***Sistema De Gestión***

***De Calificaciones***

***Realizado Por:***

***Jeremy Alexander Ramos Segura***

***Daniel Alejandro Carmona Rodríguez***

Índice

[Introducción - 2 -](#_Toc100341964)

[Requisitos Funcionales, No funcionales y de Información - 3 -](#_Toc100341965)

[Requisitos Funcionales - 3 -](#_Toc100341966)

[Requisitos No Funcionales - 4 -](#_Toc100341967)

[Requisitos De Información - 4 -](#_Toc100341968)

[Diagrama De Estructura - 5 -](#_Toc100341969)

[Diagrama de clases. - 5 -](#_Toc100341970)

[Diagramas De Comportamiento - 6 -](#_Toc100341971)

[Diagrama de caso de uso. - 6 -](#_Toc100341972)

[Arquitectura del Software Aplicada - 7 -](#_Toc100341973)

[Modelo Vista Controlador - 7 -](#_Toc100341974)

[Patrones de diseño y ejemplo de cómo se han aplicado. - 8 -](#_Toc100341975)

[Patrón Builder - 8 -](#_Toc100341976)

[Patrón Singleton - 9 -](#_Toc100341977)

[Principios SOLID aplicadas y ejemplos. - 10 -](#_Toc100341978)

[S – Single Responsibility Principle (SRP) - 10 -](#_Toc100341979)

[O – Open/Closed Principle (OCP) - 11 -](#_Toc100341980)

[I – Interface Segregation Principle (ISP) - 12 -](#_Toc100341981)

[D – Dependency Inversion Principle (DIP) - 13 -](#_Toc100341982)

[Resumen - 13 -](#_Toc100341983)

[Diseño de Pruebas unitarias. - 14 -](#_Toc100341984)

[Tests - 14 -](#_Toc100341985)

[Ejemplo de uso y capturas del mismo. - 15 -](#_Toc100341986)

[Gestión de Alumnos - 15 -](#_Toc100341987)

[Creación de Alumnos - 15 -](#_Toc100341988)

[Muestreo de Alumnos - 15 -](#_Toc100341989)

[Gestión de Categorías - 16 -](#_Toc100341990)

[Creación de Categorías - 16 -](#_Toc100341991)

[Muestreo de Categorías - 16 -](#_Toc100341992)

[Modificación de Categorías - 16 -](#_Toc100341993)

[Gestión de Pruebas de Evaluación - 17 -](#_Toc100341994)

[Creación de Prueba de Evaluación - 17 -](#_Toc100341995)

[Modificación de las Pruebas de Evaluación - 17 -](#_Toc100341996)

[Exportación a Json - 18 -](#_Toc100341997)

[Importación de Json - 18 -](#_Toc100341998)

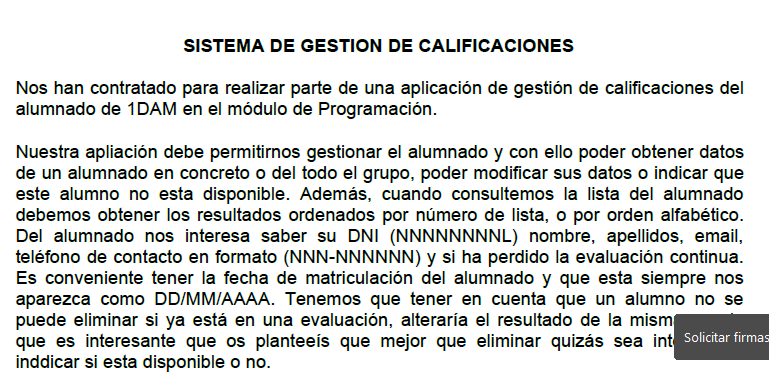
[Json - 18 -](#_Toc100341999)

# Introducción

En este proyecto expuesto en este pdf veremos cómo gestionamos la creación modificación y uso de nuestro programa de gestión de calificaciones de un centro.  
Veremos que la variedad de ideas expuesta es la correcta y que hemos tenido en cuenta todos los aspectos necesarios a relucir en nuestro proyecto.

Algunos aspectos que veremos en este pdf serán:

* Tipos de registros pensados y expuestos en nuestro proyecto
* Diagramas de Caso de Uso
* Diagramas de Clases
* Que arquitectura software se ha aplicado
* ¿Hemos aplicado SOLID?
* ¿Test?
* Ejemplos de Uso de nuestro Programa.

**Inicio de nuestro problema**

**Para más información sobre nuestro problema mirar el Readme Anclado a nuestro repositorio GitHub.**

# Requisitos Funcionales, No funcionales y de Información

Hemos proporcionado un tiempo esencial a pensar en este apartado ya que todos los requisitos son desde una vista real variados dependiendo del nivel de vista de cada desarrollador. En conjunto hemos redactado lo que creemos que son los requisitos más prescindibles de este proyecto.

