The Dependency Inversion Principle

The Dependency Inversion Principle

A. High-level modules should not depend on low-level modules. Both should depend on abstractions.

B. Abstractions should not depend on details. Details should depend upon abstractions.

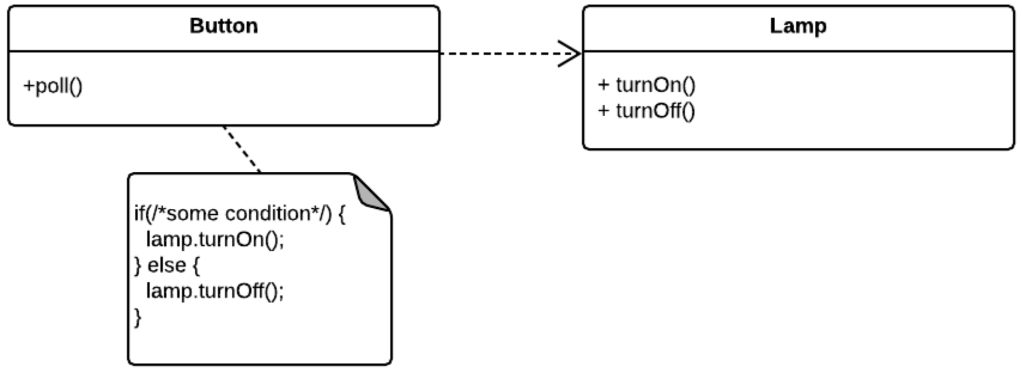
Martin, pg. 154

What Is It?

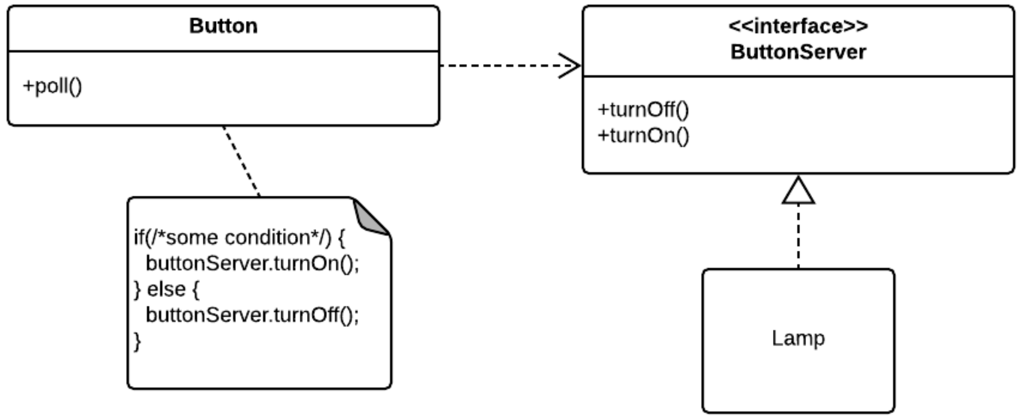
- A way to decouple classes or components by making it so neither knows about the other
  - Why is it good for neither to know about the other?
- A way to reverse the order of layer dependencies
- Can be extended to eliminate layer dependencies

When Do We Use It?

- When not using it will make a class that could be generally useable, less useable or useable in fewer cases
  - Button / Lamp example from Martin Book (pgs. 158- 159) (with improved notation)



```
if(/*some condition*/) {
  lamp.turnOn();
} else {
  lamp.turnOff();
}
```



```
if(/*some condition*/) {
  buttonServer.turnOn();
} else {
  buttonServer.turnOff();
}
```

Bad.
Why is this bad?

Better.
Why is this better?

