<!--SOLID-->

Herencia Vs Composición

{

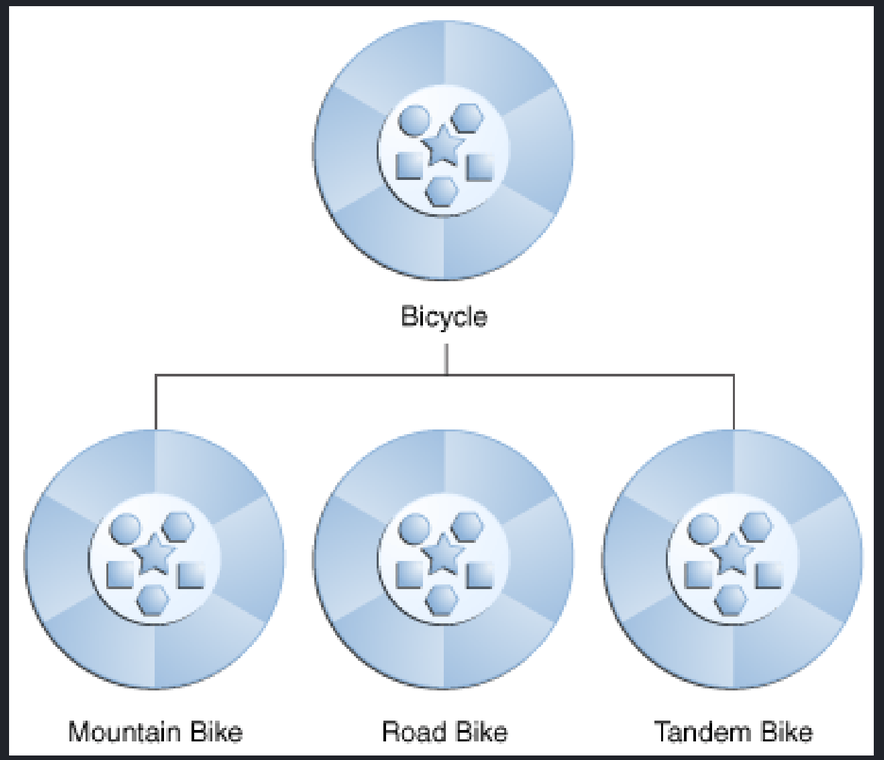
<Por="Raquel Martinez"/>

}

Contenidos

Herencia {

es un / es una



}

Ventajas de la herencia {

- Reutilización de código
- Polimorfismo
- Sobreescritura de métodos
- Desarrollo guiado

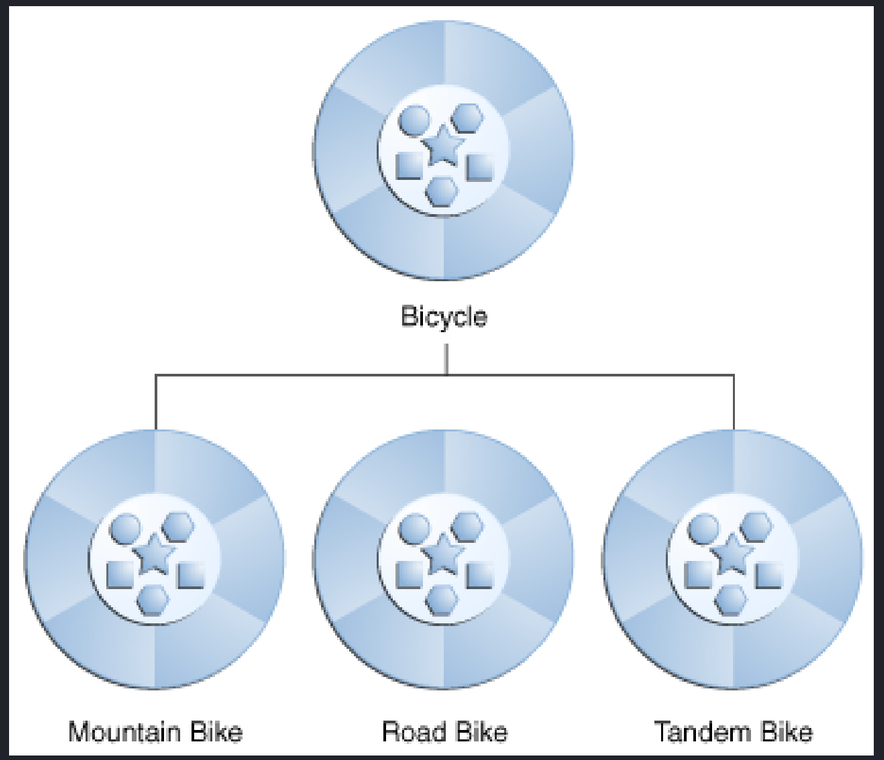
```python
class Figura:
    def calcular_area(self):
        return 0

class Rectangulo(Figura):
    def calcular_area(self):
        return self.ancho * self.altura
```

