**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

**DISEÑO DE SOFTWARE**

**TAREA DE PRINCIPIOS SOLID**

**Objetivos Específicos**

1. Diseñar un producto de software aplicando principios de diseño orientado a objetos para que sea robusto, mantenible y escalable.
2. Emplear herramientas informáticas en el control de versiones para la generación de software de calidad en un entorno colaborativo.

**Resultado de Aprendizaje**

1. Funcionar efectivamente como miembro o líder de un equipo involucrado en actividades apropiadas para la disciplina del programa.
2. Habilidad para aplicar teoría de ciencias computacionales y fundamentos de desarrollo de software para producir soluciones basadas en computación.

**Descripción**

En equipos de trabajo, conformados por entre tres a cinco estudiantes, se debe (i) seleccionar no más de 3 repositorios públicos en GitHub con código fuente en Java [1] a fin de (ii) rediseñar partes del código fuente en donde sean identificables 3 violaciones (en total) a cada principio de diseño SOLID.

**Especificaciones**

**Sección A**

Seleccione no más de 3 repositorios públicos en GitHub, creados antes de la fecha de publicación de esta tarea, en los que pueda ilustrar violaciones a principios SOLID (todos los principios deben ser ilustrados al menos 3 veces). Cada repositorio solo puede ser analizado por un equipo (incluidos semestres anteriores). Por lo tanto, una vez seleccionado debe reservarlo. Para reservar un repositorio, un integrante del equipo debe indicar su selección en el foro AulaVirtual titulado “Reserva de repositorios Tarea SOLID”. Solo puede haber un máximo de tres participaciones por equipo en el foro. Antes de intentar reservar un repositorio, asegúrese que no haya sido reservado previamente. **[10%]**

**Sección B**

En los repositorios seleccionados, identifique 3 violaciones a cada uno de los principios SOLID. Justifique su argumentación. Además del texto, puede hacer uso de diagramas UML. **[45%]**

**Sección C**

Proponga una solución que aplique correctamente cada principio SOLID que se haya violado. Su solución debe ser reportada como incidente en GitHub y debe contener el código fuente propuesto y diagramas UML. **[45%]**

**Entregables (en un archivo ZIP o RAR)**

1. Un **documento** (en formato docx) que identifique los integrantes del equipo en la primera página y tenga un índice de contenido en la segunda. El documento debe identificar los repositorios seleccionados, contener su justificación que argumente cada violación de los principios SOLID. Puede hacer uso de cualquier asunción razonable. Se debe incluir capturas de pantalla del código fuente inicial y su estado posterior luego del rediseño. Utilice diagramas UML para ilustrar su solución. Finalmente, el documento debe contener alguna evidencia del reporte de su sugerencia al propietario del repositorio pertinente.

**Rúbrica de Calificación**

|  |  |
| --- | --- |
| **Descripción** | **Valor** |
| **Sección A** | |
| Selección de repositorio(s) único(s) | 10 |
| **Sección A** | |
|  | |
| Identificación y justificación de violaciones a principio SRP | 09 |
| Identificación y justificación de violaciones a principio OCP | 09 |
| Identificación y justificación de violaciones a principio LSP | 09 |
| Identificación y justificación de violaciones a principio ISP | 09 |
| Identificación y justificación de violaciones a principio DIP | 09 |
| **Sección B** | |
| Rediseño aplicando principio SRP, incluyendo diagramas UML y reporte con solución al propietario del repositorio | 09 |
| Rediseño aplicando principio OCP incluyendo diagramas UML y reporte con solución al propietario del repositorio | 09 |
| Rediseño aplicando principio LSP incluyendo diagramas UML y reporte con solución al propietario del repositorio | 09 |
| Rediseño aplicando principio ISP incluyendo diagramas UML y reporte con solución al propietario del repositorio | 09 |
| Rediseño aplicando principio DIP incluyendo diagramas UML y reporte con solución al propietario del repositorio | 09 |
| **Total** | **100** |
| No subir a SidWeb los entregables requeridos de acuerdo con lo especificado | -50 |
| Penalidad por día o fracción de día de retraso en la entrega | -30 |

