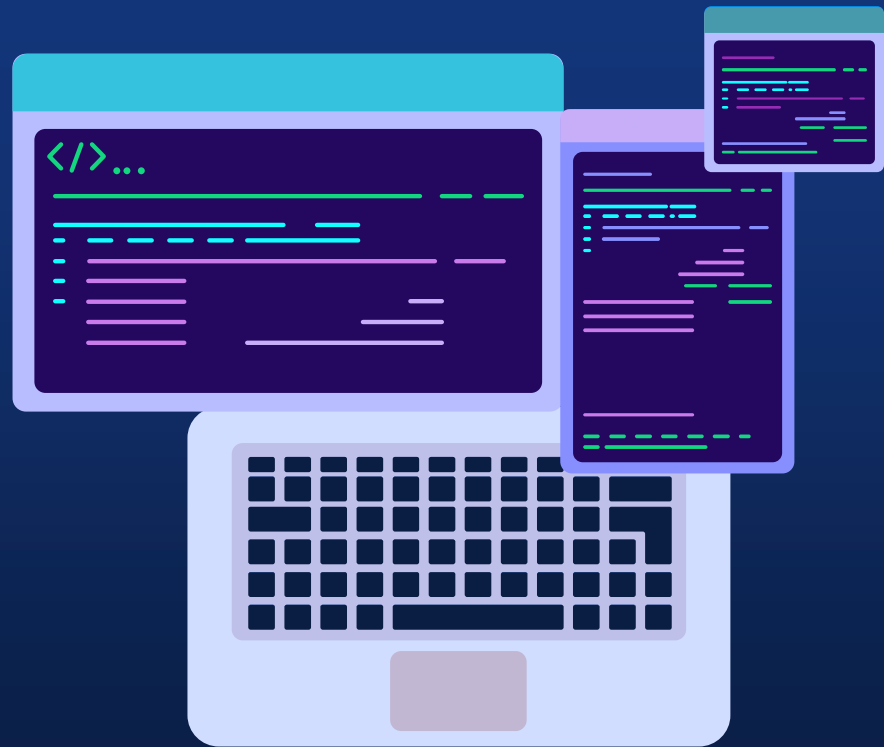# Software Design and Important concepts

Mentor: Einar Rocha

CONTENT

01
OOP Pillars

Inheritance, Polymorphism
Encapsulation, Abstraction

02
Clean Code

Meaningful Names,
Functions, Unit test
Code Smells…

03
SOLID

Single Responsiblity
Open closed
Liskov Substitution
Interface Segregation
Dependency Inversion

04
Design patterns

Singleton, Factory Method
Strategy, Observer
Builder…

03

SOLID

Purpose

To create understandable, readable, and testable code that many developers can collaboratively work on.

Agenda

Dependency Inversion Principle

Layering...

Final Work

Explanation...
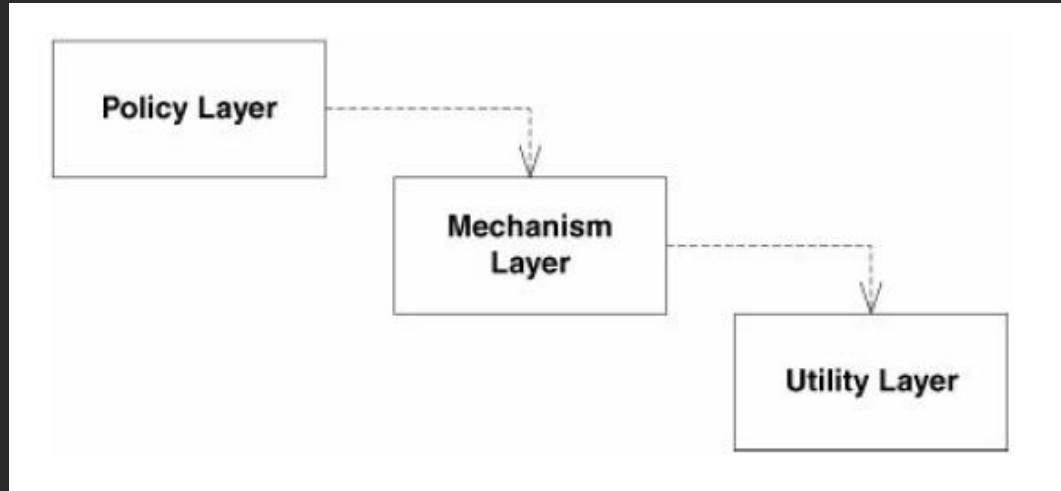
Dependency Inversion Principle

