Juan Guadalupe, Roberto Patiño, Derek Avilés, Jaime Hurtado, Jose Zambrano

Integrantes

Principios SOLID

Grupo 6

Contenido

[Contenido 1](#_Toc105703432)

[1.- Principios SOLID Sección A 1](#_Toc105703433)

[1.1.- Single Responsibility Principle (SRP) 1](#_Toc105703434)

[1.2.- Open-Closed Principle (OCP) 1](#_Toc105703435)

[1.3.- Liskov Substitution Principle (LSP) 2](#_Toc105703436)

[1.4.- Interface Segregation Principle (ISP) 3](#_Toc105703437)

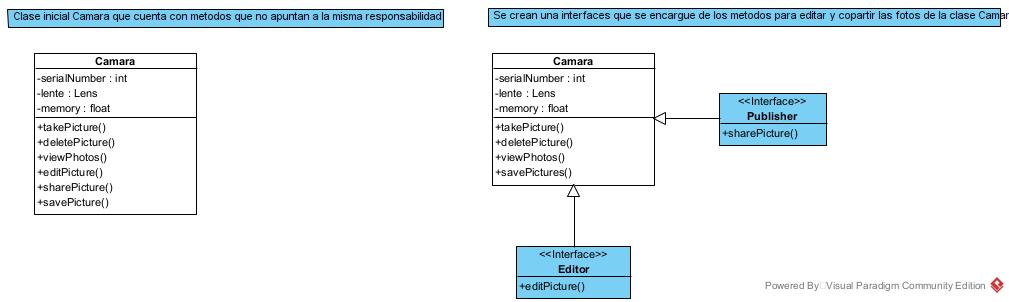
[1.5 Dependency Inversion Principle (DIP) 4](#_Toc105703438)

[2.- Caso de estudio código fuente Sección B 5](#_Toc105703439)

# 1.- Principios SOLID Sección A

## 1.1.- Single Responsibility Principle (SRP)

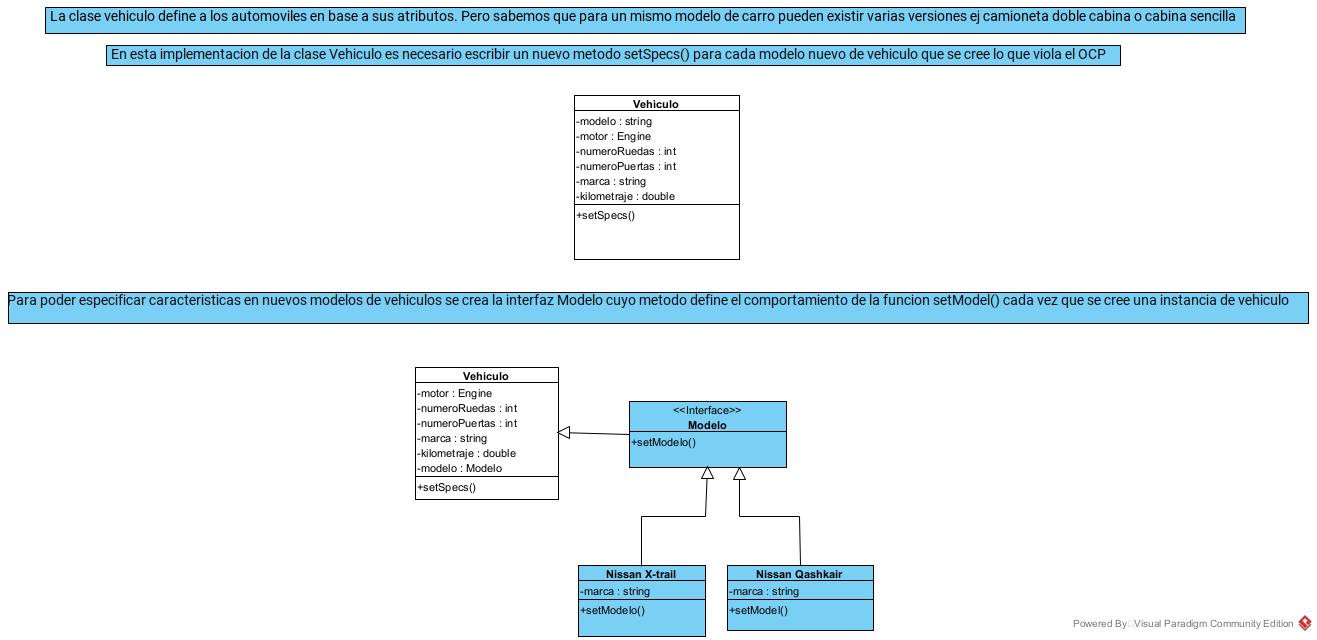
El principio de Single Responsibility establece que las clases deben tener una sola razón para modificar su estado o en otras palabras cumplir un solo trabajo.



