<!--SOLID-->

SOLID - Segregación de interfaz

<Por="Raquel Martinez"/>

}

Definición{

No se debe obligar a los clientes a depender de métodos que no utilizan. Las interfaces pertenecen a clientes, no a jerarquías.

}

Definición{

Si una clase no utiliza métodos o atributos particulares, entonces esos métodos y atributos deben segregarse en clases más específicas.

}

Interfaz {

En la programación orientada a objetos, un interfaz define al conjunto de métodos que tiene que tener un objeto para que pueda cumplir una determinada función en nuestro sistema. Dicho de otra manera, un interfaz define como se comporta un objeto y lo que se puede hacer con el.

}

Interfaz informal {

```python
class Mando:
    def siguiente_canal(self):
        pass
    def canal_anterior(self):
        pass
    def subir_volumen(self):
        pass
    def bajar_volumen(self):
        pass
```

```python
class MandoLG(Mando):
    def siguiente_canal(self):
        print("LG->Siguiente")
    def canal_anterior(self):
        print("LG->Anterior")
    def subir_volumen(self):
        print("LG->Subir")
    def bajar_volumen(self):
        print("LG->Bajar")
```

}

Interfaz formal {

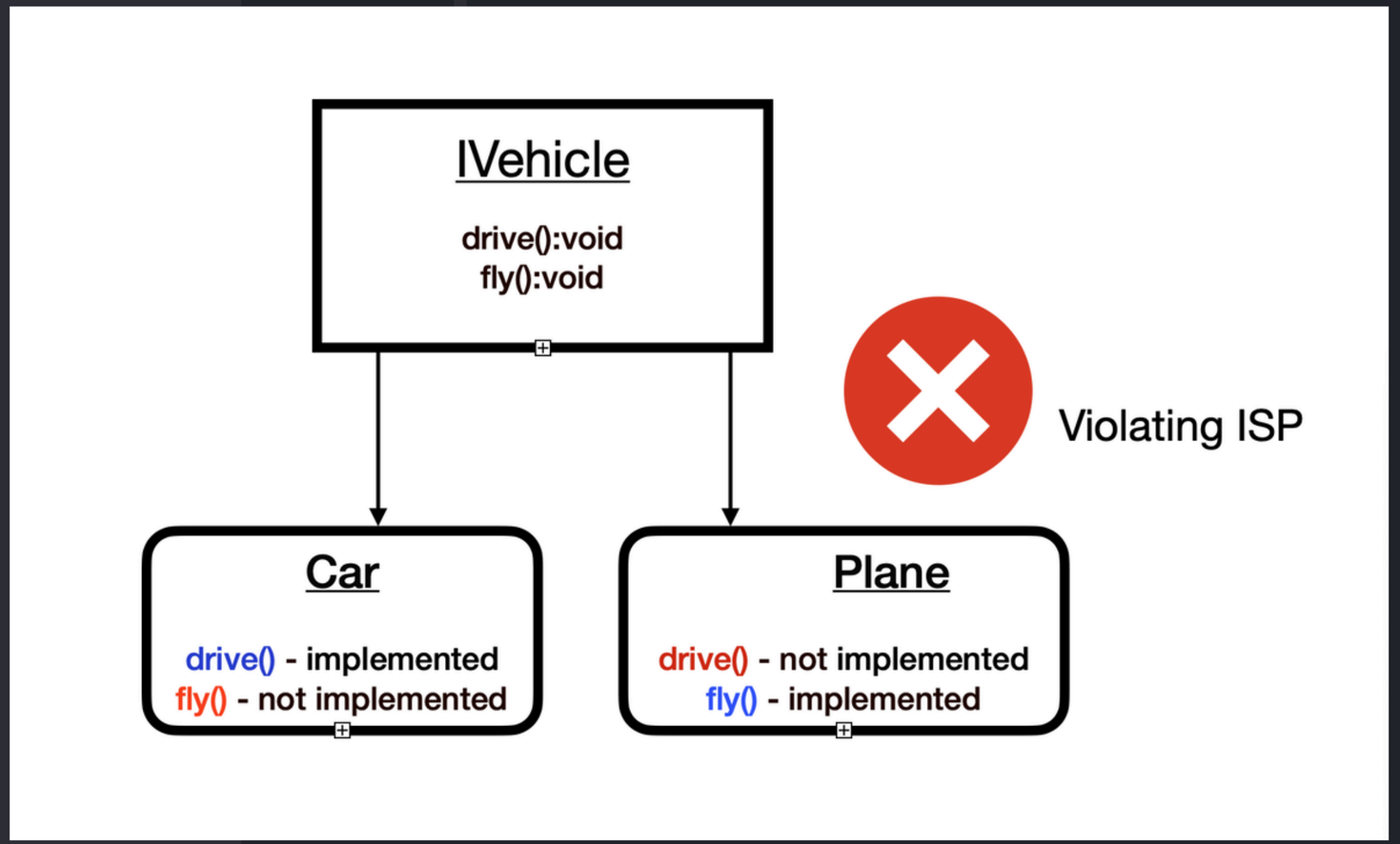
```python
from abc import abstractmethod
from abc import ABCMeta

class Mando(metaclass=ABCMeta):
    @abstractmethod
    def siguiente_canal(self):
        pass

    @abstractmethod
    def canal_anterior(self):
        pass

    @abstractmethod
    def subir_volumen(self):
        pass

    @abstractmethod
    def bajar_volumen(self):
        pass
```

```python
class MandoSamsung(Mando):
    def siguiente_canal(self):
        print("Samsung->Siguiente")
    def canal_anterior(self):
        print("Samsung->Anterior")
    def subir_volumen(self):
        print("Samsung->Subir")
    def bajar_volumen(self):
        print("Samsung->Bajar")
```