**Referencias**

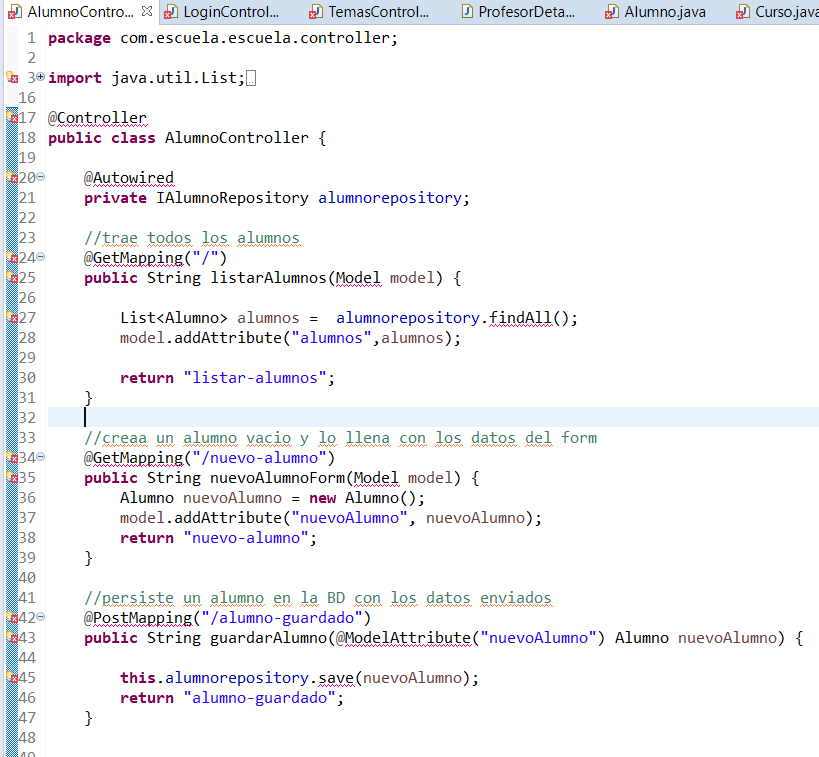
1. <https://github.com/topics/java>

**Principios Solid**

**Steven Choez**

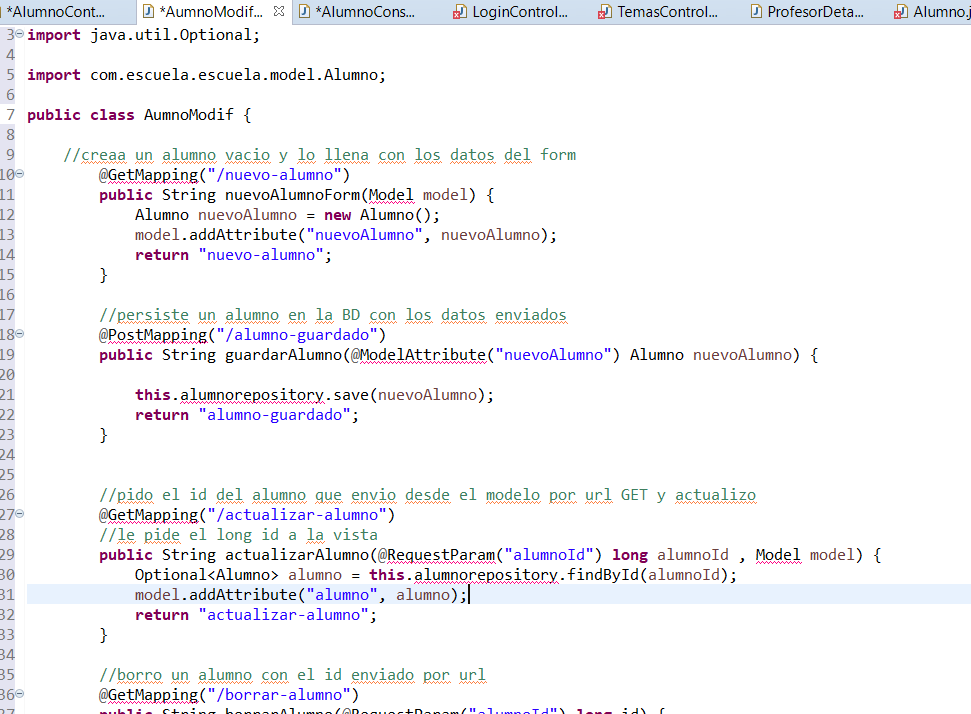
**Repositorio:** [**https://github.com/Valkyries12/escuelita-capgemini/tree/master**](https://github.com/Valkyries12/escuelita-capgemini/tree/master)

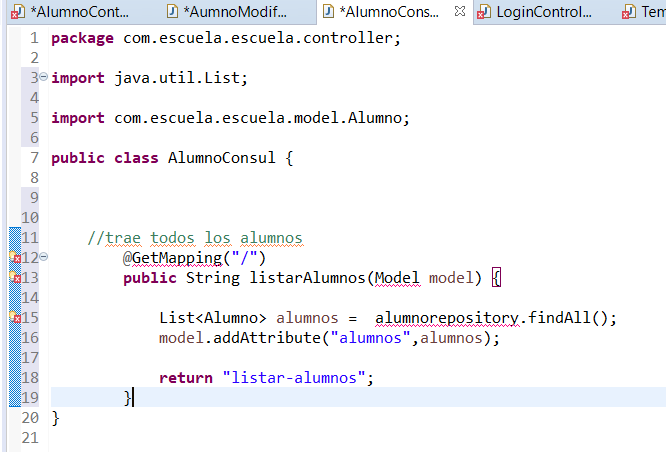
**Principio de responsabilidad única**



AlumnoController realiza procesos de modificación y de consulta en la misma clase por lo que se nota cargada, se pueden separar los métodos en una clase de Modificación llamada AlumnoModif y una de consulta llamada AlumnoConsul de manera que se les da una responsabilidad única.