}

Desventajas de la herencia {

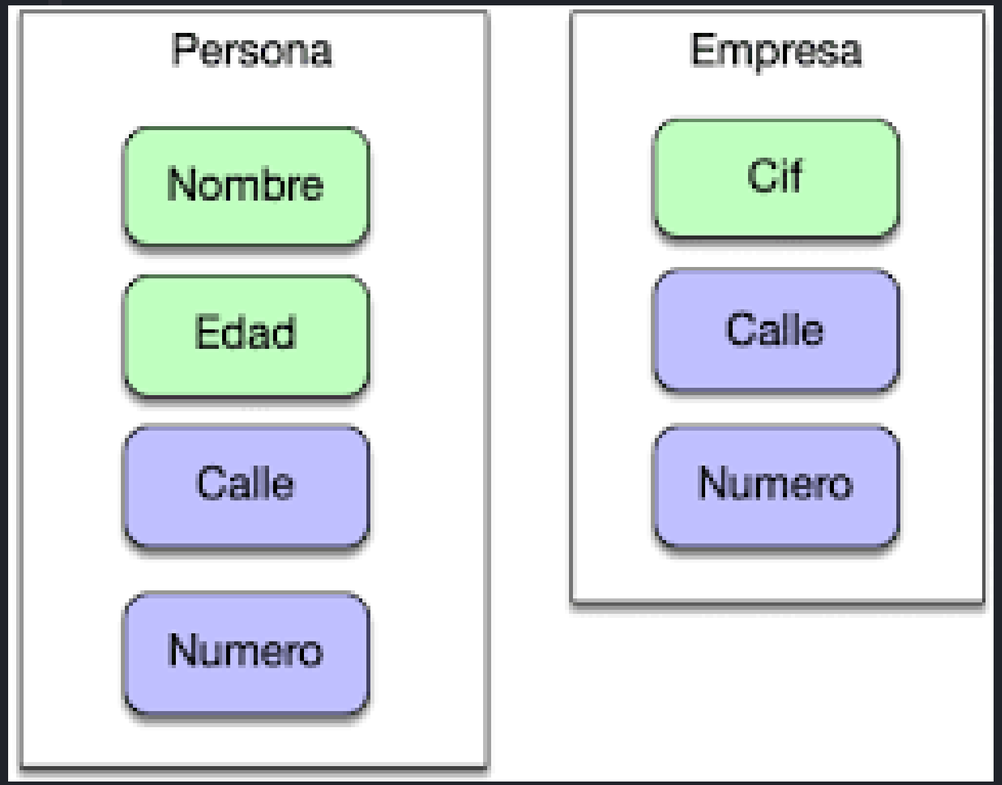
- Proyectos ágiles
- "la fiesta de los override"
- "la herencia de 7 niveles"