En este ejemplo se describe la clase Camara que en primera instancia se encarga de contener la información fundamental de la cámara y métodos involucrados en el proceso de la toma de fotografías, edición y publicación de fotos. En primera instancia podemos decir que no se está cumpliendo el SRP ya que una cámara debería preocuparse únicamente por tomar fotos y guardarlas en su memoria. Es por esto que se crean las interfaces Editor y Publisher que definen los métodos para editar y publicar las imágenes y se procede a hacer que la clase Camara implemente las interfaces.

## 1.2.- Open-Closed Principle (OCP)

La definición del Open-Closed Principle establece que las clases creadas deben ser aptas para la extension más no para su modificación. En este caso la clase vehículo no lo cumple en primera instancia.

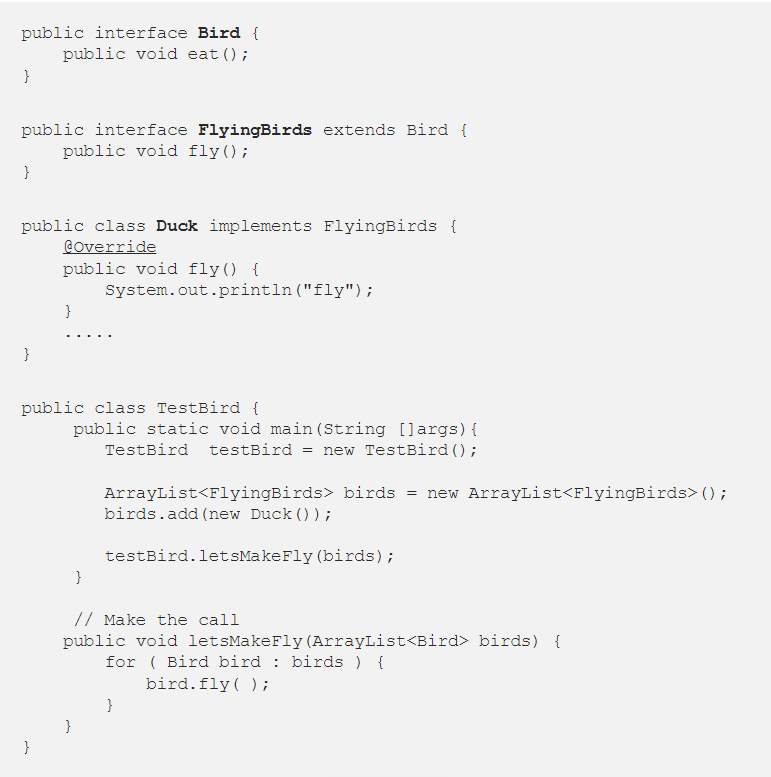


Ésta es una posible solución a este principio ya que si necesitamos más modelos de vehículos se define las características de ese modelo en específico al momento de su creación sin necesidad de alterar el método de setSpecs() dentro de la clase Vehiculo.

## 1.3.- Liskov Substitution Principle (LSP)

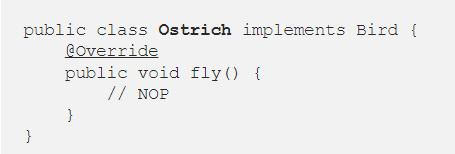
Presentado por Barbara Liskov, el principio de Liskov Substitution nos indica que cualquier clase representada como un subtipo puede ser reemplazada por la clase original de la cual se deriva, sin causar problemas o comportamientos inesperados en el programa.

Puesto de otra manera, el Liskov Substitution Principle es violado cuando una clase derivada de una clase base puede presentar problemas o excepciones que la clase base no.