## Requisitos Funcionales

|  |  |
| --- | --- |
| **RF01** | Nuestro programa será capaz de Gestionar la entrada, modificación y salida de los datos del Alumnado del Centro, sobre las Distintas Categorías de las Distintas Pruebas de Evaluación Existentes. |
| **RF02** | El programa será capaz de Almacenar los Datos en la Base De Datos de nuestro Centro. |
| **RF03** | Al sistema podrá acceder el personal autorizado en el centro (Profesores, Tutores… etc.) |
| **RF04** | El sistema nos mostrará el listado del alumnado, categorías y pruebas requeridas en todo momento dependiendo de la necesidad del profesorado o responsable. |
| **RF05** | A cada alumno se le asignará un DNI el cuál será único para cada alumno. |
| **RF06** | En el menú de inicio del programa tendrá además de las opciones de gestión las opciones para importar y exportar los datos a nuestra base de datos. |
| **RF07** | El proceso de Gestión de Pruebas de Evaluación tendrá asociadas las calificaciones de los alumnos que hayan realizado dicha prueba. (Dichas pruebas de evaluación se encapsularán dentro de cada evaluación ejercida primera, segunda o tercera). |
| **RF08** | El proceso de Gestión de Alumnos contará con la información personal de cada alumno incluso si este alumno está de baja o no tanto como darle de baja. |

## Requisitos No Funcionales

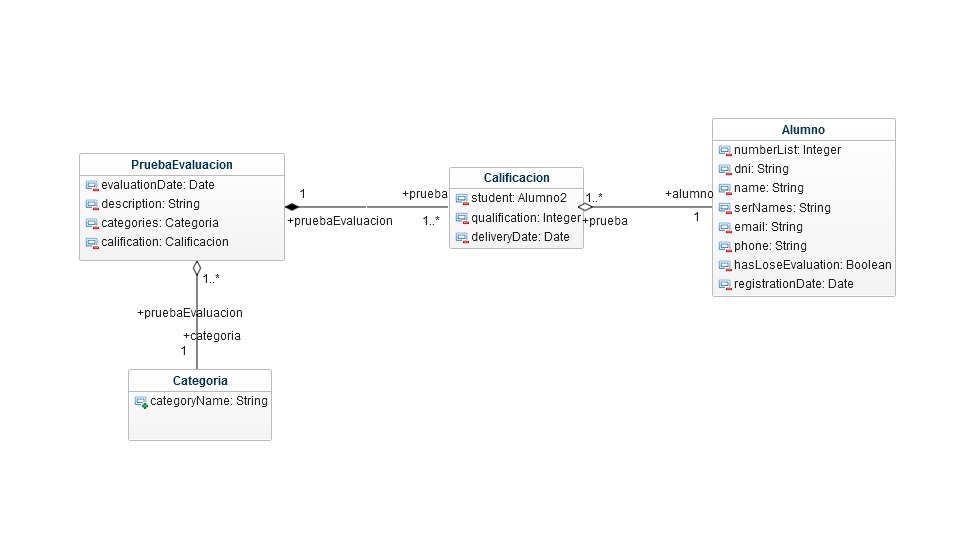
|  |  |
| --- | --- |
| **RF01** | El sistema debe de ser compatible con la mayoría de los equipos propuestos por el cliente. |
| **RF02** | El Formato de los datos referentes a importación y exportación de la información a nuestra base de datos se debe de hacer en JSON el cuál realizará la función de copia de seguridad (Backus). |
| **RF03** | El sistema debe de soportar el nivel de datos respectivo para cada centro al que será distribuido. |
| **RF04** | El sistema debe de ser legible para el profesorado y/o responsables del mismo. |
| **RF05** | El formato de los datos será en el Idioma Local de Implementación en este caso (Español). |
| **RF06** | El formato de entrada de datos tiene que ser aceptado en mayúscula, minúscula, con espacios, sin espacios y con la Capitalización de Títulos. |

## Requisitos De Información

|  |  |
| --- | --- |
| **RI1** | Alumno:   * DNI. * Su número de Lista. * Nombre. * Apellidos. * Correo Electrónico. * Teléfono. * Fecha de Matriculación. |
| **RI2** | Categorías:   * Nombre. |
| **RI3** | Pruebas de Evaluación:   * Fecha de Realización. * Descripción de la misma. * Categoría de la misma. * Lista de Calificaciones de dicha Prueba |
| **RI4** | Evaluación:   * Nota Máxima * Nota Mínima * Nota Media * Porcentaje de Aprobados y Suspensos |
| **RI5** | Prueba   * Alumno que realiza la prueba * Calificación * Fecha de Entrega de la misma |