**Solución Java**

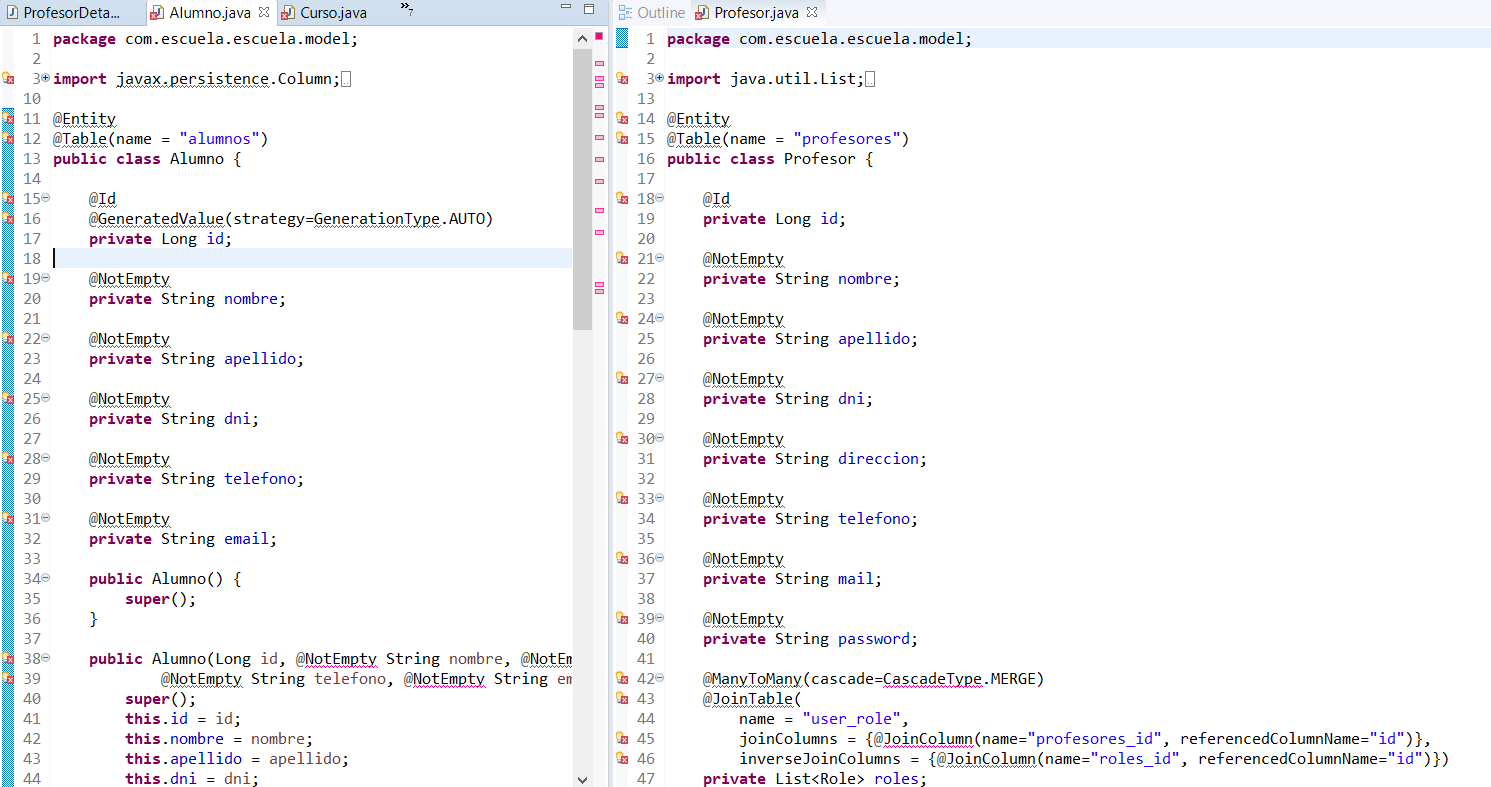
c



**Solución Uml**

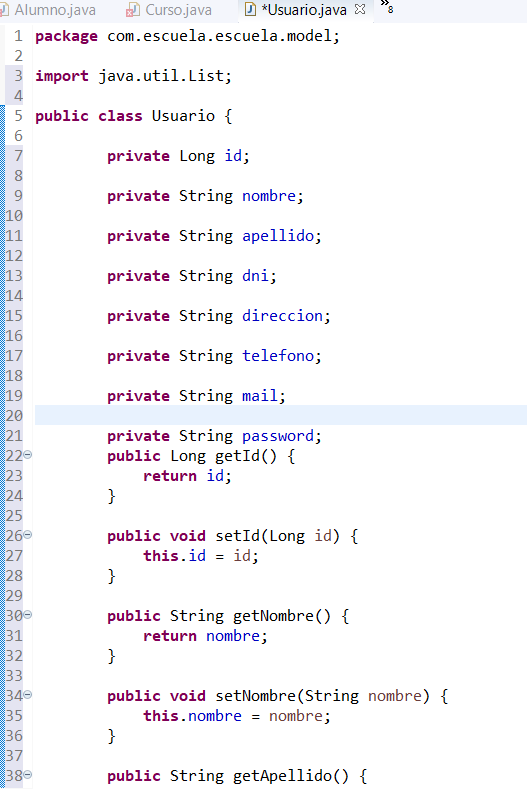


**Principio abierto cerrado**

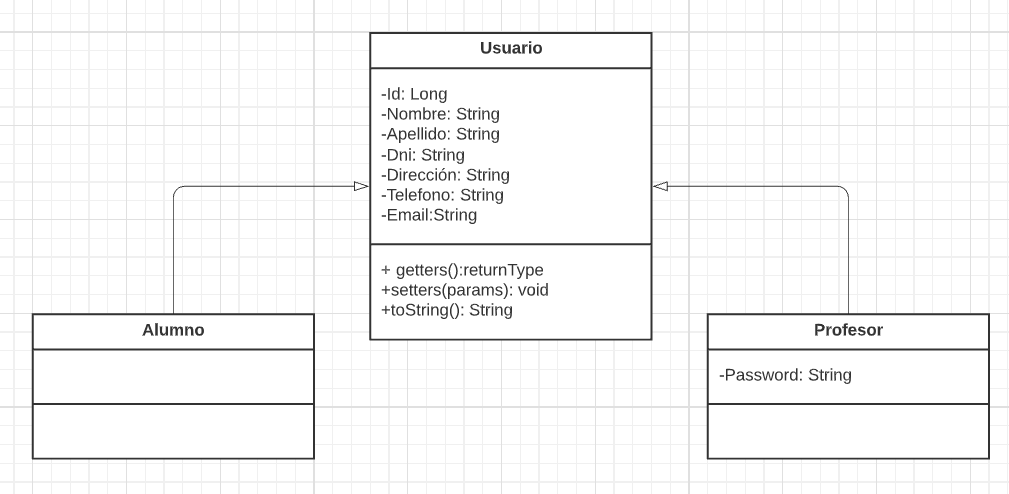


Las clases Alumno y Profesor muestran los mismos atributos y métodos por lo que sería ideal crear una clase padre llamada Usuario de la cual estas dos hereden haciéndolo abierto a la extensión y cerrado a la modificación.