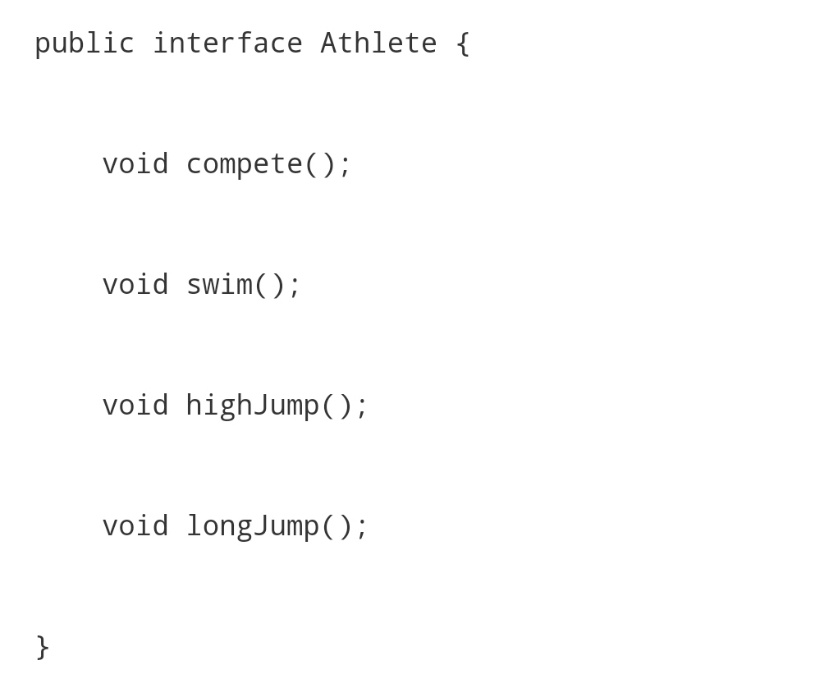
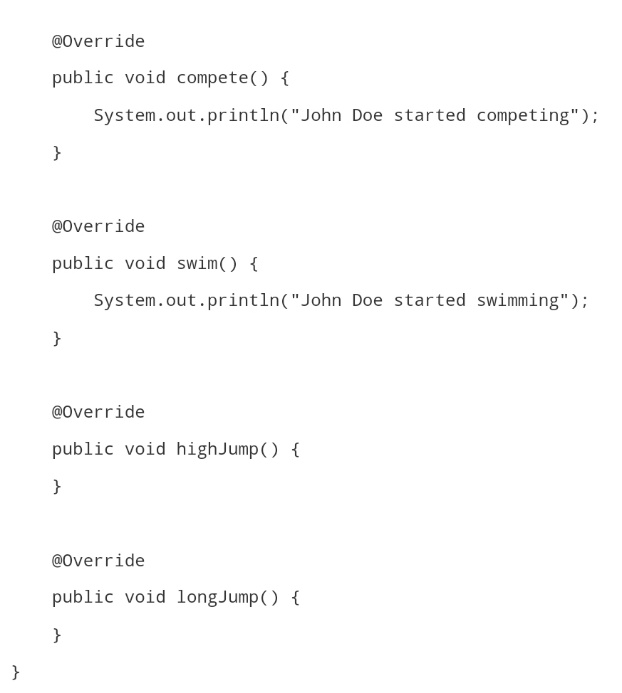
El código mostrado presenta una interfaz llamada pájaro junto a una interfaz que implementa la interfaz anterior, llamada FlyingBirds. El propósito de esta interfaz es poder separar a las aves que vuelan de las que no, y así no tener comportamientos inesperados en nuestro programa.

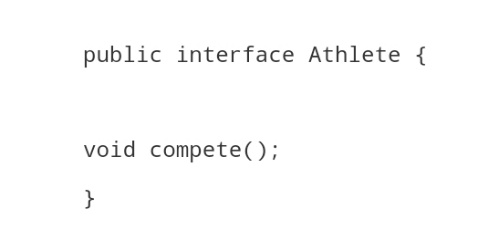
Por ejemplo:

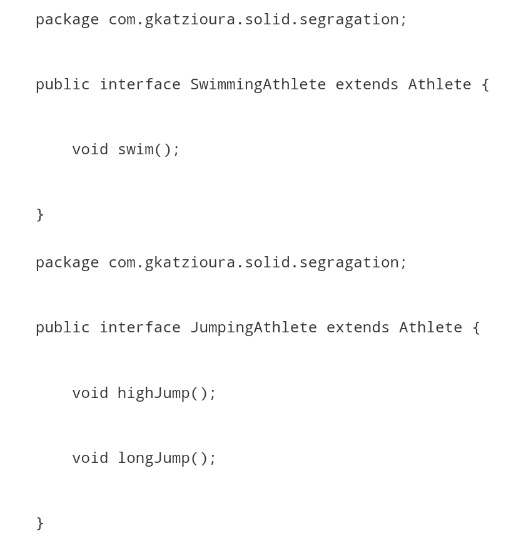


Si no hubiéramos definido la interfaz FlyingBirds el ejecutar el método fly() sobre un objeto de tipo Ostrich generaría que el programa se caiga, violando así el principio LSP, motivo por el cual debemos manejar esto para evitar comportamientos no deseados en clases que sean subtipos.