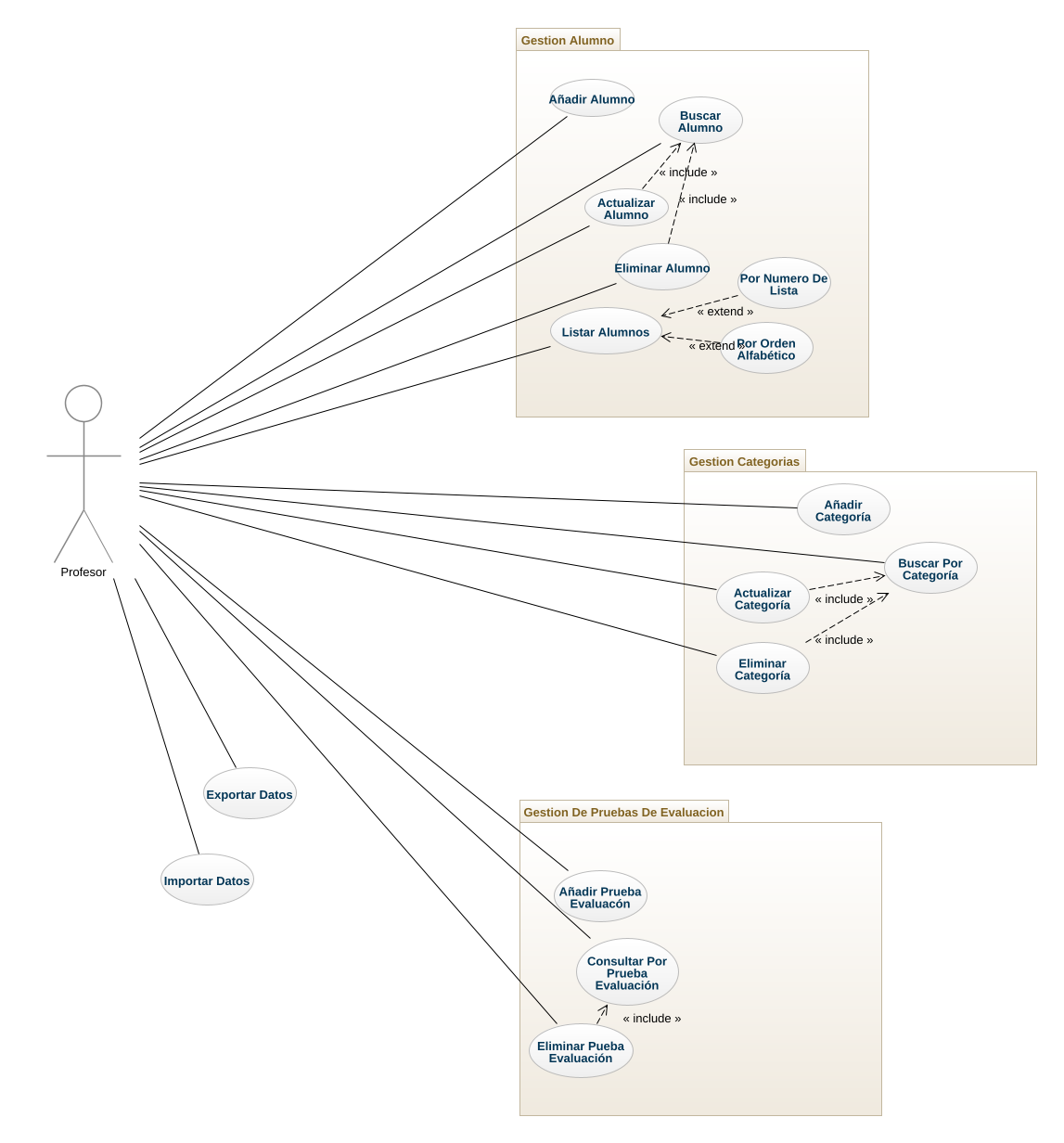
# Diagrama De Estructura

## Diagrama de clases.

Diagrama de clases implementado con la sintaxis UML correcta.

# Diagramas De Comportamiento

## Diagrama de caso de uso.

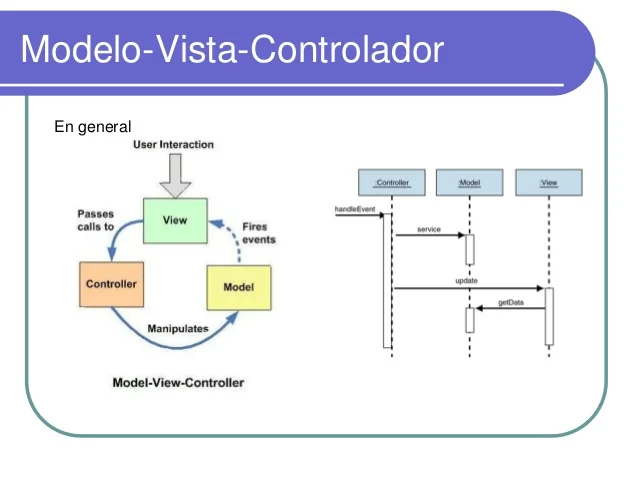
Diagrama de caso de uso implementado con la sintaxis UML correcta.

# Arquitectura del Software Aplicada

## Modelo Vista Controlador

En el paradigma MVC, las entradas del usuario, los modelos del mundo exterior y la retroalimentación visual son explícitamente separados y manejados por tres tipos de objetos, cada uno especializado para un conjunto de tareas específicas.