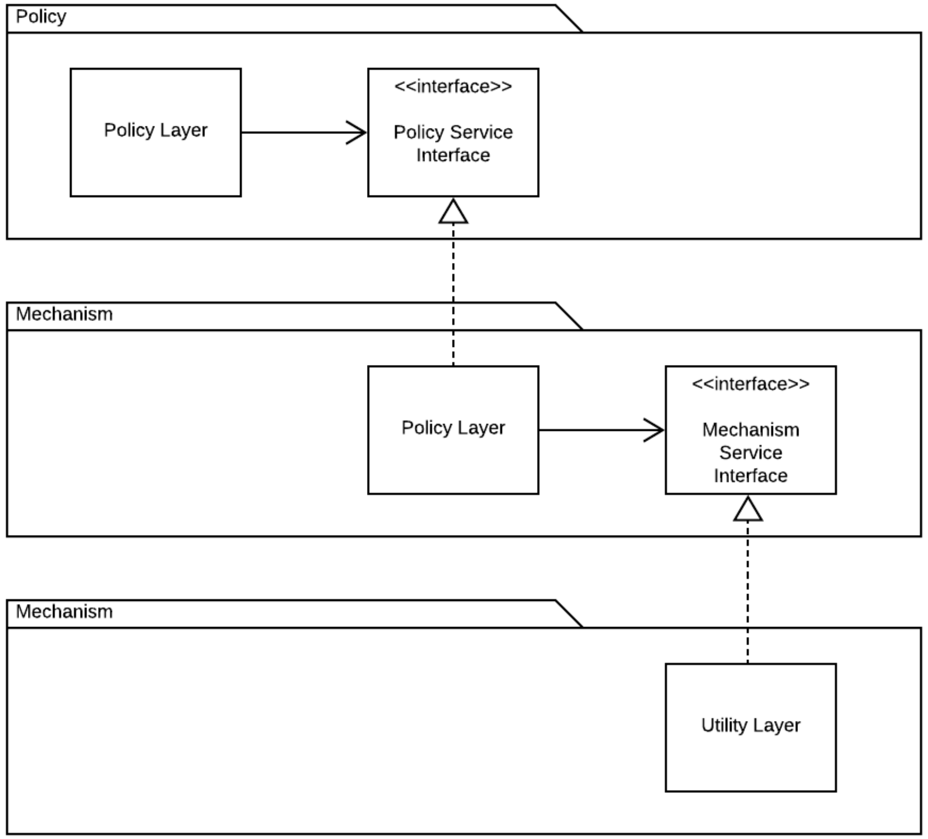
Code to the Interface

- When we use the principle to simply break the dependency between two classes, the name “dependency inversion” doesn’t seem very clear
- **General principle:** Code to the interface, not the implementation
- The name “dependency inversion” makes more sense when talking about layers
    - If button and lamp were in different layers, the bad example has the “button” layer depending on the “lamp” layer
    - The good example reverses that (assuming ButtonServer is in the “button” layer)
        - Now the “lamp” layer depends on the “button” layer (but not the button class!)

How Does this Apply to Layers?

- Normally, dependency flows from higher layers in an architecture to lower layers
- Dependency inversion provides a way to reverse that, so higher level layers are not directly affected by changes in lower layers
    - Lower layers then become dependent on interfaces provided by higher layers
    - Makes higher layers more reusable and lower layers less reusable
- Dependency can go either direction, depending on which layer contains the interface and which layer depends on it
- Either way, the interface provides a level of indirection that prevents direct dependency between classes in the two layers

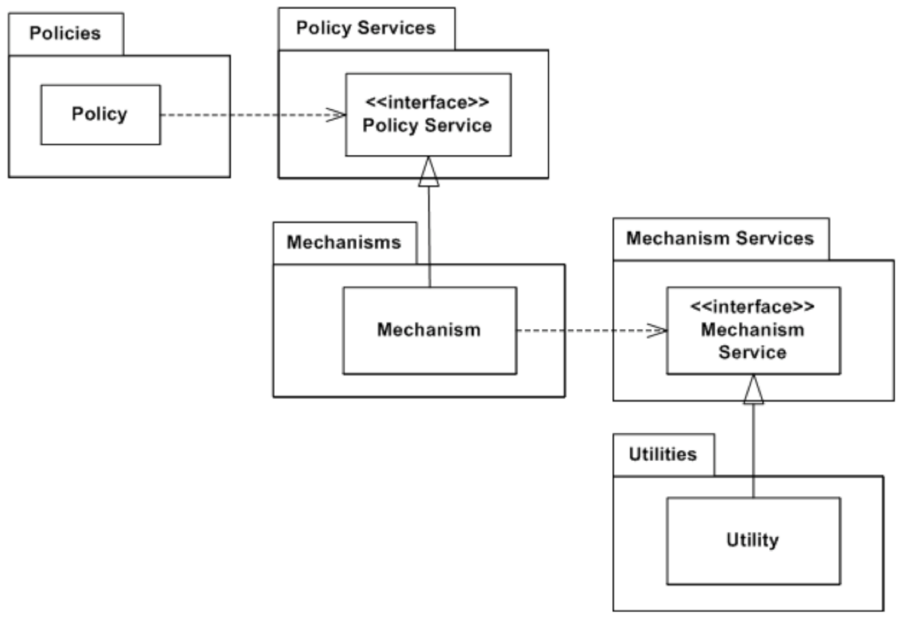
Inverted Layers Example



Martin, pg. 156

Maximizing Layer Reuse

- Can maximize layer reuse by moving the interfaces into separate packages that are not in any layer



- Now no layer depends on any other, so all can be reused independently of all others