}

Composición {

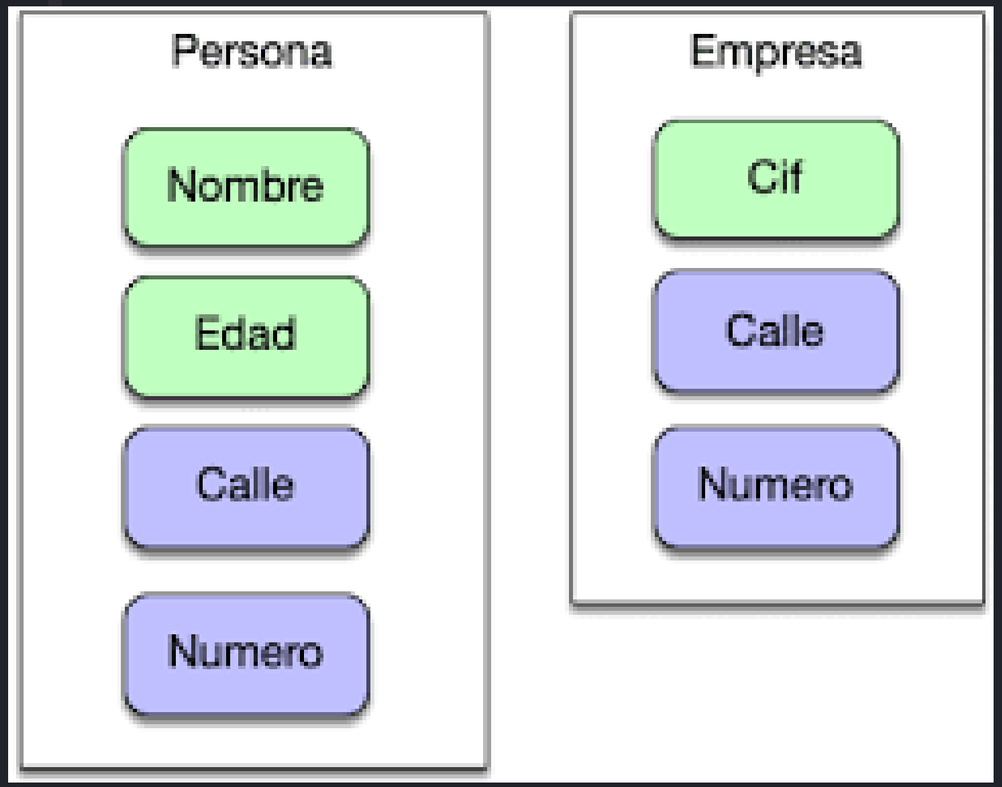
- Composición quiere decir que tenemos una instancia de una clase que contiene instancias de otras clases que implementan las funciones deseadas.



}

Ventajas de la composición {

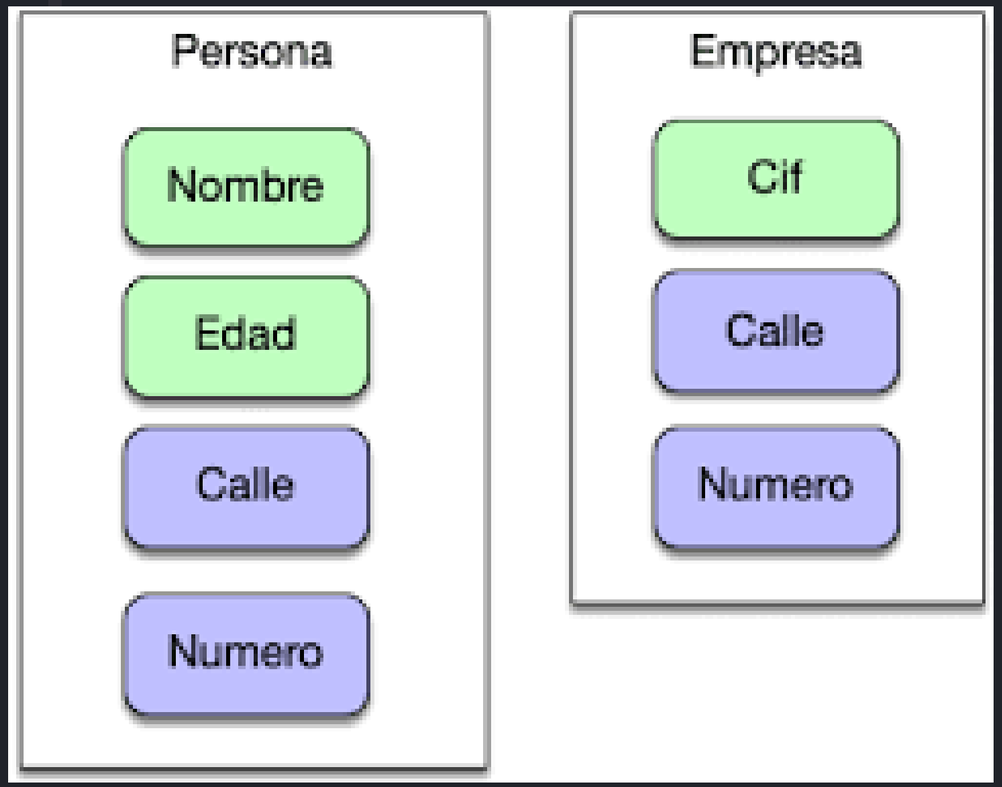
- Versatilidad
- Creación/destrucción dinámica
- Polimorfismo mediante interfaces