El objetivo primordial del patrón es dar soporte a los modelos funcionales y mapas mentales de la información relevante para los usuarios, permitiendo un modelo que facilite la consulta y manejo de los mismos. La única manera de construir artefactos manejables es ayudar al usuario a construir modelos del sistema. Pero esto es imposible si el modelo mental no ha sido diseñado dentro del artefacto desde el principio. Intentar adicionar los modelos mentales del usuario cuando ya se ha avanzado en el desarrollo puede ser imposible. A continuación, un gráfico que resume el patrón:

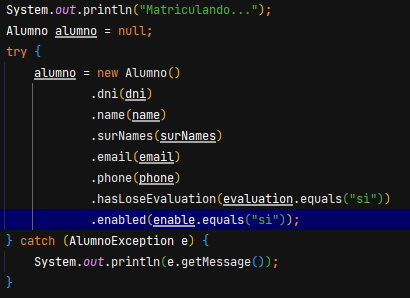


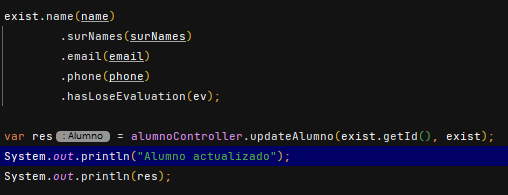
# Patrones de diseño y ejemplo de cómo se han aplicado.

## Patrón Builder

Es un patrón de diseño creacional que nos permite construir objetos complejos paso a paso. El patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.

En este caso hemos aplicado este patrón a la creación de Alumnos en nuestro programa teniendo como objetivo una gestión más organizada e profesional. De la misma forma que aplicamos este patrón sobre su modificación para que no haya discrepancias en nuestro código.



Creación de Alumnos a partir de BUILDER

Creación de Alumnos a partir de BUILDER 2

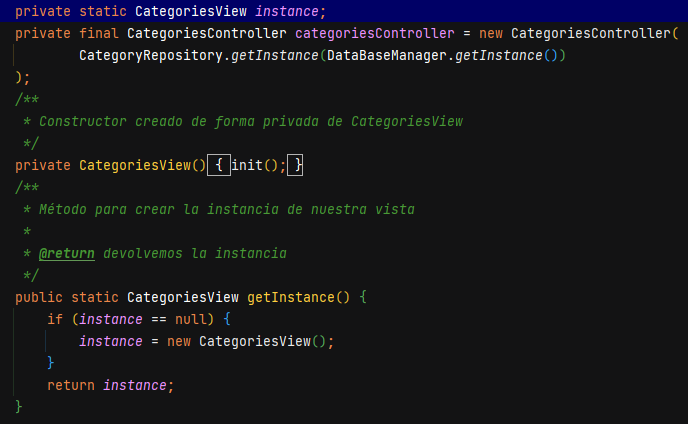
## Patrón Singleton

Es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

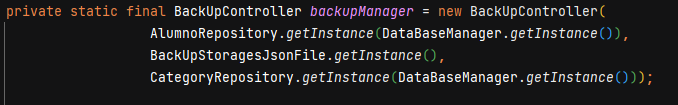
A su vez esta nos permite solventar varios problemas:

**Garantizar que una clase tenga una única instancia**. ¿Por qué querría alguien controlar cuántas instancias tiene una clase? Es decir, porque si nosotros queremos solamente tener una instancia de por ejemplo una vista la cual siempre actuará igual, para que tenemos que hacer a alguien que la controle simplemente al aplicarle este patrón en lugar de crear un objeto nuevo lo que vas a obtener es el anterior ya que es con el que quieres trabajar.

**Proporcionar un punto de acceso global a dicha instancia**. Gracias a este patrón podemos acceder a la instancia del objeto desde distintos puntos de nuestro programa sin necesidad de que estén entrelazados.



Implementación Singleton (Sobre un Obj)



Llamada del Singleton Instanciado

# Principios SOLID aplicadas y ejemplos.

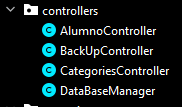
## S – Single Responsibility Principle (SRP)

