INTERFACES E INYECCION DE DEPENDENCIAS EN C#

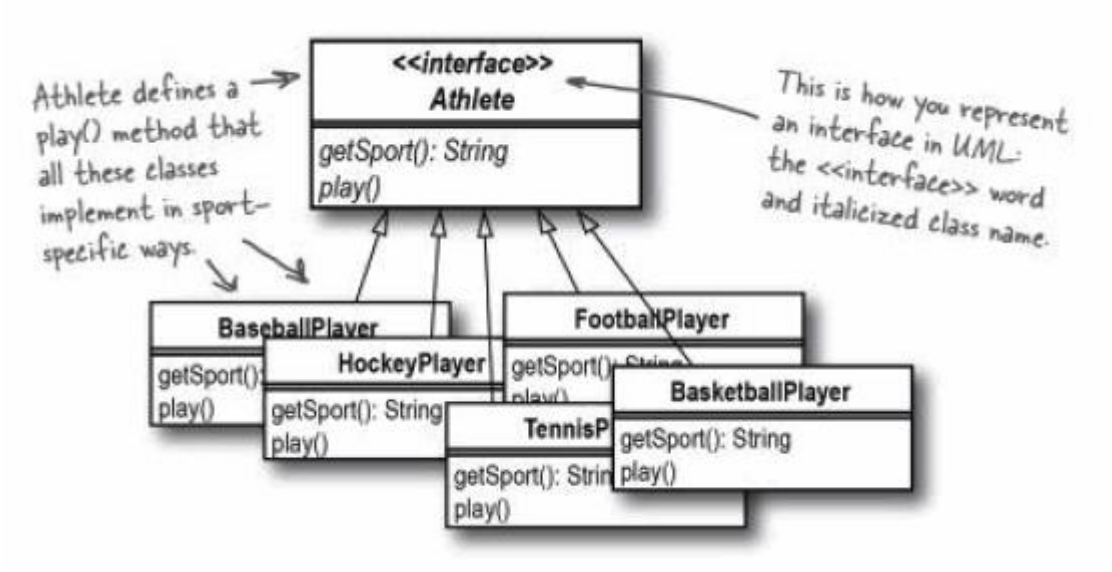
Introducción

- En esta clase, vamos a ampliar nuestros conocimientos de programación orientada a objetos en C# y vamos a enfocarnos en dos conceptos importantes: interfaces e inyección de dependencias.
- Estos conceptos nos ayudarán a escribir código más modular, reutilizable y fácil de mantener.

Interfaces

- En C#, una interfaz es una estructura que se compone de métodos y constantes.
- Los métodos de una interfaz son declarados pero no implementados.
- La interfaz indica qué métodos debe obligatoriamente implementar una clase.

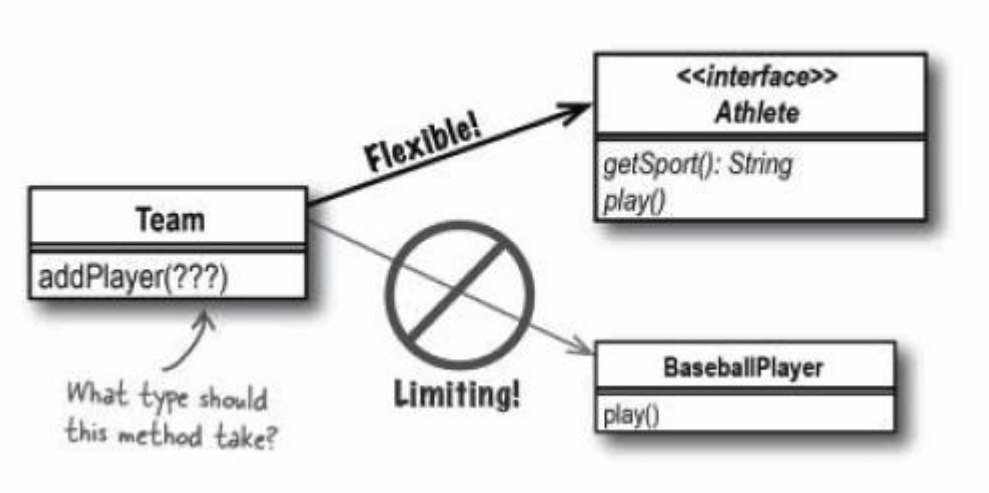
Interfaces



Fuente:

McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Interfaces



Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Interfaces

- ¿Por qué es tan importante? Porque le da flexibilidad a tu aplicación. En lugar de que tu código solo pueda trabajar con una subclase específica, como BaseballPlayer, puedes trabajar con algo más genérico, como Athlete. Esto significa que tu código funcionará con cualquier subclase de Athlete, como HockeyPlayer o TennisPlayer, e incluso con subclases que aún no han sido diseñadas (¿alguien dijo CricketPlayerP?).

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Interfaces - Ejemplo

- En este ejemplo, se define una interfaz IJugador que tiene dos métodos comunes a cualquier deporte que implique jugar en equipo: Correr() y Pasar(). Luego, se define una clase base abstracta Deporte que implementa la interfaz y dos clases concretas Futbol y Baloncesto que heredan de la clase base y la interfaz.

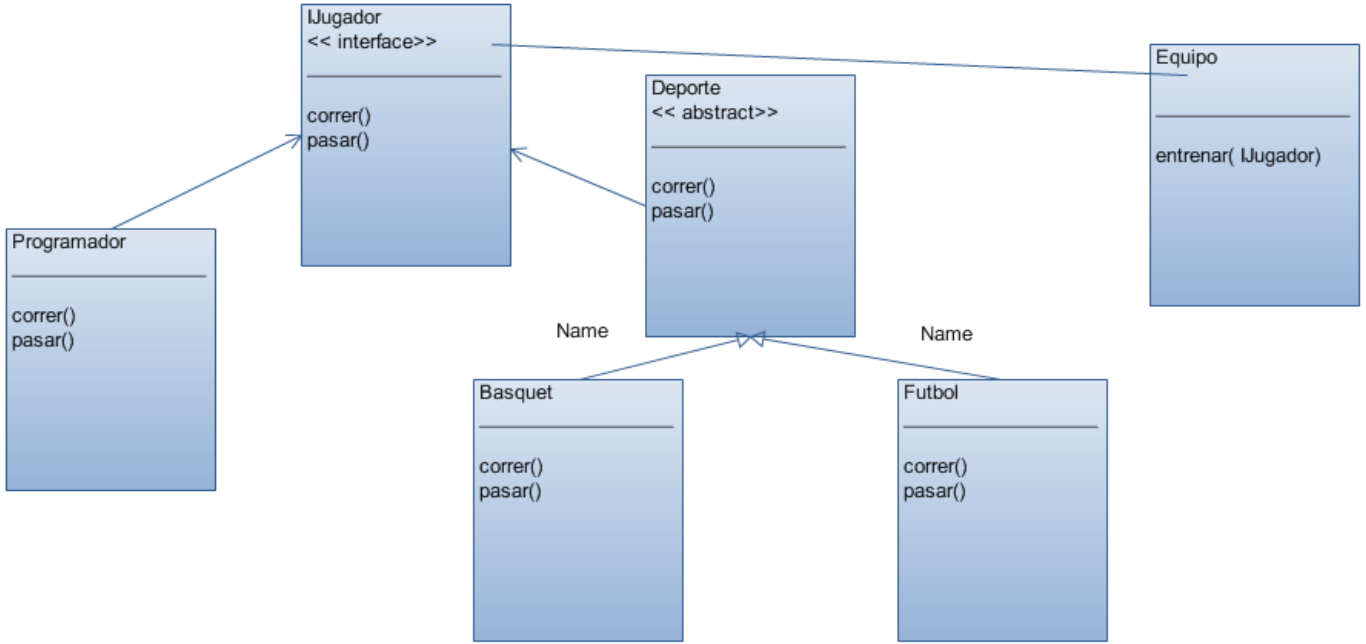
Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Interfaces - Ejemplo