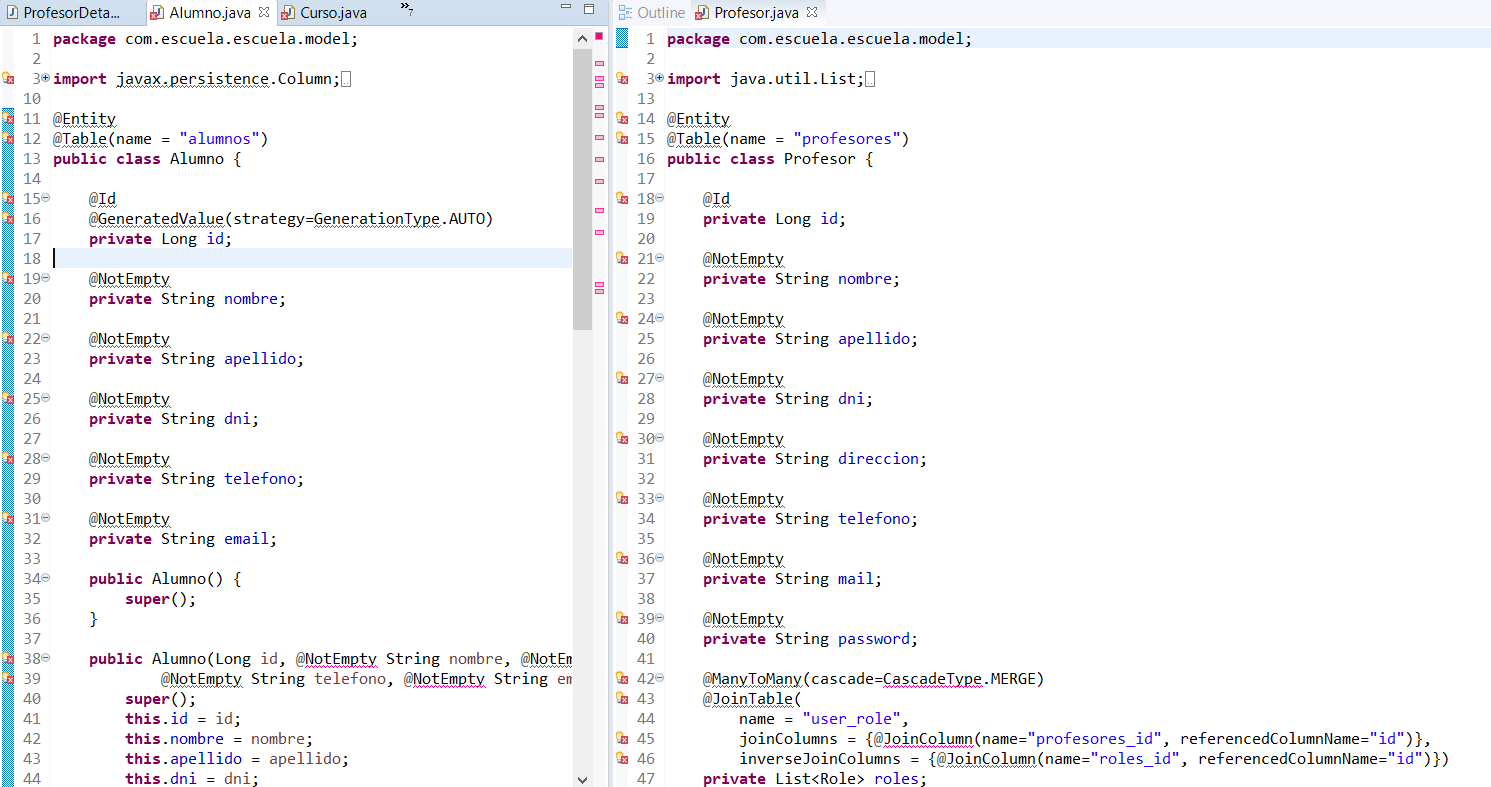
**Java**



**Uml**

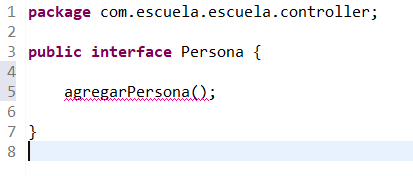


**Principio de substitución de Liskov**

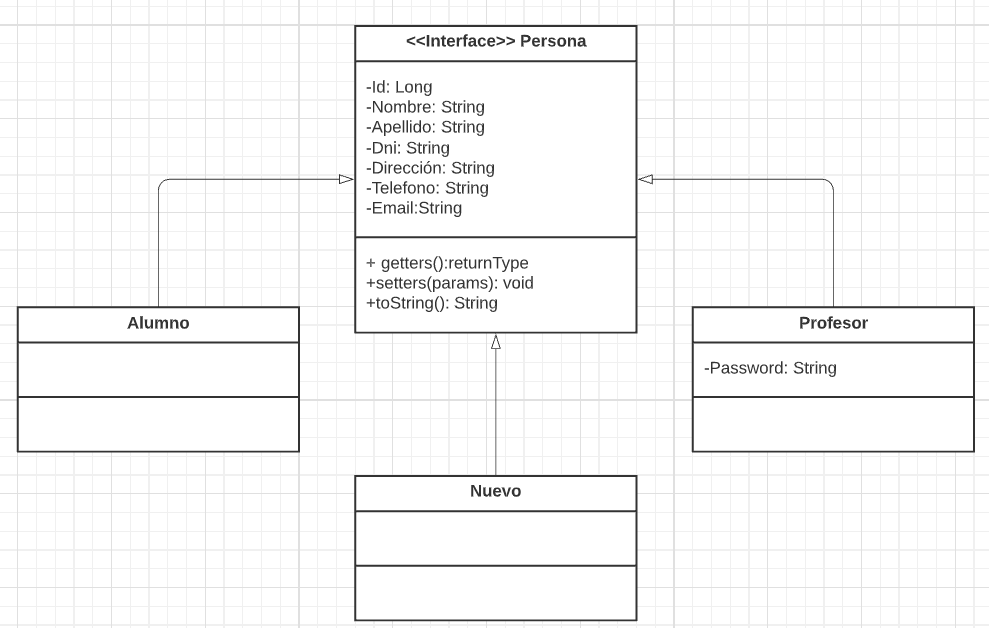


En el caso de que se quiera poner a otra persona en el sistema que no necesariamente se relaciona con el un estudiante o profesor sino por ejemplo un director, este no debería