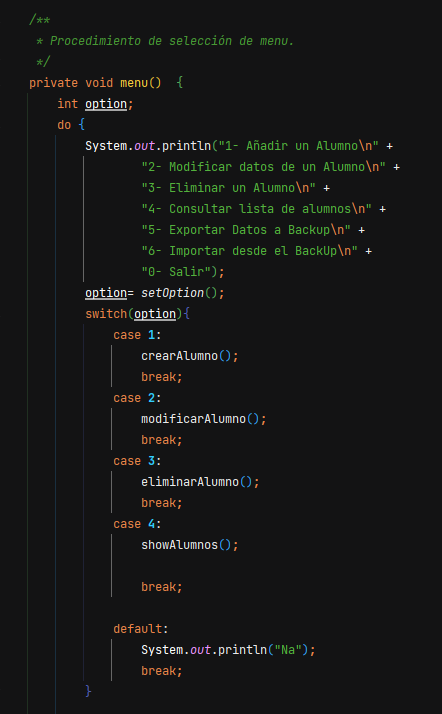
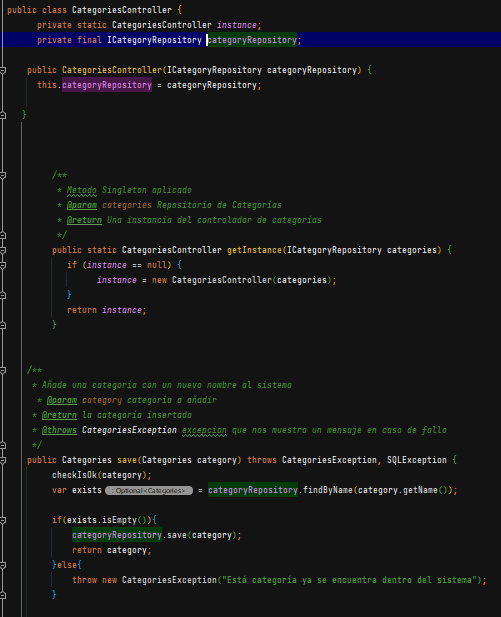
***“A class should have one, and only one, reason to change.”***

**Principio de Responsabilidad Única.**Este principio defiende el pensamiento de que una clase o método debe de tener solamente una misión sobre nuestro programa. No se cumpliría en el caso por ejemplo si nuestra vista también pudiese gestionar la creación del repositorio con sus métodos y la de Alumnos o Categorías por eso sería conveniente hacer un controlador, para cada clase.

Suele ser el principio fundamental para SOLID.

Ejemplos de Aplicación:

*En este caso lo que hemos hecho sobre nuestro programa ha sido  
una carpeta de Controladores donde se gestiona la creación de todo  
lo esencial de nuestro programa y una vista para que nos muestre los   
distintos métodos realizados sobre las partes de nuestro programa.*



## O – Open/Closed Principle (OCP)

***“You should be able to extend a classes behavior, without modifying it.”***

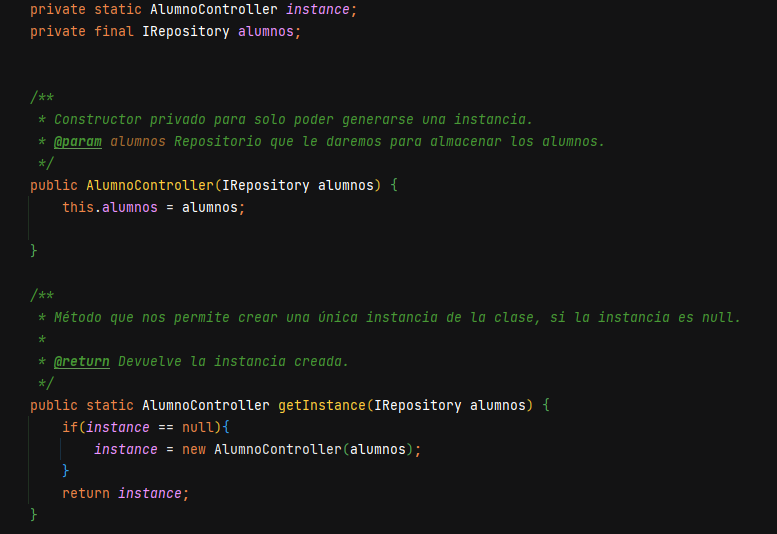
**Principio de Abierto/Cerrado.**Este principio defiende el pensamiento de que tenemos que ser capaces de extender el comportamiento de una clase sin tener que modificarlas también extendiéndolo al refrán ***las clases deben de estar cerradas a modificación y abiertas a extensión.***Por consecuencia aplicamos también el L – Liskov Substitution Principle (LSP)ya que tiene que cumplirse ambos para la perfecta cohesión de los mismos.

***Según*** [***Robert C. Martin***](https://es.wikipedia.org/wiki/Robert_C._Martin) ***incumplir el Liskov Substitution Principle (LSP) implica violar también el principio de Abierto/Cerrado.***

Deberíamos poder utilizar cualquiera de las subclases existentes sin tener que interferir en el funcionamiento de nuestro programa.

