SOLID vs CUPID

...

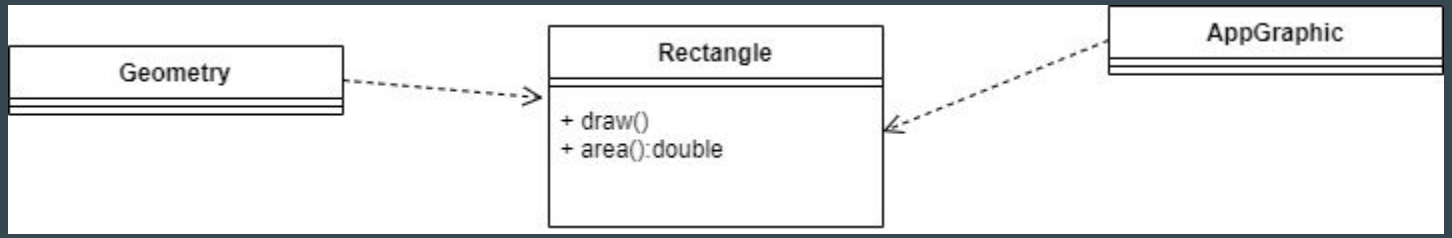
Carlos Garriga Suárez UO276903
Jesús González UO263799
Pablo López UO271580
Enzo Barbón UO270249

SOLID Principles

- (S)ingle Responsibility Principle
- (O)pen-Closed Principle
- (L)iskov Substitution Principle
- (I)nterface Segregation Principle
- (D)ependency Inversion Principle

Single Responsibility Principle

“A class only needs one reason to change”

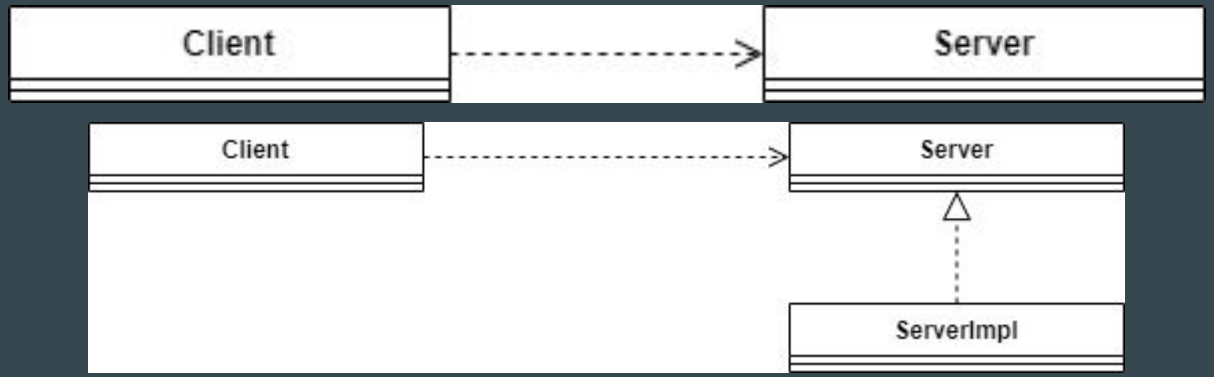


-Procedure

-Change the code you got in mind

Open-Closed Principle

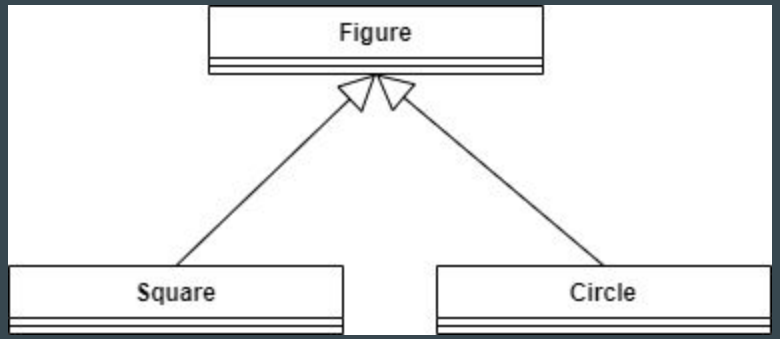
“Classes should be opened for extension but closed for modification”



-Just write easy code.

Liskov Substitution Principle

“Subtypes have to be able to substitute their base type”



-Wrong interpretation

Interface Segregation Principle

“It is better to have a lot of specific interfaces than a general one .”