- Finalmente, se muestra un ejemplo de cómo se puede utilizar la interfaz para llamar a los métodos comunes de cualquier deporte en la clase Equipo, que recibe como parámetro un objeto que implemente la interfaz IJugador.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Interfaces - Ejemplo



Interfaces - Ejemplo

```
// Declaración de la interfaz
public interface IJugador
{
    void Correr();
    void Pasar();
}

// Clase base abstracta que implementa la interfaz
public abstract class Deporte : IJugador
{
    public abstract void Correr();
    public abstract void Pasar();
}
```

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Interfaces - Ejemplo

```
// Clases que heredan de la clase base y la interfaz
public class Futbol : Deporte
{
    public override void Correr()
    {
        Console.WriteLine("Corriendo en el campo de fútbol");
    }

    public override void Pasar()
    {
        Console.WriteLine("Pasando la pelota a un compañero");
    }
}
public class Baloncesto : Deporte
{
```

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Interfaces - Ejemplo

```
// Clases que heredan de la clase base y la interfaz
public class Baloncesto : Deporte
{
    public override void Correr()
    {
        Console.WriteLine("Corriendo en la cancha de baloncesto");
    }

    public override void Pasar()
    {
        Console.WriteLine("Pasando el balón a un compañero");
    }
}
```

Fuente:

McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Interfaces - Ejemplo

```
// Clase que utiliza la interfaz para llamar a los métodos comunes
public class Equipo
{
    public void Entrenar(IJugador jugador)
    {
        jugador.Correr();
        jugador.Pasar();
    }
}
```

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Interfaces - Ejemplo

- // Ejemplo de uso
- var equipoFutbol = new Futbol();
- var equipoBaloncesto = new Baloncesto();

- var equipo = new Equipo();
- equipo.Entrenar(equipoFutbol);
- equipo.Entrenar(equipoBaloncesto);

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Vehículos: supongamos que tienes una aplicación que tiene una clase llamada "Vehicle". Podrías crear una interfaz llamada "IDriveable" que tenga un método "Drive()" que todos los vehículos deberían implementar. De esta manera, podrías crear diferentes tipos de vehículos (automóviles, motocicletas, camiones, etc.) que tengan su propia implementación del método "Drive()".

Animales: podrías tener una interfaz llamada "IAnimal" que tenga un método "MakeSound()" que todos los animales deberían implementar. Luego, podrías crear diferentes tipos de animales (gatos, perros, vacas, etc.) que tengan su propia implementación del método "MakeSound()".

Inyección de dependencias

- DEFINICIÓN La inyección de dependencia es un conjunto de principios y patrones de diseño de software que nos permiten desarrollar código poco acoplado.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Inyección de dependencias

- La DI (Inyección de Dependencias) no es un objetivo final en sí mismo, sino un medio para lograr otros objetivos.
- La DI permite el acoplamiento débil, lo que hace que el código sea más mantenible.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Inyección de dependencias

- En el desarrollo de software, estamos aprendiendo aún cómo implementar una buena arquitectura, pero otras profesiones, como la construcción, ya han encontrado soluciones a problemas similares.
- El acoplamiento estrecho (tightly coupled) es como conectar directamente un secador de pelo a la pared, lo que dificulta la modificación de uno sin afectar al otro.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Inyección de dependencias

Figure 1.4 In a cheap hotel room, you might find the hair dryer wired directly into the wall outlet. This is equivalent to using the common practice of writing *tightly coupled* code.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Inyección de dependencias

- El acoplamiento débil (loosely coupled) se parece más a la conexión de electrodomésticos con enchufes y tomas de corriente. Esto permite conectar dispositivos de diferentes tipos de forma flexible.
- Los enchufes y tomas de corriente pueden compararse a patrones de diseño de software conocidos, como el principio de sustitución de Liskov.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Inyección de dependencias

- La flexibilidad de las tomas de corriente y enchufes permite adaptarse a necesidades futuras que no se habían previsto originalmente.
- La utilización de enchufes y tomas de corriente es un ejemplo de cómo se pueden aplicar principios de diseño en diferentes campos.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Inyección de dependencias

- Inyección de dependencias en el constructor
- Inyección de dependencias en métodos

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Encapsulación

- La encapsulación es importante para prevenir la duplicación de código.
- La encapsulación también ayuda a proteger tus clases de cambios innecesarios.
- Si tienes comportamientos en tu aplicación que probablemente cambiarán, debes separarlos de las partes que no cambiarán con tanta frecuencia.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Encapsulación

- La encapsulación implica separar y ocultar la complejidad de una clase, y exponer solo la interfaz necesaria para trabajar con ella.
- Al encapsular correctamente, puedes hacer que tu código sea más mantenible y escalable en el futuro.