}

Beneficios {

- Mantenimiento
- Flexibilidad
- Reutilización



}

Ejemplo {

```python
from abc import ABC, abstractmethod

class Worker(ABC):
    @abstractmethod
    def work(self):
        pass

    @abstractmethod
    def eat(self):
        pass
```

}

Ejemplo {

```python
class HumanWorker(Worker):
    def work(self):
        print("Human is working")

    def eat(self):
        print("Human is eating")

class RobotWorker(Worker):
    def work(self):
        print("Robot is working")

    def eat(self):
        raise NotImplementedError("Robots do not eat")
```

}

Ejemplo {

```python
from abc import ABC, abstractmethod

class Workable(ABC):
    @abstractmethod
    def work(self):
        pass

class Eatable(ABC):
    @abstractmethod
    def eat(self):
        pass
```

}

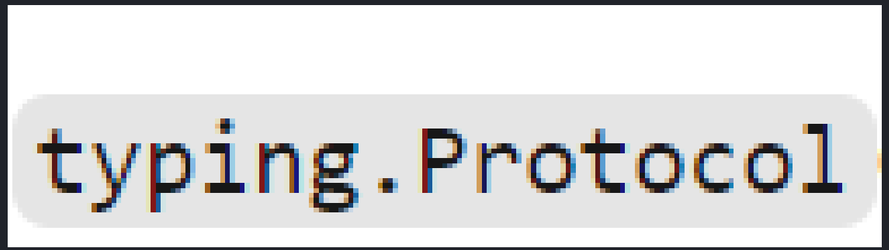
Ejemplo {

```python
class HumanWorker(Workable, Eatable):
    def work(self):
        print("Human is working")

    def eat(self):
        print("Human is eating")

class RobotWorker(Workable):
    def work(self):
        print("Robot is working")
```

}

Protocol {

- Un protocolo es un conjunto de métodos o atributos que un objeto debe tener para ser considerado compatible con ese protocolo. Los protocolos le permiten definir interfaces sin crear explícitamente una clase o heredar de una clase base específica.



}

<!--SOLID-->

Codifiquemos! {

<Ejemplo/>

}

<!--SOLID-->

Gracias {

<Por="Raquel Martinez/>

}