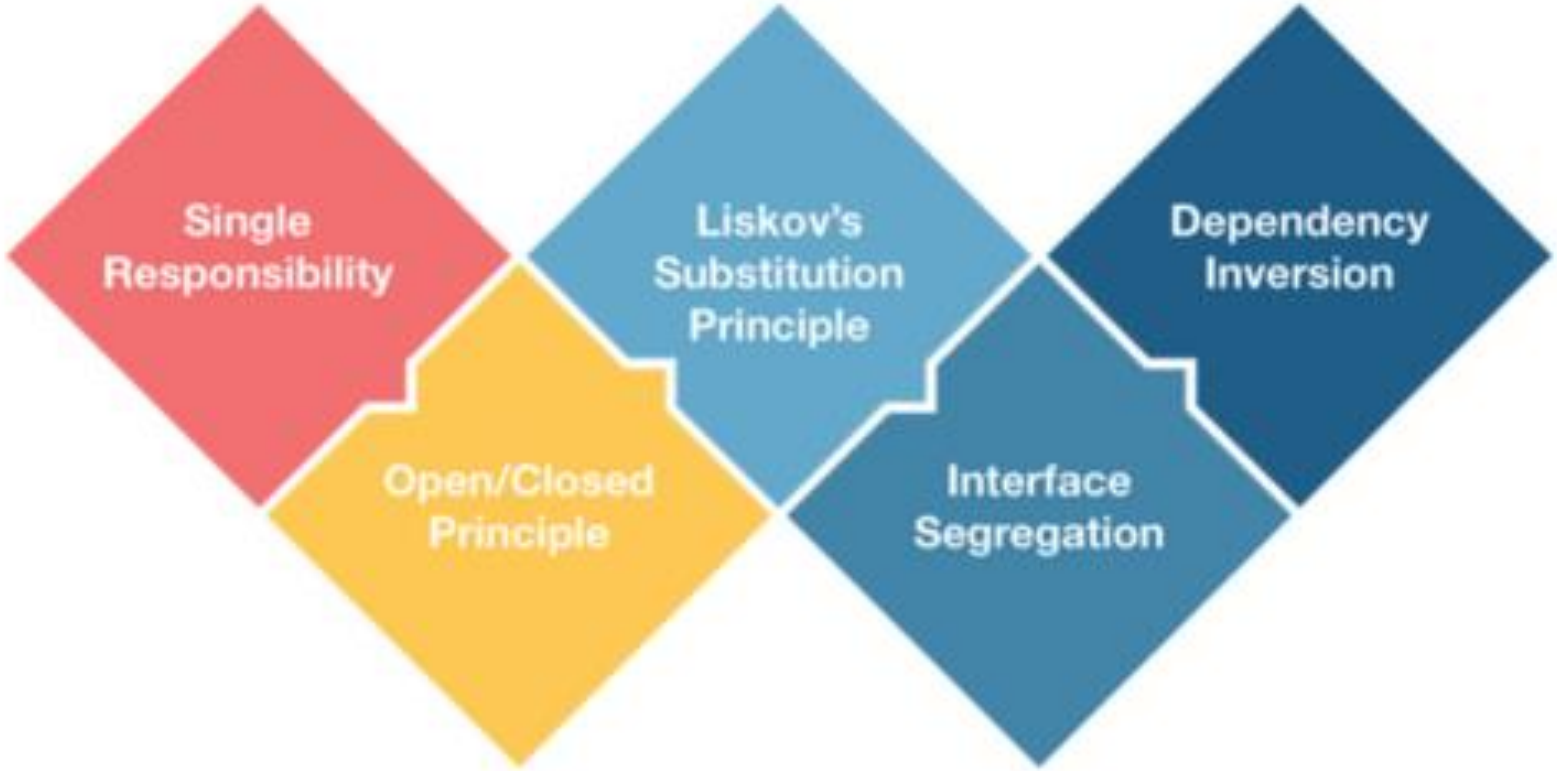
SOLID vs CUPID

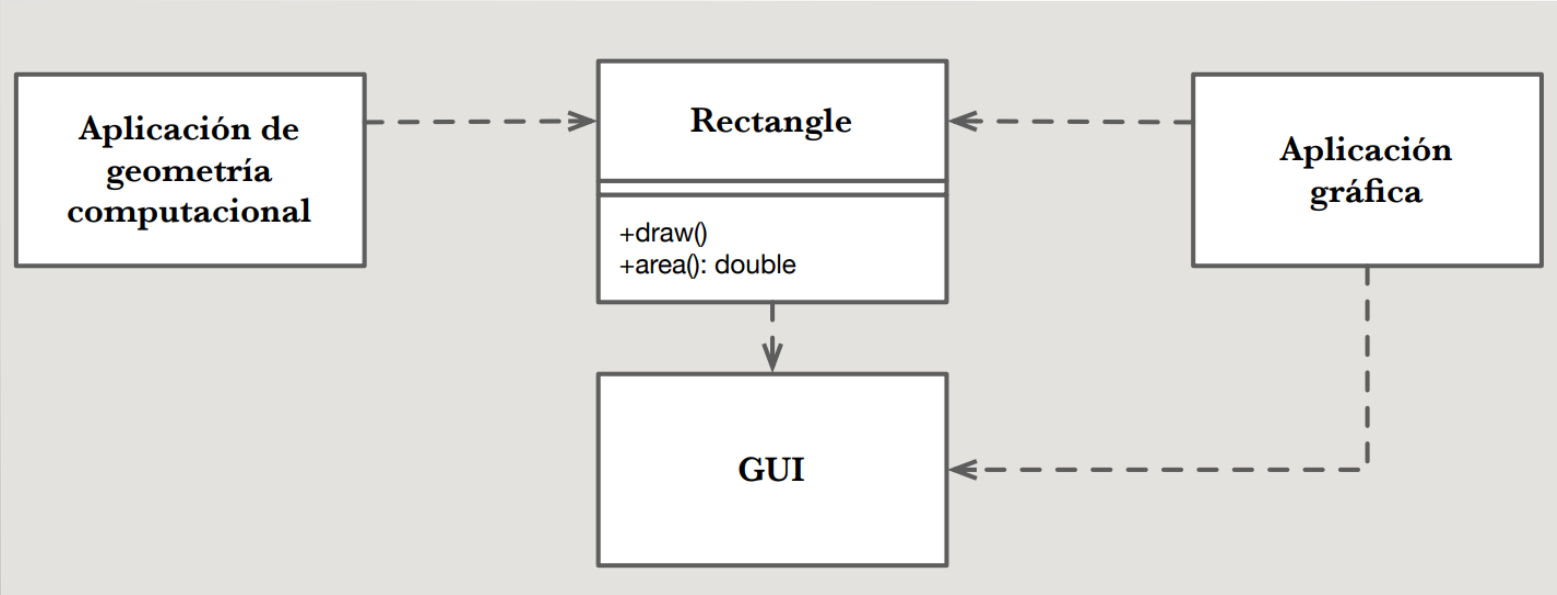
- Lucas Martínez Rego
- Hugo Gutiérrez Tomás
- Paula Puerta González
- Daniel Fernández Bernardino

S.O.L.I.D.


Single Responsibility
Open/Closed Principle
Liskov's Substitution Principle
Interface Segregation
Dependency Inversion