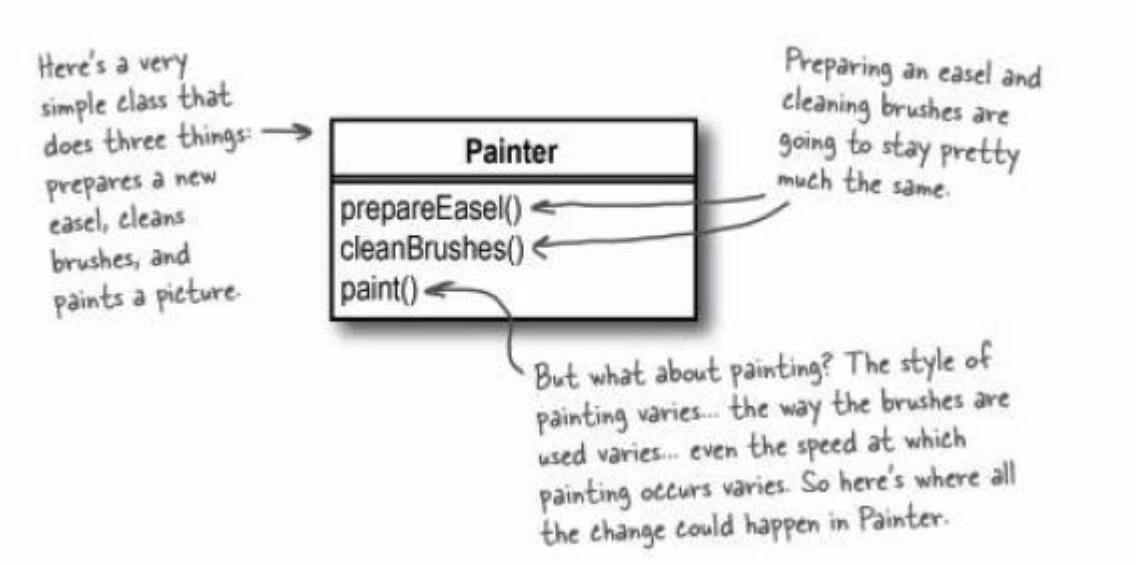
Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Encapsulación

- La encapsulación implica separar y ocultar la complejidad de una clase, y exponer solo la interfaz necesaria para trabajar con ella.
- Al encapsular correctamente, puedes hacer que tu código sea más mantenible y escalable en el futuro.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Encapsulación



Fuente:

McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

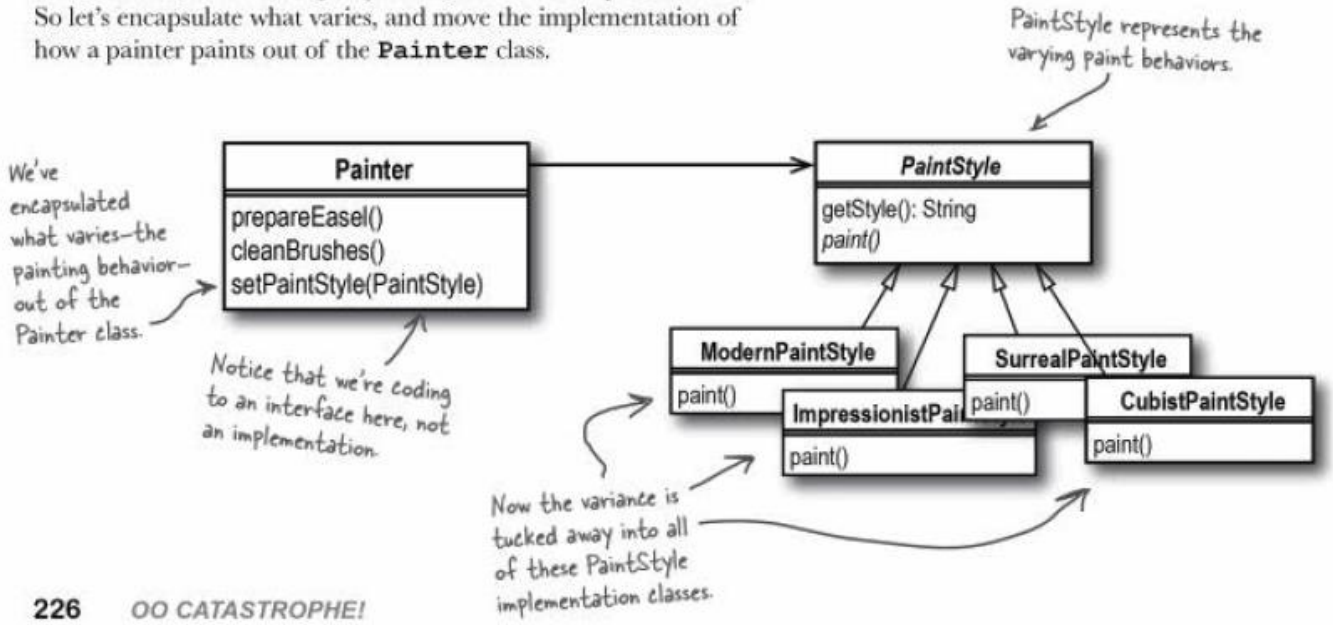
Encapsulación

- Parece que la clase Pintor tiene dos métodos que son bastante estables, pero el método paint() va a variar mucho en su implementación. Así que encapsulemos lo que varía y movamos la implementación de cómo un pintor pinta fuera de la clase Pintor.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Encapsulación

that `paint()` method is going to vary a lot in its implementation. So let's encapsulate what varies, and move the implementation of how a painter paints out of the **`Painter`** class.



Fuente:

McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

El cambio

- El cambio es la única constante en el software.
- El software mal diseñado se desmorona al primer signo de cambio, mientras que el gran software puede cambiar fácilmente.
- La forma más fácil de hacer que el software sea resistente al cambio es asegurarse de que cada clase tenga solo una razón para cambiar.

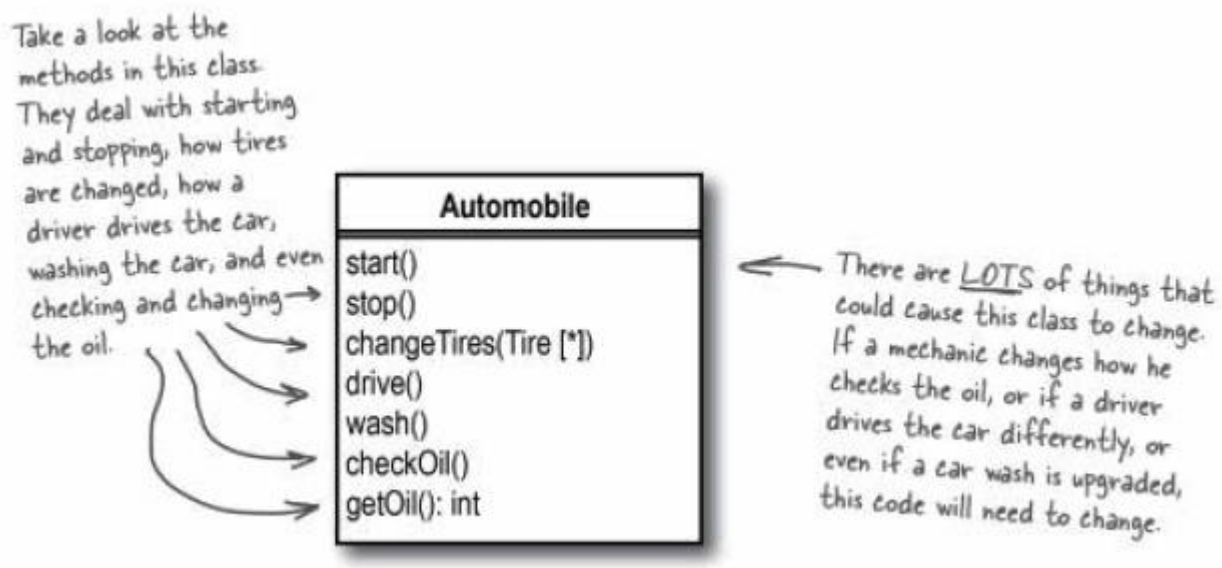
Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