Ejemplos de Aplicación:

*En este caso lo que hemos hecho es que en vez de que   
el controlador tenga que entrarle la clase AlumnoRepository,  
le entrase la interfaz (IRepository) relacionada con esa frase así evitamos  
tanto el acoplamiento como el incumplir este principio.  
Con que en vez de que haga referencia al propio repositorio  
lo hace a la interfaz para que haya más elegibilidad.*



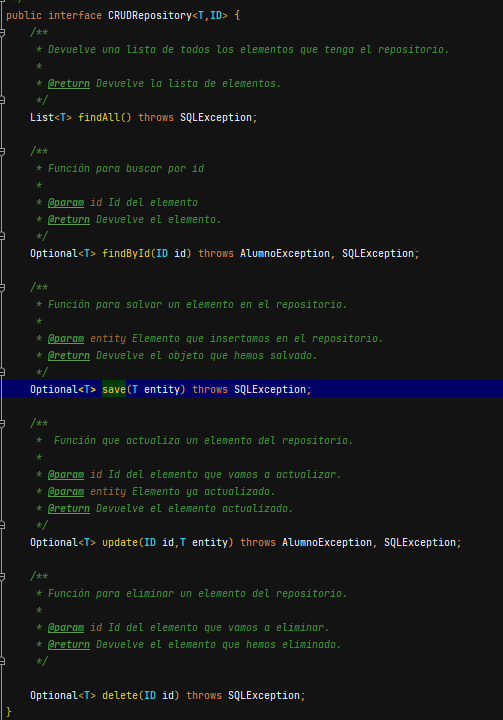
## I – Interface Segregation Principle (ISP)

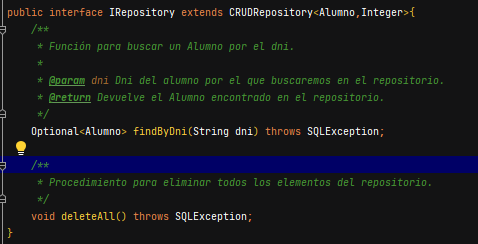
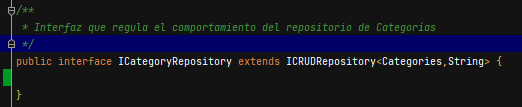
***“Make fine grained interfaces that are client specific.”***

**Principio de Segregación de interfaz.**Este principio defiende el pensamiento de que hemos de tener como preferencia en nuestro programa muchas interfaces las cuales implementen métodos específicos para cada momento que contar con una Mega-Interfaz la cual disponga de muchos métodos para clases distintas, los cuales no se le dará uso ya que dependiendo de la clase que la implemente tendrá que implementar todos aun así no los vaya a utilizar.

Ejemplos de Aplicación:

*En este caso lo que hemos hecho ha sido creado una interfaz CRUD  
la cual dispone de distintos métodos ya expuestos en anteriores  
ocasiones para más información haga clic* [*aquí*](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html)*. De la cual van a   
salir varias variantes en este caso 2 las cuales son para Alumnos y  
Categorías. Como observamos cada una dispone de métodos específicos para   
cada clase los cuales solo se usarán en esas clases.*

**

**

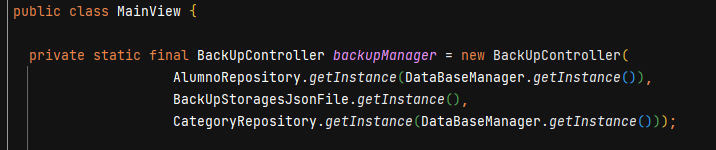
## D – Dependency Inversion Principle (DIP)

***“Depend on abstractions, not on concretions.”***

**Principio de Inversión de Dependencias.**Este principio defiende el pensamiento de que hagamos que dependamos de abstracciones y no de las clases concretas. Algunas ideas que tenemos que tener en claro son: No deben los módulos de alto nivel depender de módulos de bajo nivel, a su vez ambos deberían de depender de las abstracciones.

Este principio se basa en reducir las dependencias entre los módulos de nuestro código para evitar el acoplamiento entre clases.

Ejemplo de Aplicación:

*En este caso lo que hemos hecho como ya hemos visto anteriormente  
hacer que los controladores y las vistas tengan las dependencias de lo que necesitan y no las propias clases creadas desde 0.*

Aplicación de Inversión de Dependencias

## Resumen

Hemos tratado de implementar de forma íntegra los principios de la filosofía SOLID expuesta por [***Robert C. Martin***](https://es.wikipedia.org/wiki/Robert_C._Martin)ya que así nuestro código se creará de forma profesional y nos evitará incongruencias al modificarlo y actualizarlo en el futuro ya que seguimos un orden en nuestro programa preestablecido lo que dará lugar a un código organizado y eficaz.