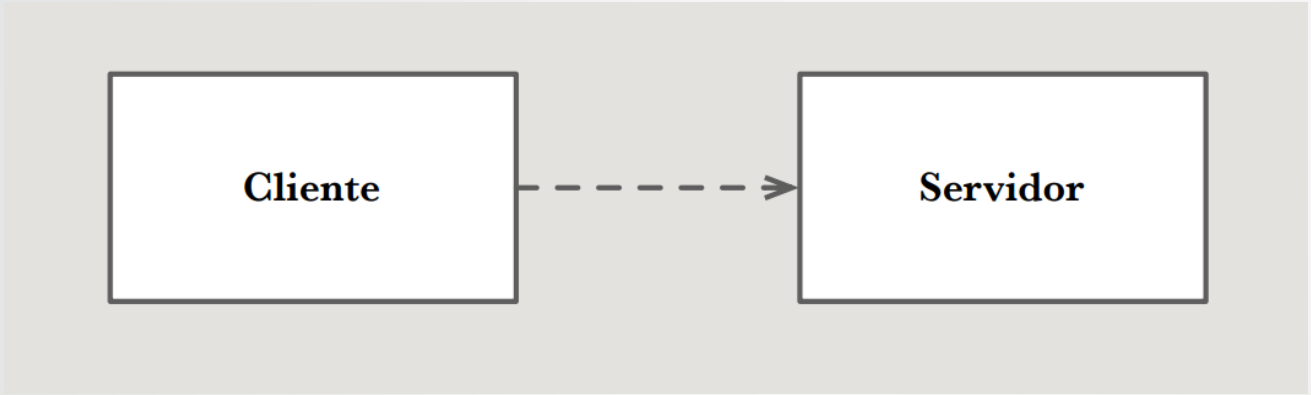
SOLID

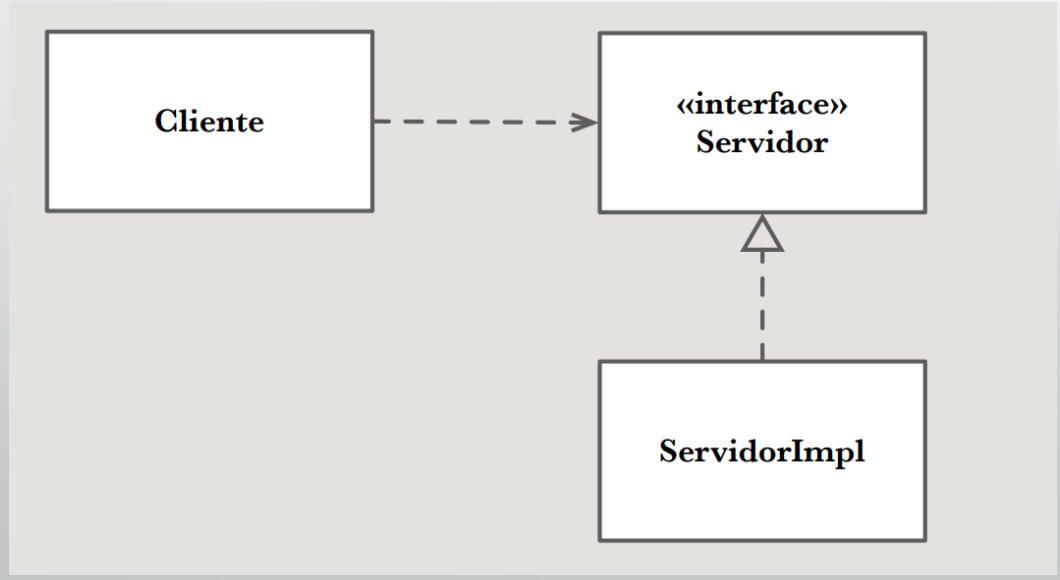
Principio de Responsabilidad Única (SRP)



SOLID

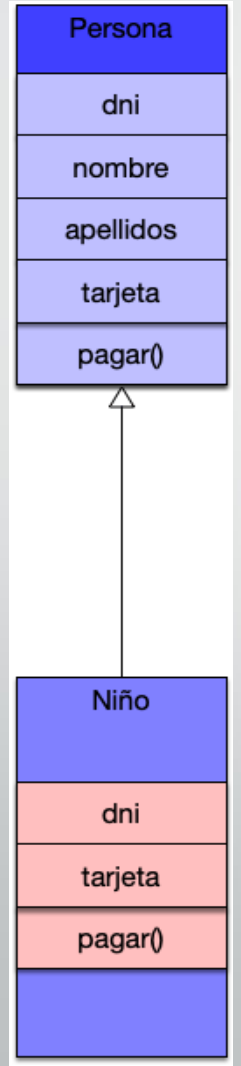
Principio de abierto-cerrado(OCP)