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

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

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

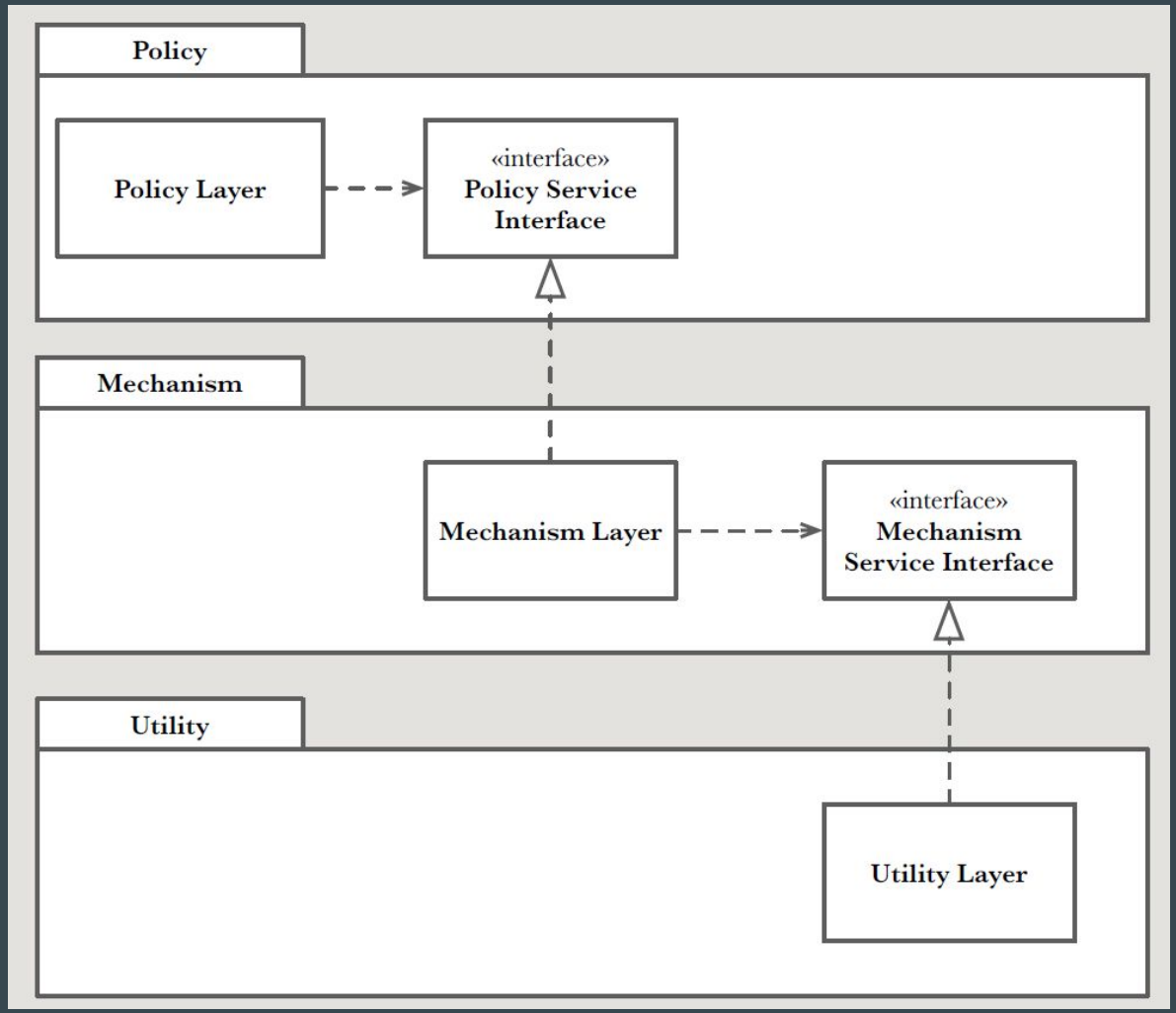
-Is more a design than a principle

Dependency Inversion Principle

“Top level modules must not depend on low level ones; both must depend on abstractions”

“Abstractions must not depend on details, it is the opposite way”

-Obsession have result into spending more money.


Policy
Policy Layer
«interface»
Policy Service
Interface
Mechanism
Mechanism Layer
«interface»
Mechanism
Service Interface
Utility
Utility Layer

Why every single element of Solid is wrong?

- Dan North propose CUPID

- Principles are rules 

- Properties are goals 

Properties of properties

- Practical
  - Easy to articulate
  - Easy to assess
  - Easy to adopt

- Human

- Layered

CUPID Properties

- Composable: Plays well with each other
- Unix philosophy: Does one thing well
- Predictable: Does what you expect
- Idiomatic: Feels natural
- Domain-based: The solution domain models the real problem domain

Composable

Software that is easy to use gets used, and used, and used again

- Small surface area

- Intention revealing

- Minimal dependencies

Unix philosophy

Do one thing and do it well

- A simple, consistent model

- Single purpose (vs Single Responsability)

Predictable

Code should do what it looks like it does

- Behave as expected

- Deterministic (Robust, Reliable, Resilient)

- Observable (Instrumentation, Telemetry, Monitoring, Alerting)

Idiomatic

- Code that humans can understand

- Language idioms

Domain-based

Software to meet a need. Code should convey what it is doing.

- Domain-based language

- Domain-based structure

- Domain-based boundaries