High-level modules should not depend on low-level modules. Both should depend on abstractions.
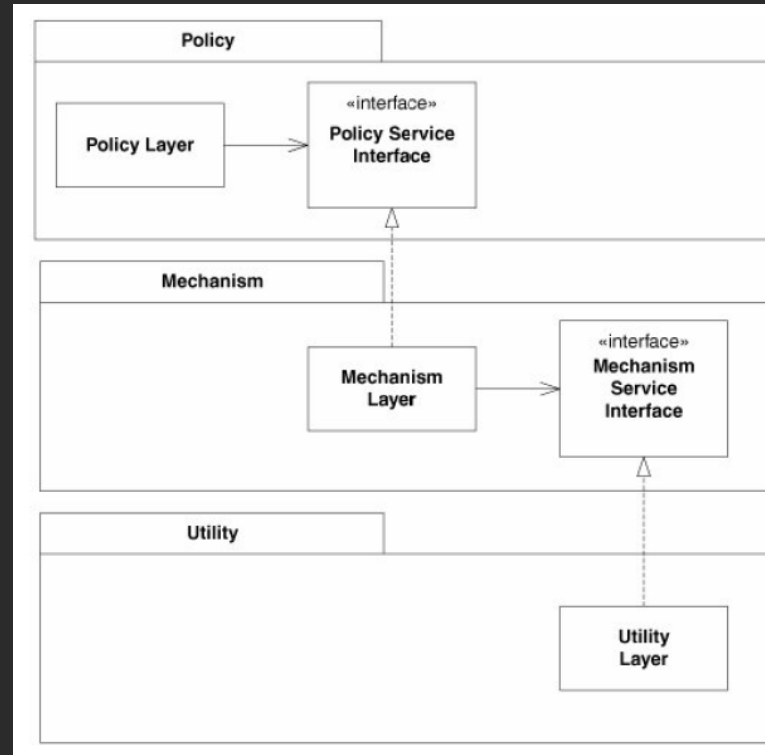
Abstractions should not depend upon details. Details should depend upon abstractions.
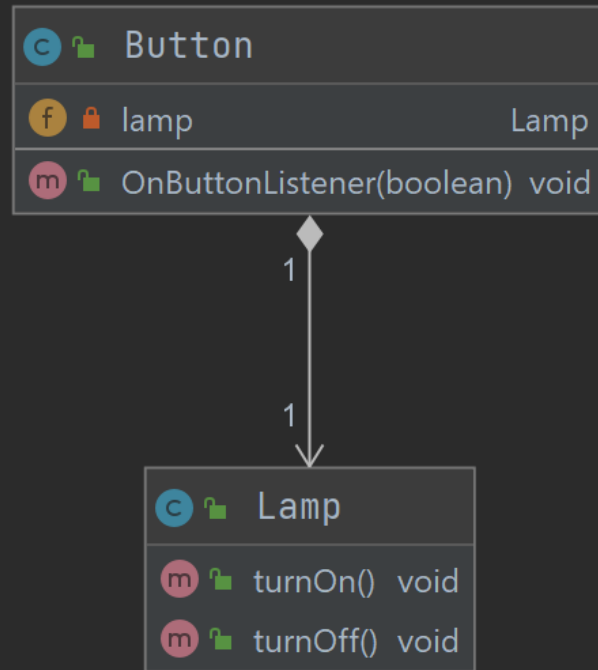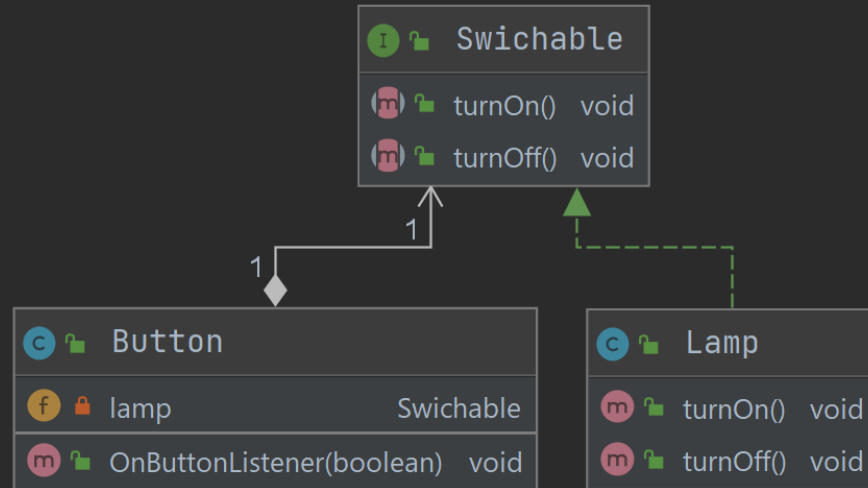
Layering

Layering

Layering

Button

f 🔒	lamp	Lamp
m 🔒	OnButtonListener(boolean)	void

1

1

Lamp

m 🔒	turnOn()	void
m 🔒	turnOff()	void

Layering

Final work

- Real app

- UI

- Class diagram

- Code

- Demonstrate that all the requirements are covered in a short video

- code + diagram + video in a Github repository

- Clean code best practices and SOLID

- 3 design patters. Describe shortly each one in the readme.md file

- Understandable commits

- Verifiable weekly progress (One commit, 3 minutes demo)

Class Diagram App

En general se desea implementar un graficador de clases, (no generar diagramas basados en código) si no, solamente graficar como se podría hacer en "https://cacoo.com/"

Requerimientos:

- Se debe poder graficar una clase.
- Se debe poder graficar una interface
- Se debe poder agregar relaciones de herencia
- Se debe poder agregar relaciones de dependencia
- Se debe poder hacer "drag" de las clases e interfaces
- Mínimamente se debe visualizar el nombre de la clase
- Se debe poder cambiar el nombre de las clases
- Debe estar implementado en Java o C#
- Debe tener un pequeño menú para poder agregar todo lo anterior al diagrama