SOLID

Principio de sustitución de Liskov(LSP)



```java
package com.arquitecturajava;

public class Niño  extends Persona{

    public Niño(String nombre, String apellidos) {
        super(null, nombre, apellidos, null);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void pagar() {
        // TODO Auto-generated method stub
        throw new RuntimeException("un niño no puede pagar");
    }


}
```

SOLID

Principio de segregación de interfaces(ISP)

```
public interface Database
{
    void add();
    void read();
}
```

```
public interface Database
{
    void add();
}
```

```
public interface DatabaseV1 extends Database
{
    void read();
}
```

SOLID

Principio de inversión de dependencias(DIP)

```
1.  class Shopping { ... }
2.
3.  class ShoppingBasket {
4.      fun buy(shopping: Shopping?) {
5.          val db = SqlDatabase()
6.          db.save(shopping)
7.          val creditCard = CreditCard()
8.          creditCard.pay(shopping)
9.      }
10. }
11.
12. class SqlDatabase {
13.     fun save(shopping: Shopping?) {
14.         // Saves data in SQL database
15.     }
16. }
17.
18. class CreditCard {
19.     fun pay(shopping: Shopping?) {
20.         // Performs payment using a credit card
21.     }
22. }
```

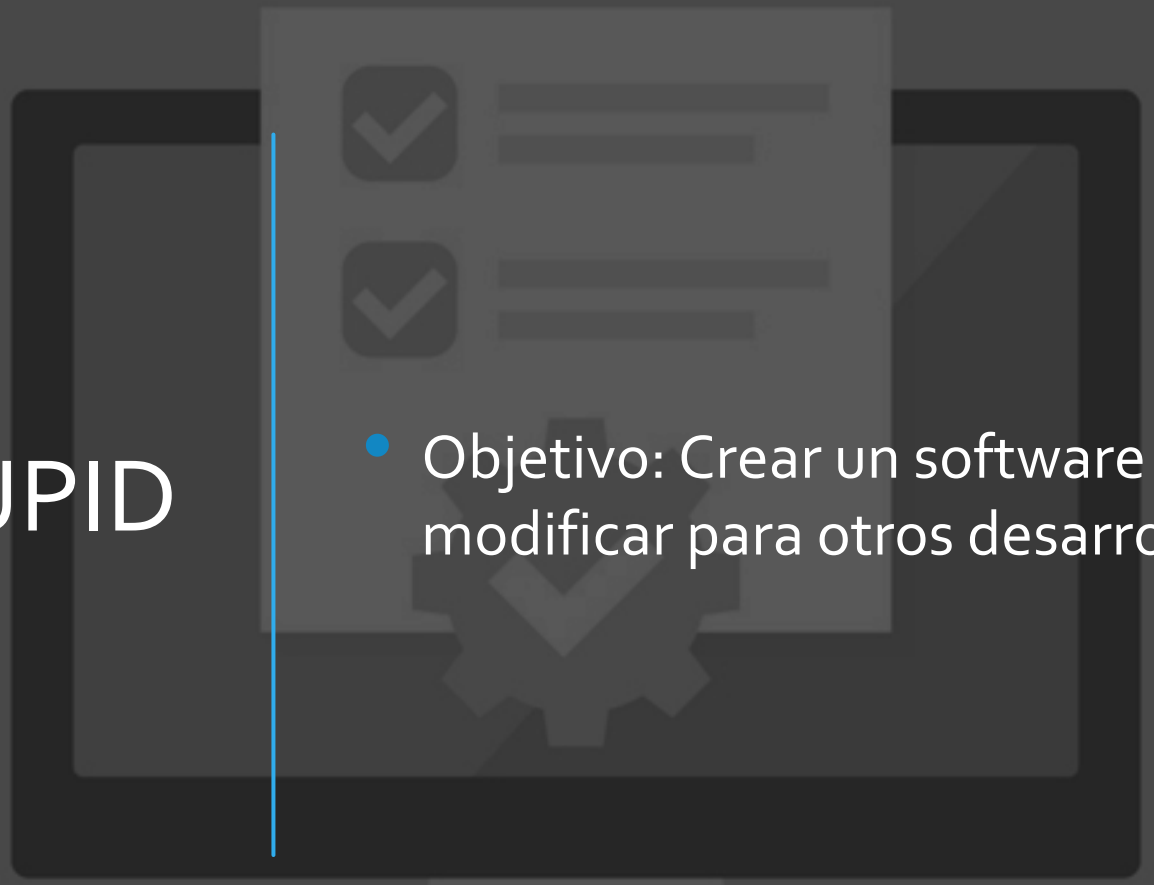
SOLID está mal según CUPID

- SOLID está anticuado
- Los problemas que soluciona ya no son comunes
- Propone soluciones demasiado enrevesadas

CUPID surge como una respuesta ante SOLID

- Propuesto por Dan North
- Rebate los principios SOLID uno a uno
- Resuelve los mismos problemas con una solución más simple
- Código más sencillo

CUPID



- Objetivo: Crear un software fácil de entender y modificar para otros desarrolladores.

Composable(Componible):

- Software fácil de usar y modificar tanto actualmente como con el paso de tiempo.
- Superficie pequeña: API limitada para evitar el sobre aprendizaje y tener menos errores.
- Intención reveladora: Nombrado de métodos y parámetros con el propósito de revelar lo que intentan hacer.

Unix-Philosophy: Hacer una cosa bien

- La filosofía Unix está presente en prácticamente todos los servidores comerciales, en la nube…