confundirse con la clase Usuario ya que no es parte del rol de estudio, pero sí debería poder acceder a sus atributos y métodos por lo que con una interfaz de Persona puede hacerlo.

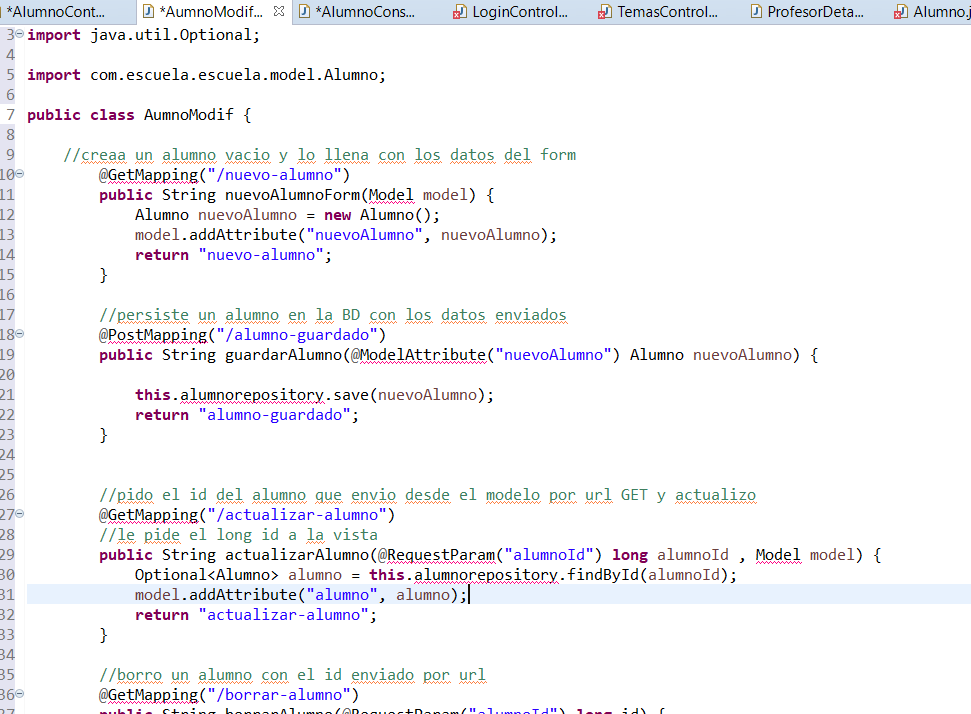
**Java**

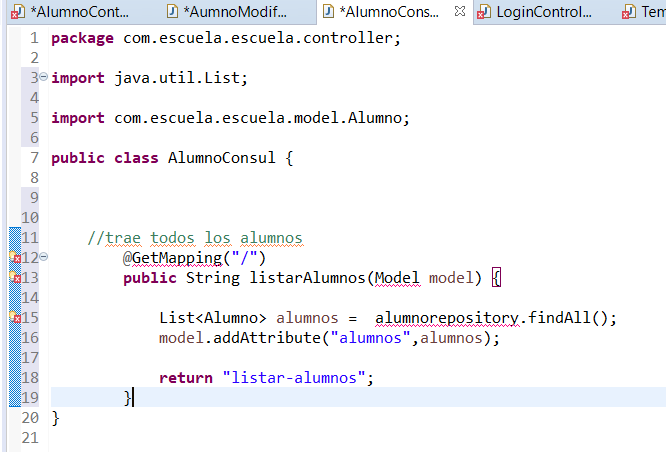


**Uml**



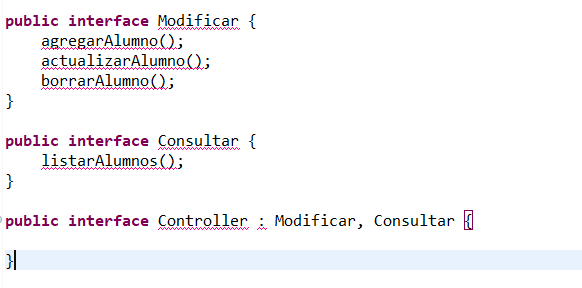
**Principio de segregación de interfaces**