}

Composición cuando? {

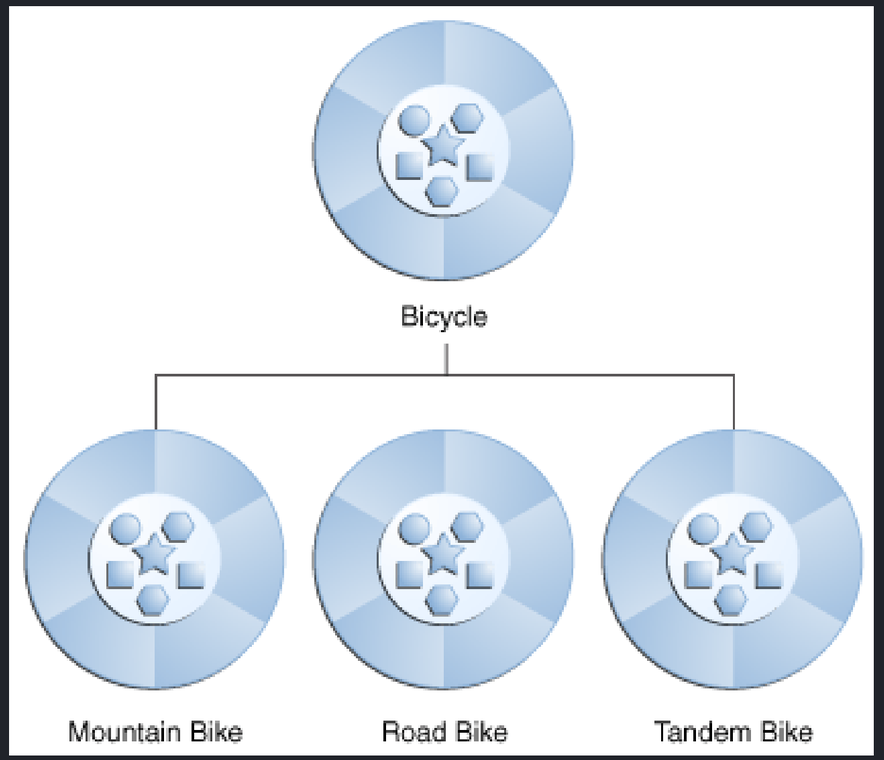
- No existe una relación de tipo "es un" clara y permanente
- Se necesita flexibilidad y capacidad de cambio
- Se busca desacoplar y reutilizar componentes
- Se requiere implementar comportamientos intercambiables
- Se tiene una relación de tipo "tiene un"



}

Herencia cuando? {

- Existe una relación de tipo "es un" clara y permanente
- Se busca reutilizar código y comportamiento común
- Se necesita polimorfismo estático
- Se modela el dominio del problema



}

<!--SOLID-->

Codifiquemos! {

<Ejemplo/>

}

Principios de diseño {

Que son?