- Modelo simple y consistente: Los componentes deben hacer una sola cosa, pero hacerla muy bien. Estos se comunicarán entre ellos para lograr más potencia.

Predictable(Predecible):

- El código debe hacer lo que parece que hace, sin sorpresas desagradables en tiempo de ejecución, con un grado de consistencia alto.
- Se comporta como se espera: Utilización de pruebas para comprobar este requisito.
- Determinista: El software hace lo mismo cada vez. Nuestro código será mejor y ayuda a cumplir el punto anterior.

Idiomatic (Idiomático):

- Se beben establecer una serie de características que todos los desarrolladores llevarán a cabo.
- Modismo Lingüístico: Referido al propio código y como lo generamos: Estructura de módulos, nombrado de funciones, parámetros… Otros temas como la sangría, posición de las llaves…
- Modismo Local: Suelen producirse cuando el idioma no tiene un consenso sobre el estilo idiomático. Puede que no se produzca en todos los proyectos.

Domain-based (Basado en el Dominio):

- Consigue que el código transmita lo que está haciendo dentro del lenguaje de dominio.
- Lenguaje basado en el dominio: Temas como el nombrado de tipos y las operaciones a realizar deben ser creadas para facilitar la navegación por el código.
- Estructura basada en el dominio: Referido al nombrado de directorios y a la estructura de los mismos junto con sus relaciones es muy importante que reflejen el sistema que se está creando.
- Resultado: Implementar nuevos componentes en nuestro sistema será una tarea mucho más sencilla de realizar.

Comparativa SOLID vs CUPID

SOLID	CUPID
Principios = reglas	Propiedades = cualidades deseadas
Enfocado en el rendimiento del código	Enfocado en código fácilmente entendible por humanos
Necesarios conceptos teóricos para aplicarlo correctamente	Intuitivos, no precisan de ejemplos para comprenderse
Difícil aplicar de forma natural por inexpertos	Novatos pueden aplicarlo con sencillez
Evita cambios en el código ya existente (recompilación muy costosa en la época)	Espera que el cambio ocurra
Acorde a las tecnologías de los 90	Actualizado teniendo en cuenta los avances

CONCLUSIÓN

- Los principios SOLID necesitan ser actualizados de igual manera que el resto de hardware y software ha ido evolucionando.

MUCHAS GRACIAS