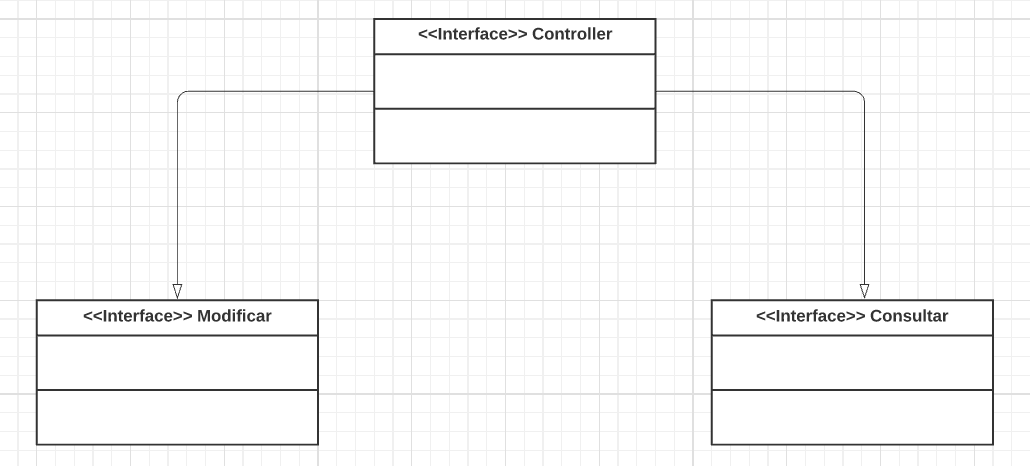


Este principio dice que no debemos dar más información de la necesaria a cada módulo para funcionar, y en el caso de la clase AlumnoController se puede decir que se conoce tanto los métodos de modificación como de consulta por lo que para resolver este problema podemos crear interfaces para cada una y una interfaz que Controller que se encargue de llamar a la otra interfaz de manera que todo está más segregado.