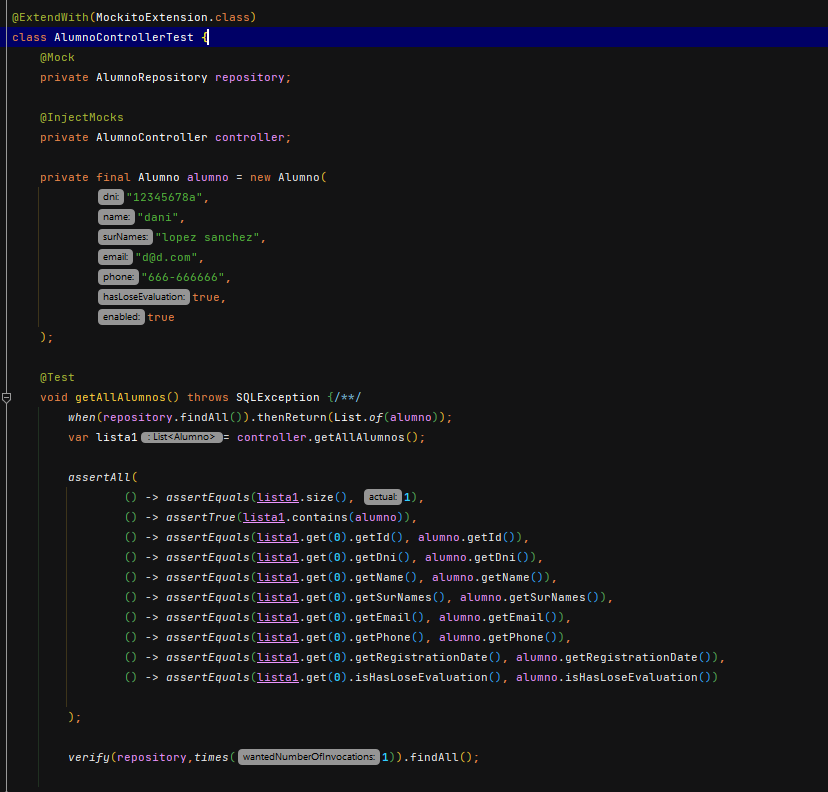
# Diseño de Pruebas unitarias.

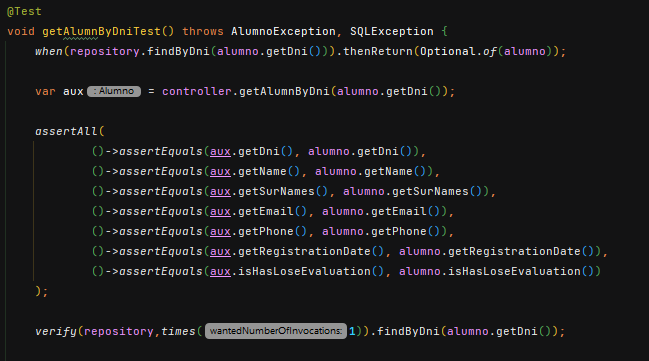
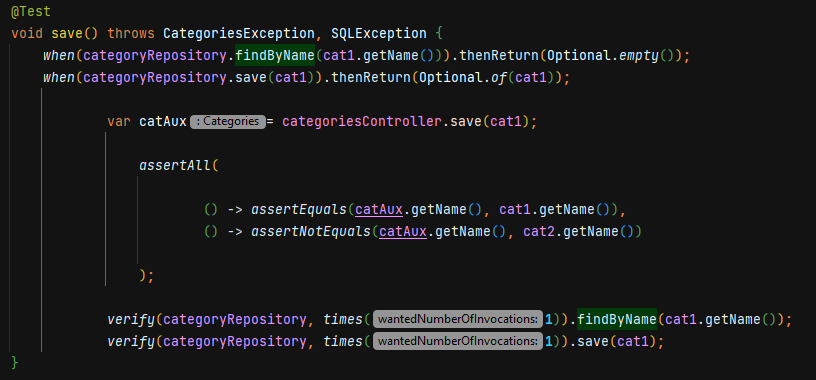
## Tests

Hemos implementado los distintos test respectivos a las necesidades sobre nuestro programa. Sobretodo hemos implementado test con [Mockito](https://site.mockito.org) el cual se encarga de simular en este caso el comportamiento de nuestro repositorio y realizar los test de una forma más eficaz. Aspectos: con el when simulamos el comportamiento y el verify lo que nos proporciona es ver si se cumple el número de veces que el repositorio hará la acción.

Aquí algunos ejemplos sobre la implementación de las pruebas unitarias implementadas:

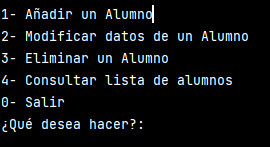
*Aquí hemos inyectado el Mock como se puede observar-.*



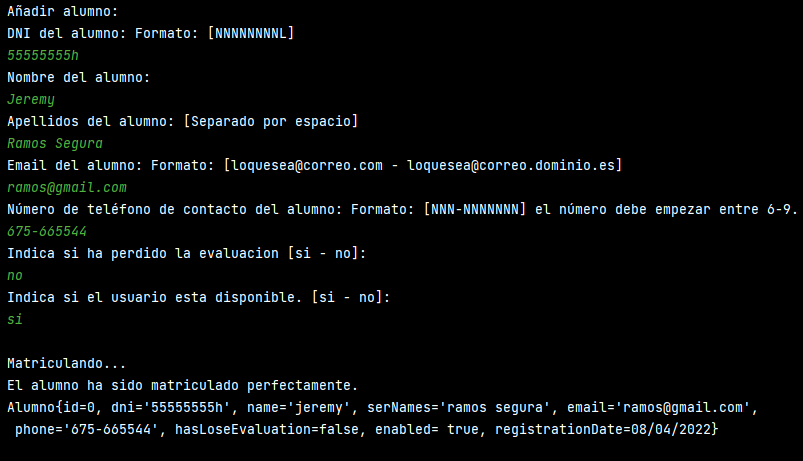


# Ejemplo de uso y capturas del mismo.

### Gestión de Alumnos



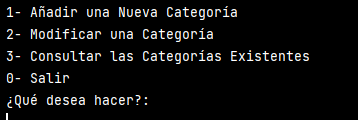
### Creación de Alumnos



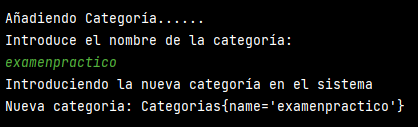
### Muestreo de Alumnos



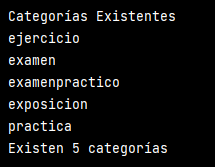
### Gestión de Categorías



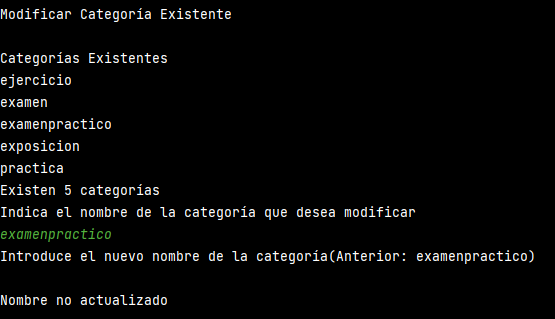
### Creación de Categorías



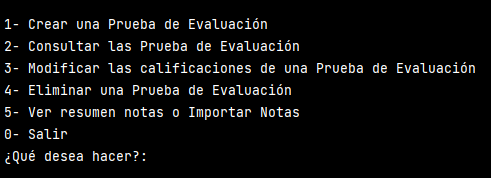
### Muestreo de Categorías



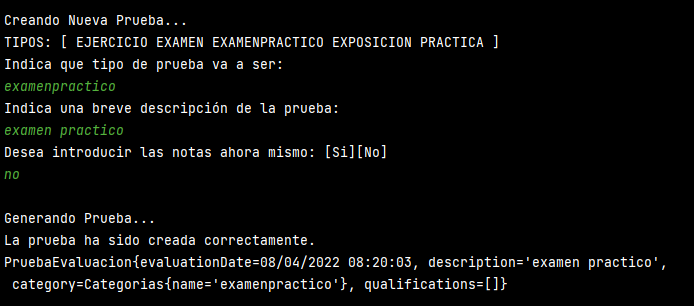
### Modificación de Categorías



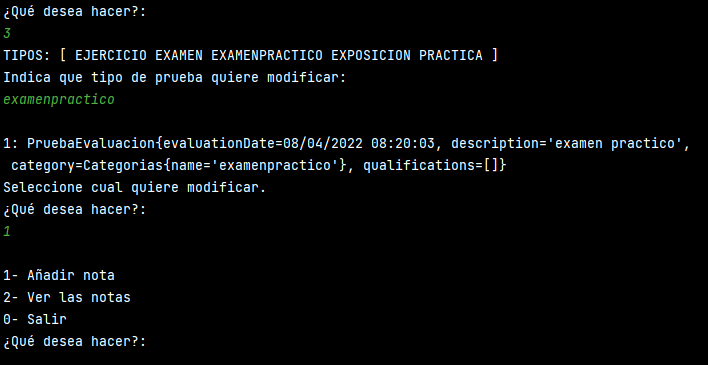
### Gestión de Pruebas de Evaluación



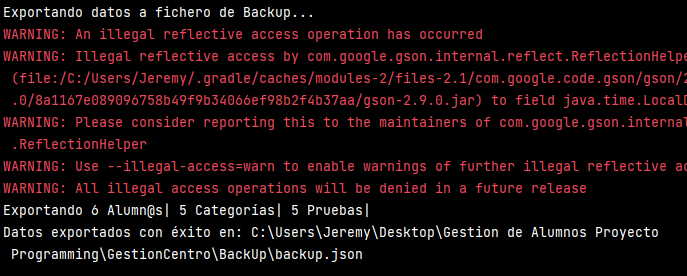
### Creación de Prueba de Evaluación



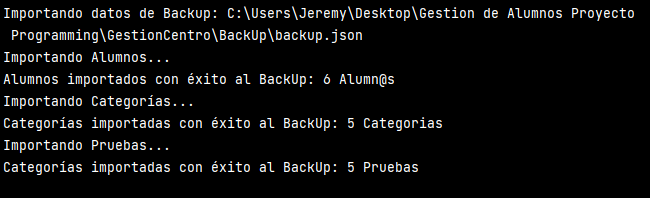
### Modificación de las Pruebas de Evaluación



### Exportación a Json



### Importación de Json



### Json