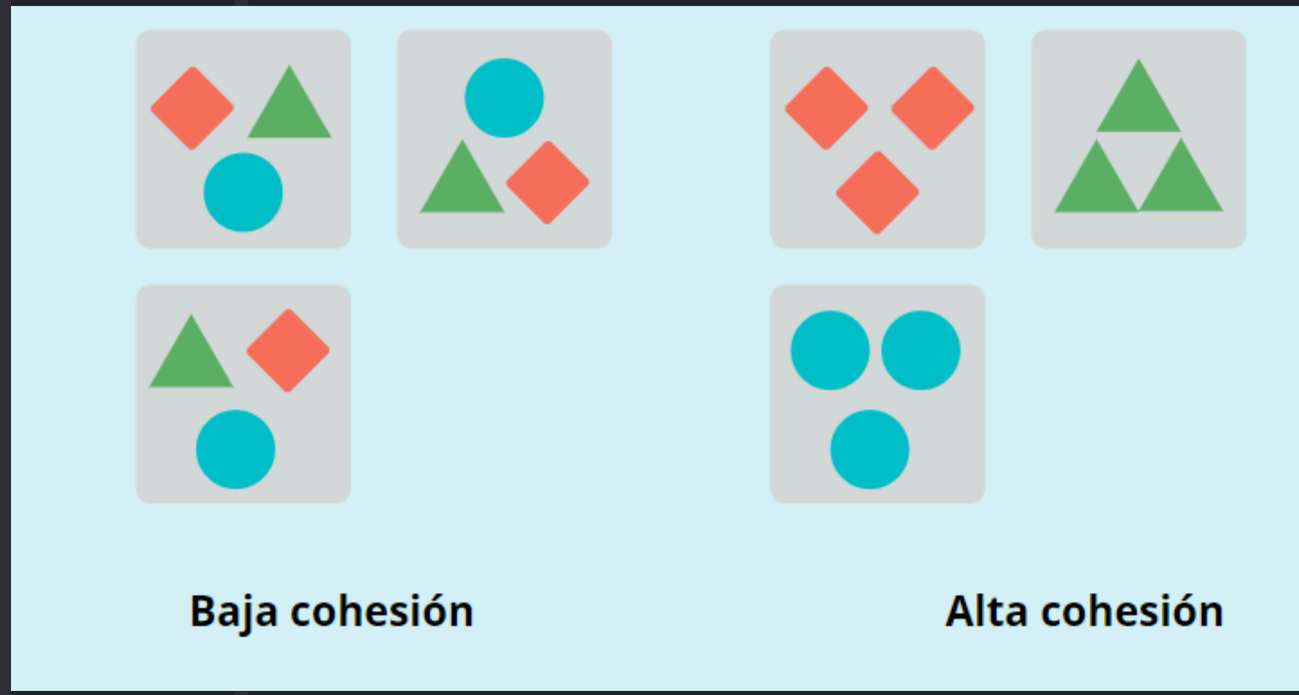
Cohesión y el acoplamiento

Los principios de diseño en software son directrices generales que ayudan a los desarrolladores a crear sistemas de software que sean fáciles de entender, mantener y extender.

}

Cohesión {

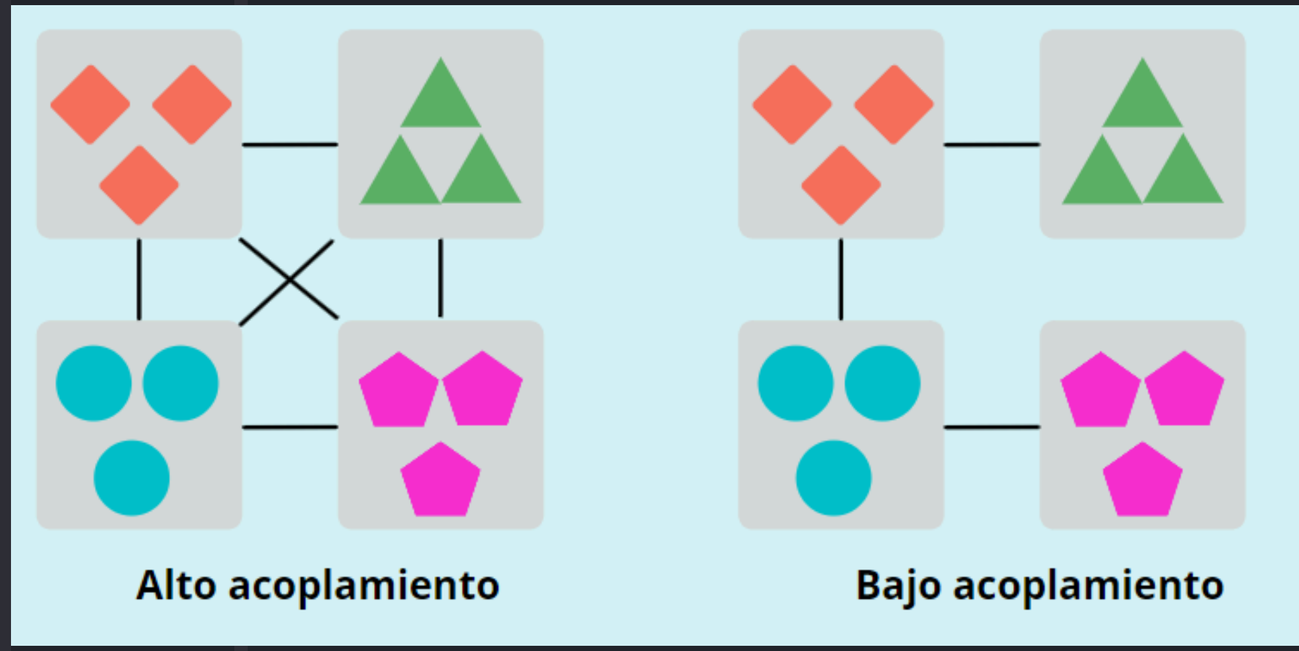
- La cohesión se refiere a la medida en que las responsabilidades y funciones de un módulo (clase, método o componente) están relacionadas y enfocadas en una única tarea o propósito.



}

Acoplamiento {

- Grado de interdependencia entre los diferentes módulos de un sistema.



}

Alta cohesión

Facilita la comprensión del código

Promueve la reutilización de código

Simplifica las pruebas y el mantenimiento

}

Bajo acoplamiento

Permite que los módulos sean más independientes

Facilita el mantenimiento y la evolución del código

Promueve la modularidad y la separación de preocupaciones

}

<!--SOLID-->

Gracias {

<Por="Raquel Martinez/>

}