**Java**



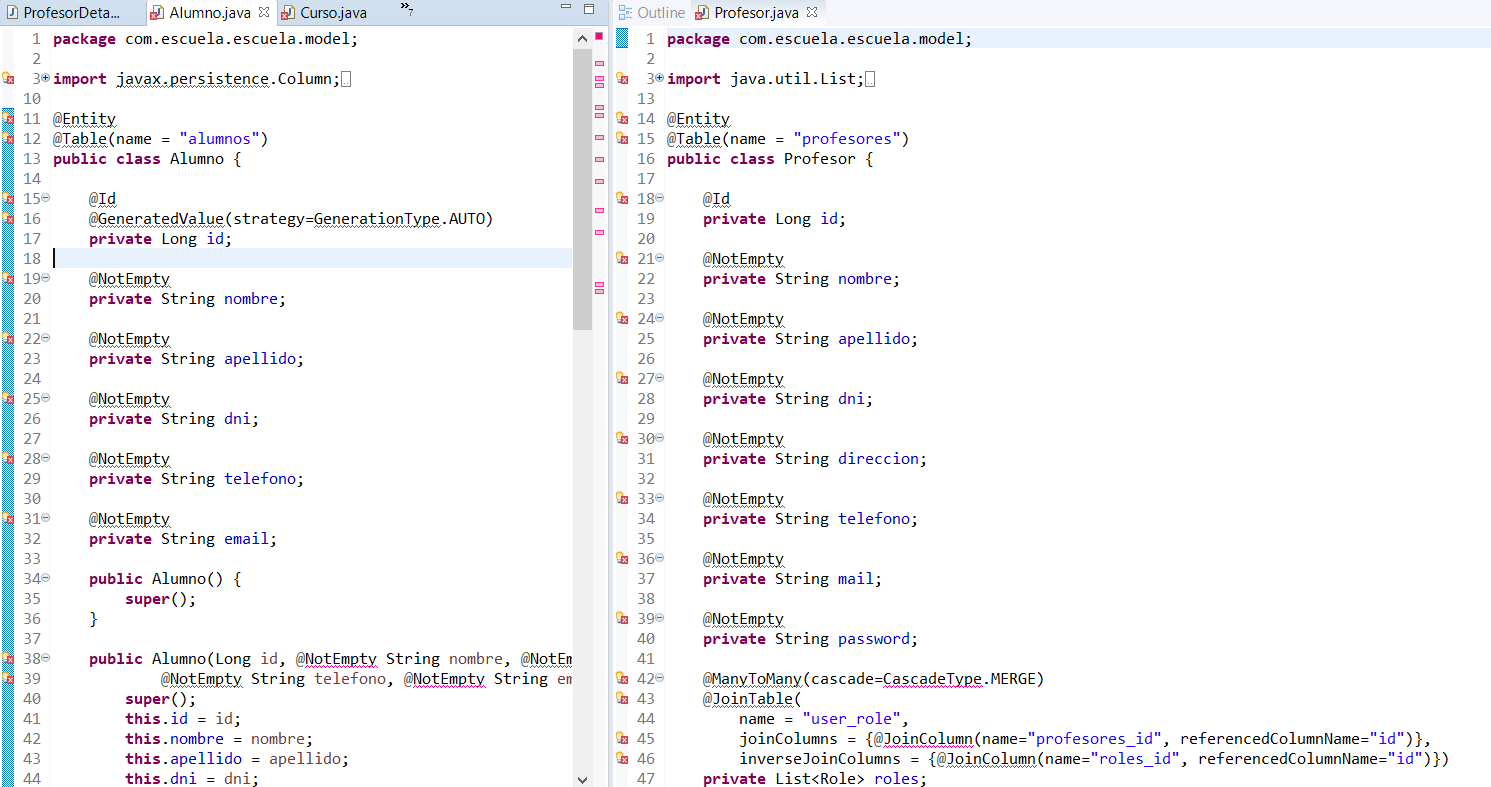
**Uml**



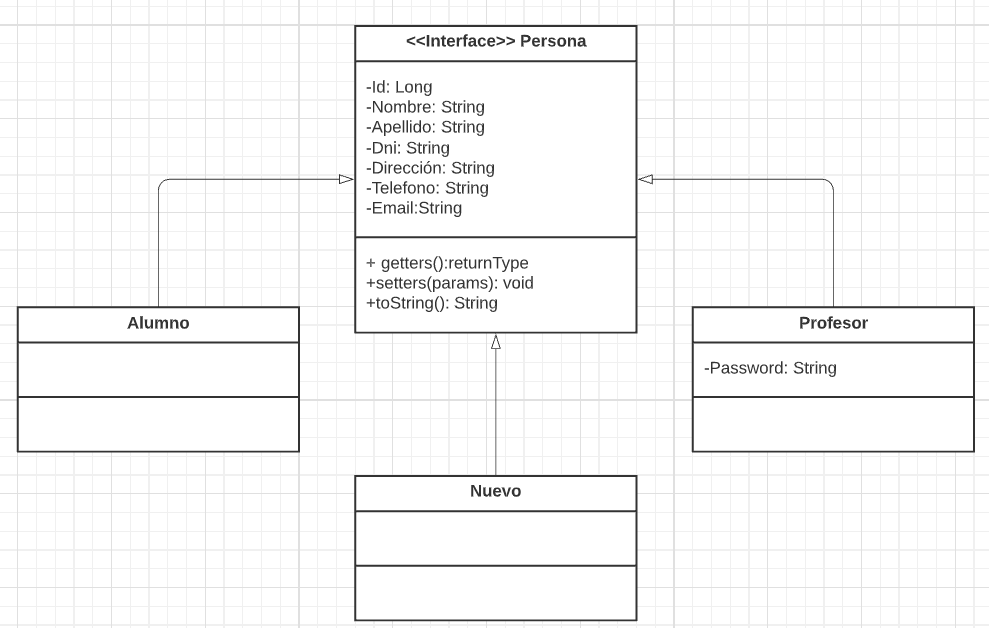
**Principio de inversión de dependencia**

Los módulos de alto nivel no deberían depender de los módulos de bajo nivel, por lo que es necesario que los componentes se comuniquen entre interfaces de manera que siempre sea posible modificar o crear un nuevo usuario por ejemplo sin que sea necesario modificar las clases.

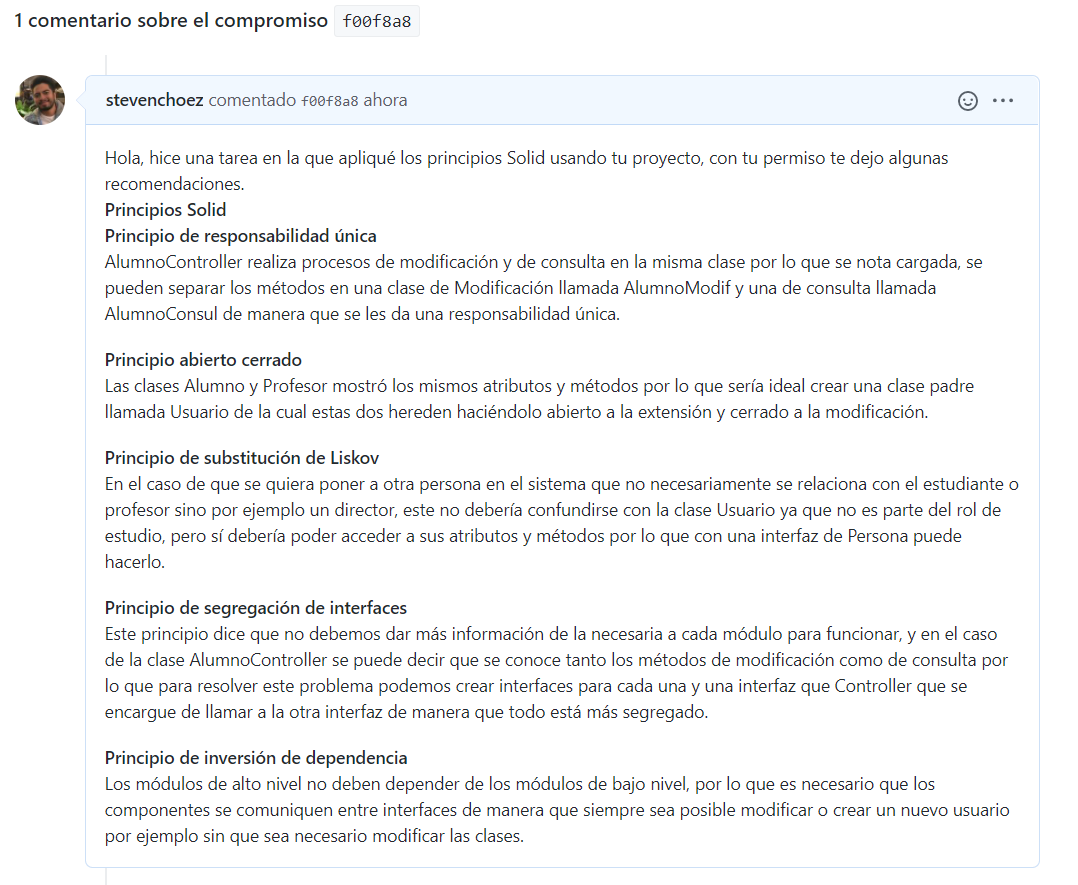
**Java**



**Uml**



**Evidencia de recomendaciones en el repositorio**



**Kevin Castro**

**Repositorio:** [**https://github.com/angelomedina/Sistema-de-venta-de-vehiculos**](https://github.com/angelomedina/Sistema-de-venta-de-vehiculos)

**Principio de responsabilidad única**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Aparte de usar métodos que manejen la info de la clase como lo son los getters y setters, utiliza métodos de verificar tarjetas, etc. Por lo que se necesita crear una clase que maneje los demás métodos de Buyer.