El cambio

- Al reducir el número de cosas en una clase que pueden causar que cambie, se minimiza la posibilidad de que la clase tenga que cambiar.
- Cada clase debe tener una sola responsabilidad y no tener múltiples razones para cambiar.
- Si una clase tiene múltiples razones para cambiar, su diseño puede ser mejorado para reducir la dependencia de la clase.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

El cambio



Fuente:

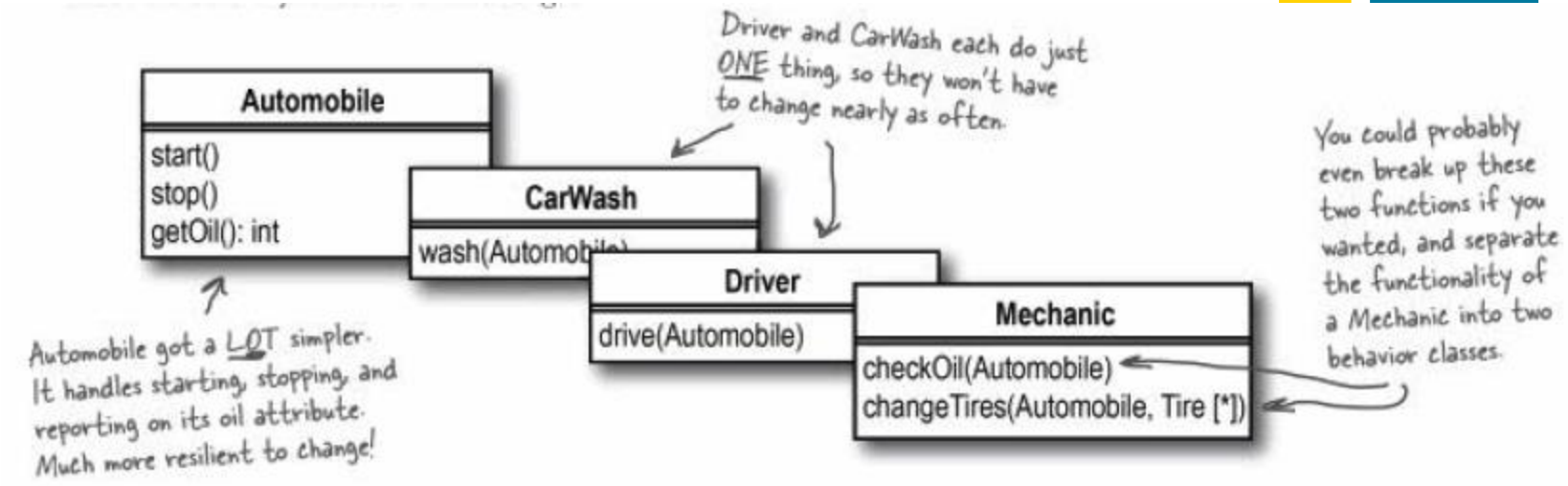
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

El cambio

- Cuando ves una clase que tiene más de una razón para cambiar, probablemente está tratando de hacer demasiadas cosas. Intenta dividir la funcionalidad en varias clases, donde cada clase individual hace solo una cosa, y por lo tanto solo tiene una razón para cambiar.

Fuente:
McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

El cambio



Fuente:

McLaughlin, B., Pollice, G., & West, D. (2007). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. " O'Reilly Media, Inc.".

Preguntas?

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash