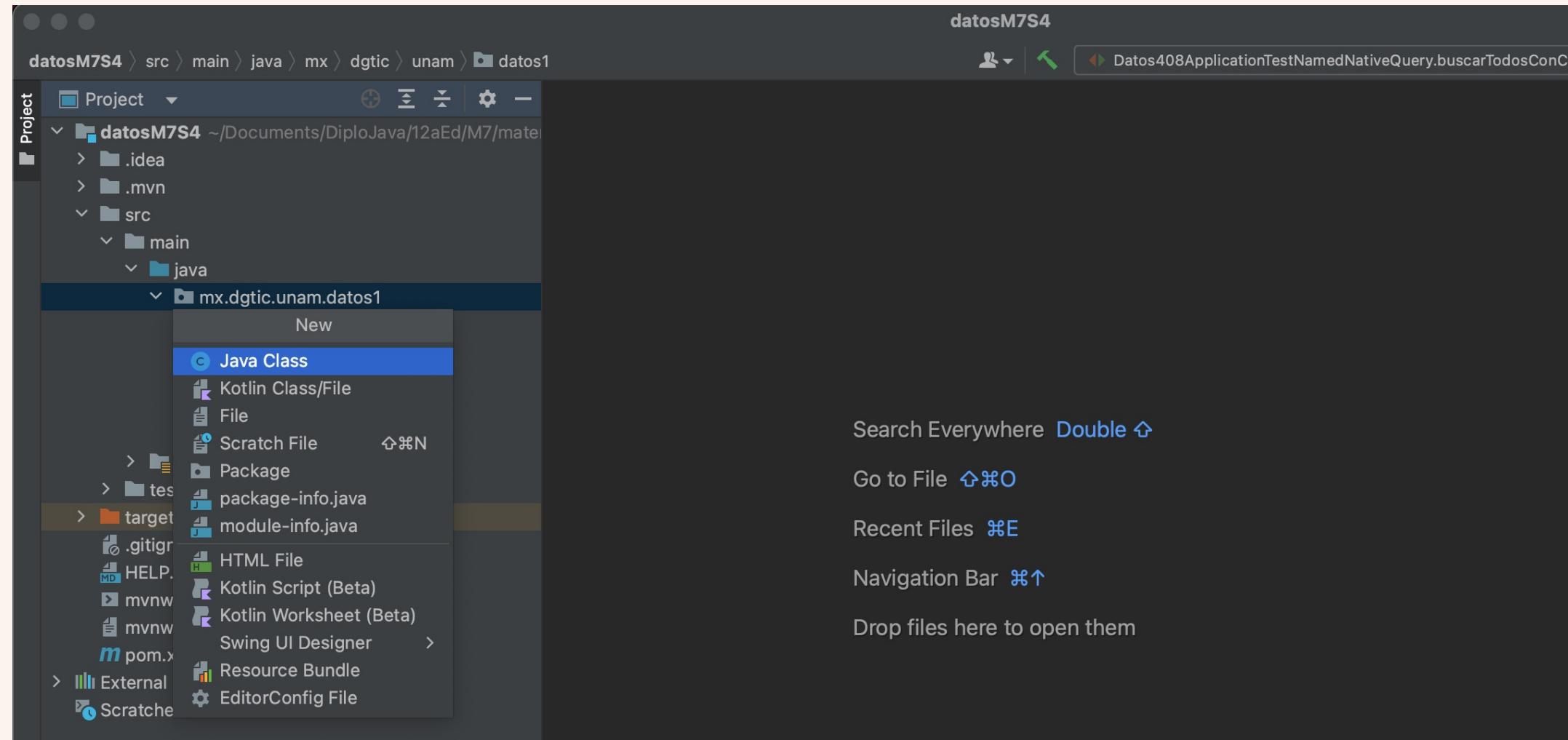
TDO



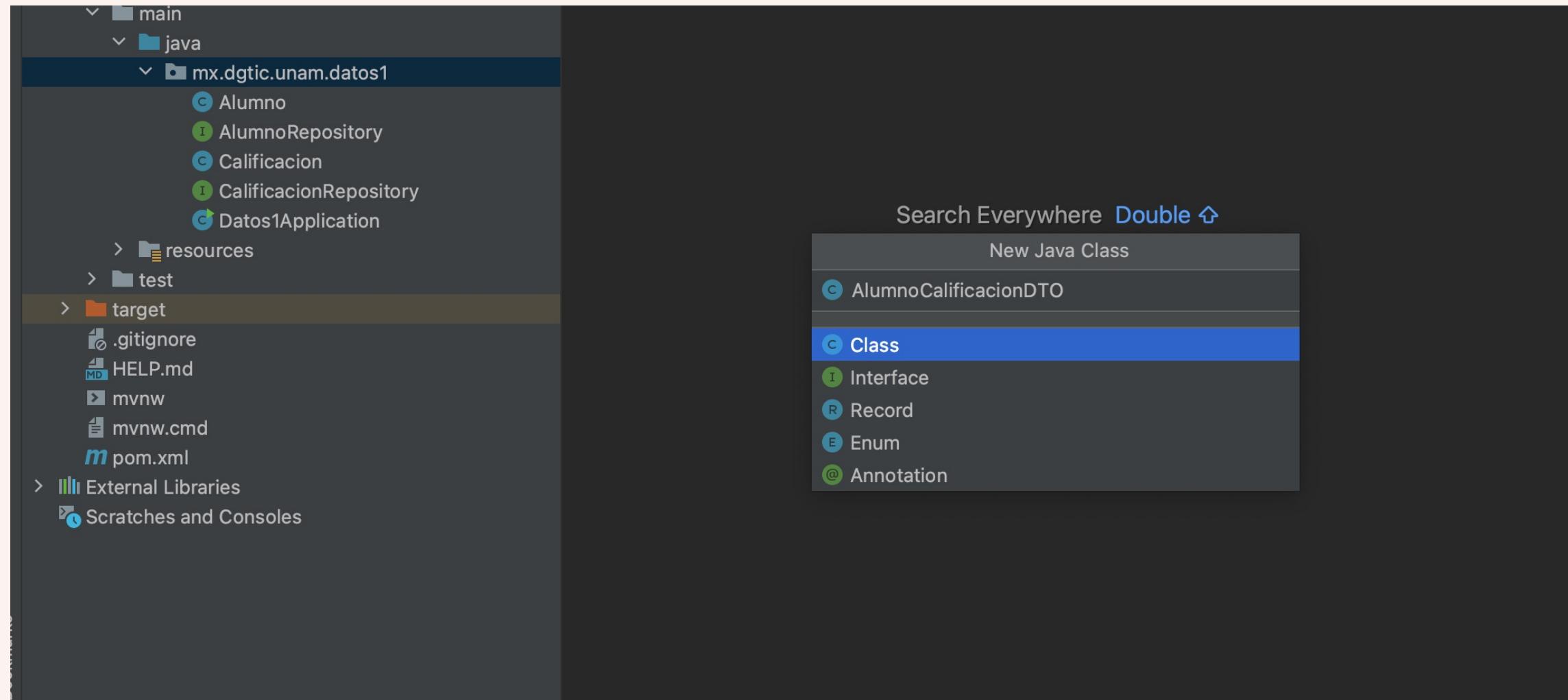
DTO Características

- **Solo lectura**
 - Dado que el objetivo de un DTO es utilizarlo como un objeto de transferencia entre el cliente y el servidor, es importante evitar tener operaciones de negocio o métodos que realicen cálculos sobre los datos, es por ello que solo se deben tener los métodos GET y SET de los respectivos atributos del DTO.
- **Serializable**
 - Si los objetos tendrán que viajar por la red, deberán de poder ser serializables, pero no solamente la clase en sí, sino que también todos los atributos que contenga el DTO deberán ser fácilmente serializables.

Crear la clase DTO



Crear la clase DTO



Crear la clase DTO

```
package mx.dgtic.unam.datos1;

public class AlumnoCalificacionDTO {
    private String nombre;
    private String paterno;
    private String materia;
    private int calificacion;
```

}

Generate

Constructor

Getter

Setter

Getter and Setter

equals() and hashCode()

toString()

Override Methods... ^O

Delegate Methods...

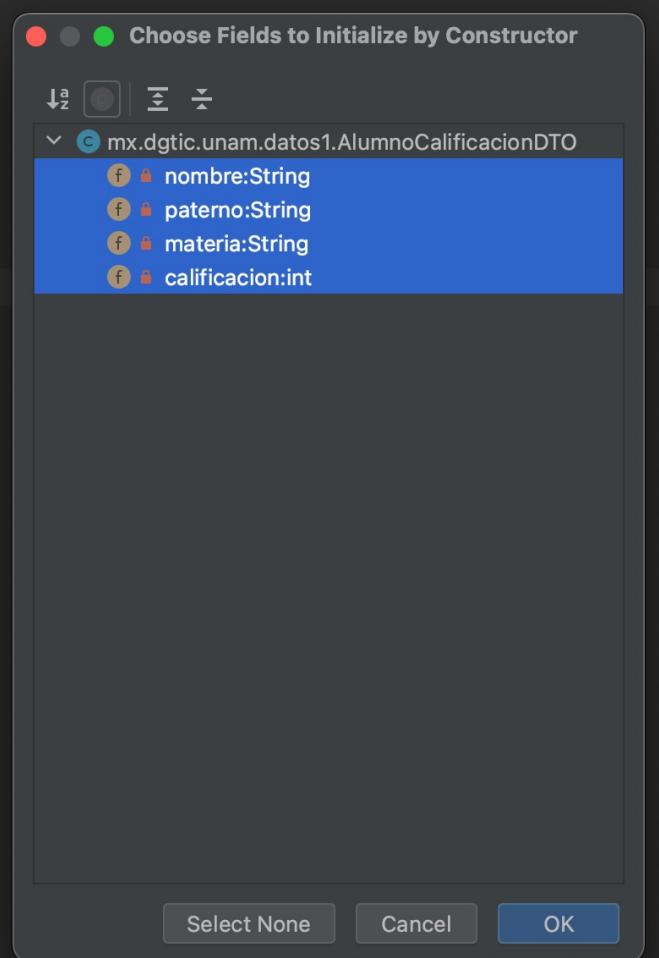
Test...

Copyright

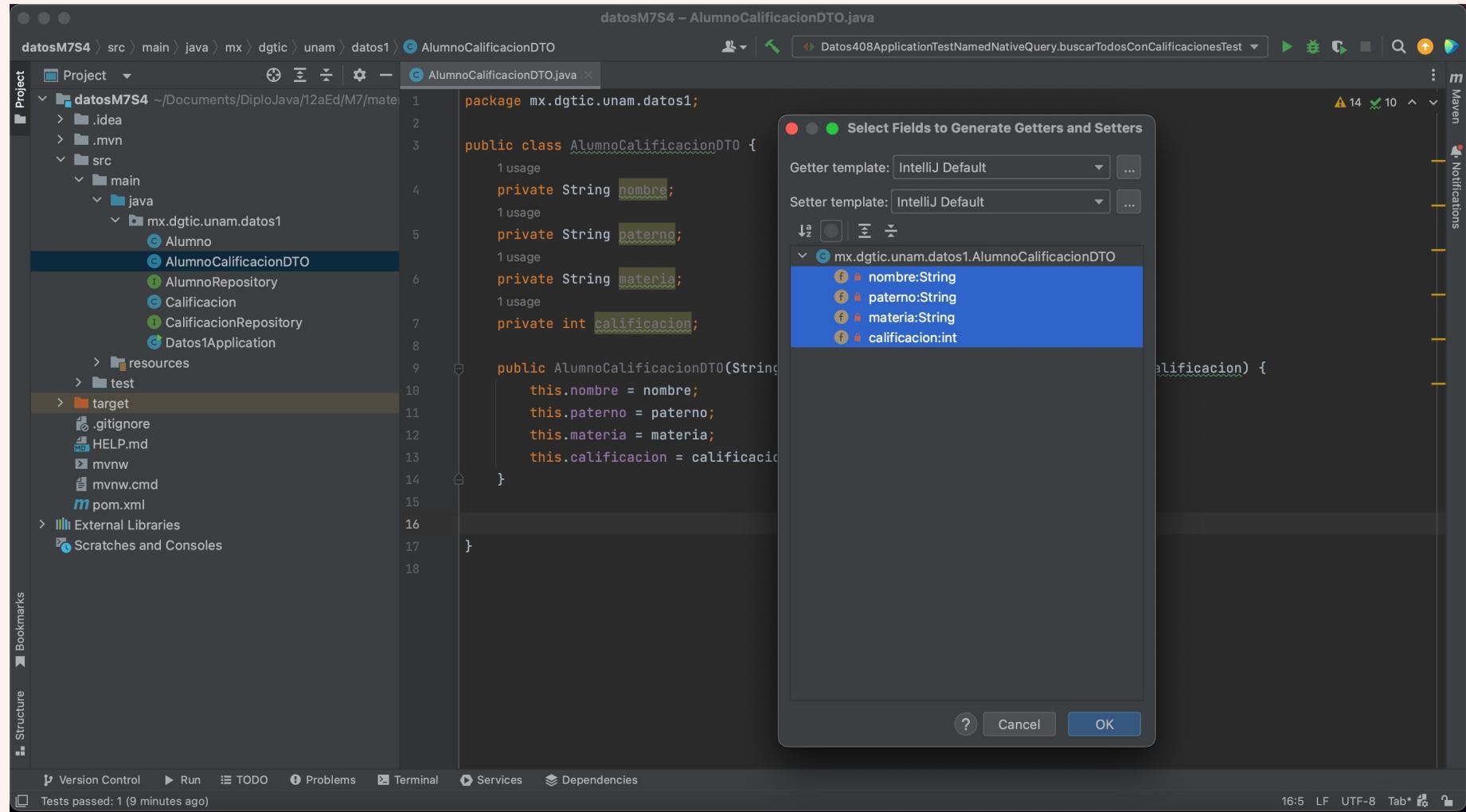
```
package mx.dgtic.unam.datos1;

public class AlumnoCalificacionDTO {
    private String nombre;
    private String paterno;
    private String materia;
    private int calificacion;
```

}



Crear la clase DTO



The screenshot shows the IntelliJ IDEA interface with the project 'datosM7S4' open. The 'src/main/java/mx/dgtic/unam/datos1' package contains several classes: Alumno, AlumnoCalificacionDTO, AlumnoRepository, Calificacion, CalificacionRepository, and Datos1Application. The 'AlumnoCalificacionDTO.java' file is currently selected and displayed in the editor. A code completion dialog titled 'Select Fields to Generate Getters and Setters' is open over the code. The dialog lists four fields: 'nombre: String', 'paterno: String', 'materia: String', and 'calificacion: int'. The 'Getter template' dropdown is set to 'IntelliJ Default' and the 'Setter template' dropdown is also set to 'IntelliJ Default'. At the bottom of the dialog are 'Cancel' and 'OK' buttons.

```
package mx.dgtic.unam.datos1;

public class AlumnoCalificacionDTO {
    private String nombre;
    private String paterno;
    private String materia;
    private int calificacion;

    public AlumnoCalificacionDTO(String nombre, String paterno, String materia, int calificacion) {
        this.nombre = nombre;
        this.paterno = paterno;
        this.materia = materia;
        this.calificacion = calificacion;
    }
}
```

AlumnoRepository

AlumnoRepository

findAlumnoCalificacionDTO()

AlumnoRepository

```
//DTO
@Query(value = "select distinct new mx.dgtic.unam.datos1.AlumnoCalificacionDTO(
    + \"a.nombre, a.paterno, c.materia, c.calificacion) "
    + "from Alumno a, Calificacion c "
    + "where a.matricula = c.alumno")
public List<AlumnoCalificacionDTO> findAlumnoCalificacionDTO();
```

Contacto

Dr. Omar Mendoza González

omarmendoza564@aragon.unam.mx