**Solución Java**

Graphical user interface, text, application, email

Description automatically generated

**Solución UML**

Diagram

Description automatically generated

**Principio abierto cerrado**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Las clases wishes y purchases poseen atributos en común, por lo que debería existir una nueva clase llamada Venta que extienda a estas dos, y sea cerrada para implementación y abierta para su extensión.

**Solución java**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated



**Solución UML**

Diagram

Description automatically generated

**Principio de substitución de Liskov**

Text

Description automatically generated

Text

Description automatically generated

Al querer implementar un método que verifique el dinero que posee el usuario, tendríamos que cambiar dos clases en especifico (Debts, creditCard) aparte si se necesita añadir otro método de pago como Paypal se tendría que modificar también el código para hacer la verificación del dinero.

**Solución Java**

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated

Text

Description automatically generated

**Solución UML**

Diagram

Description automatically generated

**Principio de segregación de interfaces**

Graphical user interface, text, application

Description automatically generated

La interfaz publicInformation, imprime tanto los vehículos nuevos como los viejos, se debería poder separar estas funciones con el fin de que el codigo sea mas entendible.

**Solución Java**

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

**Solución UML**

Diagram

Description automatically generated

**Principio de inversión de dependencia**

Text

Description automatically generated with low confidence

Debe haber más relación entre la clase Vehicule y las clases usedVehicules y newVehicules, haciendo notar que los métodos de bajo nivel deben depender de los de alto nivel.

**Solución Java**

Text

Description automatically generated

**Solución UML**

Diagram

Description automatically generated

Graphical user interface, text, application, Teams

Description automatically generated

**Pratt García**

**REPOSITORIO:** <https://github.com/wzalazar/Cine/tree/master>

**SINGLE RESPONSABILITY PRINCIPLE**

A picture containing text, screenshot, indoor

Description automatically generated

En la clase DatosEspectadores que esta encargada de manejar todo lo que tenga que ver con los datos de un Espectador, se encuentra un método ComprobarPassword, método que esta mas ambientado a la seguridad del sistema y que bien podría ser ubicado en una clase que solamente se encargue de verificar si el Espectador se encuentra en el sistema .

**RESOLVIENDO VIOLACION DE SRP**

A picture containing text, screenshot, indoor

Description automatically generated

Para solucionar esto se puede crear una clase aparte con un método estatico cuya única función sea la de la verificación del usuario en el sistema

Diagram

Description automatically generated

**OPEN-CLOSED PRINCIPLE**Graphical user interface, text, application

Description automatically generated

En la clase “Espectador” hay un atributo llamado “Tipo” el cual asigna al Espectador un tipo de atención y se lo puede setear haciendo uso de su respectivo método set si seguimos la lógica del programa, sin embargo esto puede ocasionar problemas de OCP si en un futuro se requiere crear una clase especial de Espectador ya que si a este se le quieren agregar atributos especiales o métodos que trabajen solo para este tipo de Espectador, esto requerirá un cambio de código en la clase Espectador ya que habría que alterar el atributo “Tipo” por algo más complejo.

**RESOLVIENDO VIOLACION DE OCP**

Para resolverlo se crea una interfaz “TipoEspectador” de tal manera que cuando se quiera agregar alguna funcionalidad extra a algún tipo se Espectador en concreto o se requiera crear un nuevo tipo de Espectador solo se deba agregar código al programa mas no modificarlo.

Diagram

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application

Description automatically generated

**INTERFACE SEGREGATION PRINCIPLE**

Graphical user interface, text, application, chat or text message

Description automatically generated

EspectadorVIP no cuenta con el atributo DNI pero debe implementar métodos que hacen uso de ese atributo, esto es una violación al principio de ICP debido a que esos métodos están obligados a utilizarce en una clase que no tiene uso para ellos

**RESOLVIENDO VIOLACION ICP**

Graphical user interface, text, application

Description automatically generated

Se crea una interfaz con los métodos puntuales que la clase EspectadorVIP utilizara ya que si se realiza esta acción, la clase no se vera en la obligación de tener que implementar métodos que no usa

**LISKOV SUBSTITUTION PRINCIPLE**

Graphical user interface, text, application, chat or text message

Description automatically generated

En la clase EspectadorVIP hay un método que valida si un espectador es del tipo VIP examinando su Id y revisando que este no contenga un 9, a diferencia de las otras subclases que implementan TipoEspectador esta es la única que tiene una restricción y que no puede reemplazar a su clase base

**RESOLVIENDO VIOLACION LSP**

Graphical user interface, text, application

Description automatically generated

Diagram

Description automatically generatedNos aseguramos de que si usamos este método significara que el Espectador es VIP

**DEPENDENCY INVERSION PRINCIPLE**

A picture containing text, screenshot, indoor

Description automatically generated

Graphical user interface, text, application

Description automatically generated

La clase Funcion incumple el principio DIP debido a que depende de 3 clases concretas, en este caso son los objetos de tipo Sala,Pelicula,Horario. Se sabe que los modulos de alto nivel como lo es la clase “Funcion” no deberían depender de modulos de bajo nivel como los listados anteriormente.

**RESOLVIENDO VIOLACION DE DIP**

Para corregir este problema debemos sacar provecho de la utilización de interfaces

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Diagram

Description automatically generatedGraphical user interface, text, application

Description automatically generated

De esta manera la clase que maneja las Funciones ya no dependería de clases concretas sino de abstracciones que es lo que estipula el principio de DIP

Prueba de sugerencia al owner del repositorio

Graphical user interface, text, email

Description automatically generated