## 1.4.- Interface Segregation Principle (ISP)

El principio de segregación nos indica que debemos separar los métodos que no sean necesarios o no serán implementados, así no necesitaremos alterar todo cuando ciertos métodos no sean necesario implementarlos o por el contrario necesitemos implementarlo.

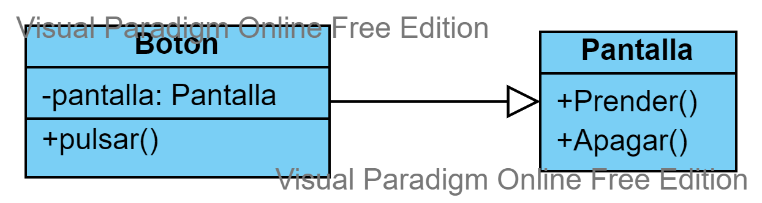
En el ejemplo podemos ver como no se debe implementar puezs la interfaz atleta tiene métodos que no son implementados por ello no cumple con el ISP.

Una correcta forma de implementar es.

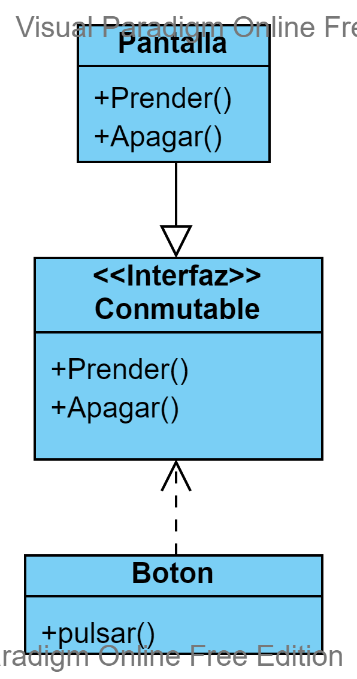
Como podemos ver no todos los atletas practican todos los deportes, pero lo que si tienen en común es que todos compiten por ello la interfaz atleta tiene integrado competir y de ahí se crean nuevas interfaces según el tipo de atleta, claro adquiriendo también competir.

## 1.5 Dependency Inversion Principle (DIP)

El principio de inversión de dependencias nos dice que los módulos de alto nivel que en este ejemplo lo ocupa la clase pantalla, no debe ni debería depender de módulos de bajo nivel como lo es la clase botón. De igual manera se puede evidenciar que las abstracciones dentro de botón no han sido separadas de los detalles, como resultado de esto se sabe que botón conoce la forma de prender y apagar la pantalla provocando una notable dependencia en los detalles de la clase pantalla.



La manera en que se puede solucionar y aplicar el principio de inversión de dependencias, es creando una interfaz que está ligada con la clase botón y esta pueda ser implementada desde cualquier otra clase como lo es pantalla. Como resultado de esto, botón ya no se ve obligado a depender de ninguna otra clase y los cambios que se realicen en la clase pantalla no afectaran en absoluto a la clase botón.



# 2.- Caso de estudio código fuente Sección B

La clase pago incumple el OCP ya que cuenta con métodos que inicializan la clase Compra por cada uno de los tipos de pago que existen, en el caso de que se quiera agregar más métodos de pago sería necesario modificar la clase Compra. Es por esto que se incumple OCP. Una posible solución es crear una interfaz llamada pago que tenga el método setPago() que define la lógica de cada forma de pago nueva que se necesite agregar en un futuro.

La clase Notificacion incumple OCP ya que verifica el tipo de notificación cada vez que se ejecuta el método notificar() lo que imposibilita la creación de nuevos tipos de notificaciones, ya que se debería agregar código por cada tipo que se cree. Cambiar la clase notificación por una interfaz que solo tenga el método notificar y crear clases que implementen a notificación que describan el comportamiento de la notificación exacta que se quiere mandar

La clase pago PayPal incumple el principio de inversión de dependencias debido a que pago depende de un estado de login de una clase derivada a Pago. Lo que ocasiona un mal acoplamiento entre las clases. La clase Pago además incumple el LSP ya que se está implementando un comportamiento totalmente nuevo a la clase PagoPayPal. Una posible solución para eliminar esas dependencias seria la creación de una interfaz que actúe de mediador entre los distintos métodos de pago que puedan existir, y proveer los métodos correspondientes a la forma de pago sin ocasionar dependencias innecesarias.