

Software Engineering Design & Construction

Dr. Michael Eichberg
Fachgebiet Softwaretechnik
Technische Universität Darmstadt

Dependency-Inversion Principle

Dependency-Inversion Principle

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

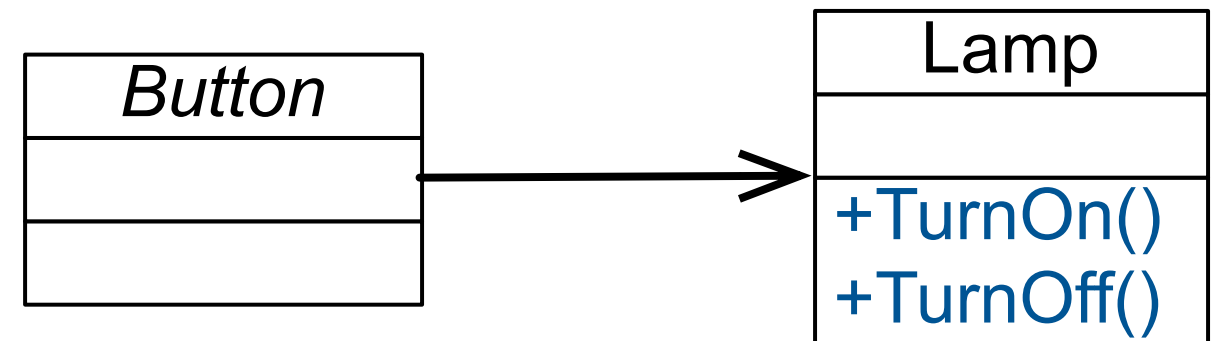
*Abstractions should not depend on details. Details should
depend on abstractions.*

–Agile Software Development; Robert C. Martin; Prentice Hall, 2003

Introduction by Example

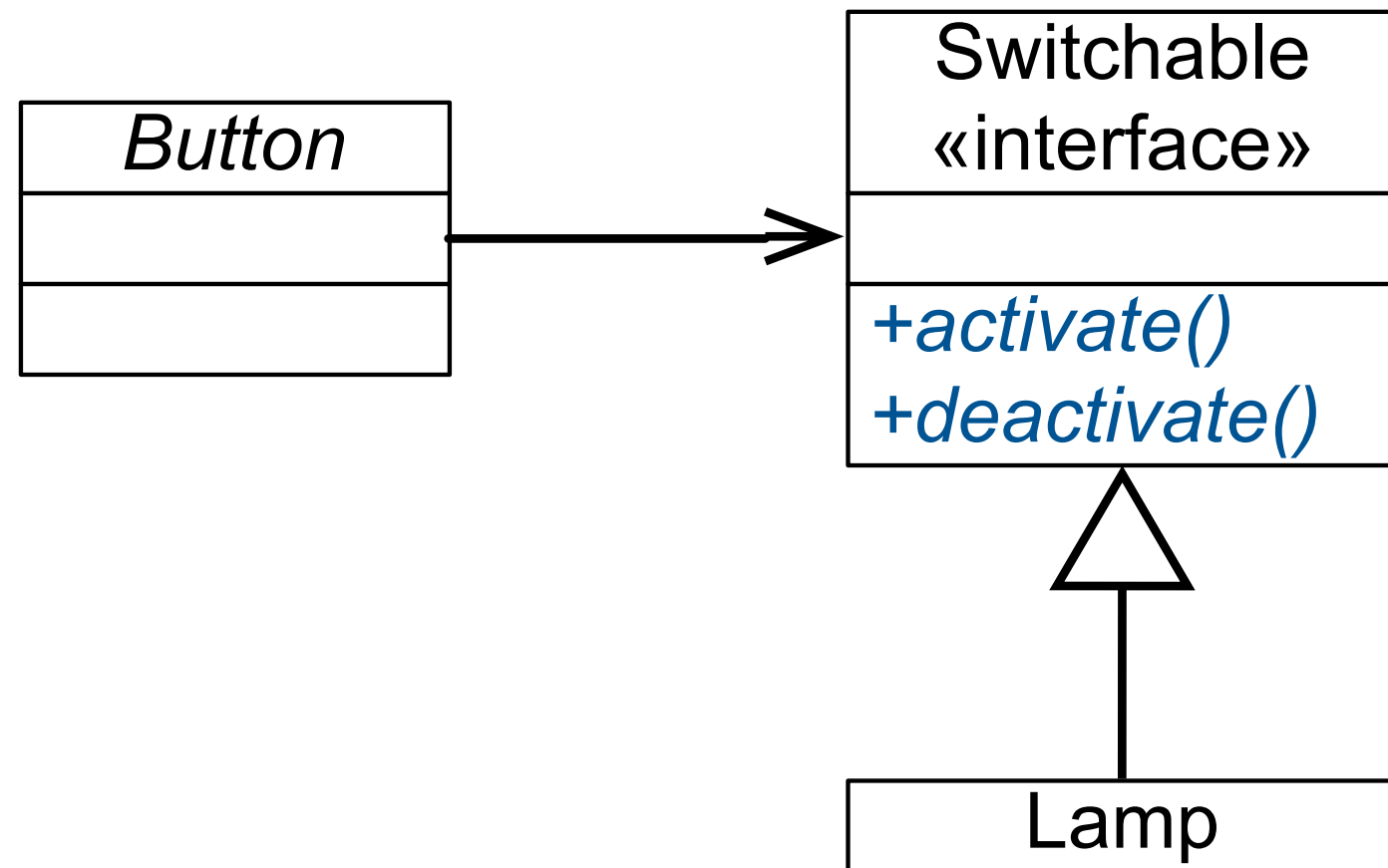
A Small Design Exercise

- Behavior of **Button**:
 - The button is capable of “sensing” whether it has been activated/deactivated by the user.
 - Once a change is detected, it turns the Lamp on, respectively off.



Issues?

A Dependency-Inversion Principle Compliant Solution



The Rationale behind the Dependency-Inversion Principle

- High-level, low-level modules. Good software designs are structured into modules.
- **High-level modules** contain the important policy decisions and business models of an application – The identity of the application.
- **Low-level modules** contain detailed implementations of individual mechanisms needed to realize the policy.

The Rationale behind the Dependency-Inversion Principle

- High-level, low-level modules. Good software should be structured into modules

High-level Policy

The abstraction that underlies the application; the truth that does not vary when details are changed; the system inside the system; the metaphor.

Low-level modules contain detailed implementations of individual mechanisms needed to realize the policy.

*„...all well-structured object-oriented architectures **have clearly defined layers**, with each layer providing some coherent set of services **through a well-defined and controlled interface...**“*

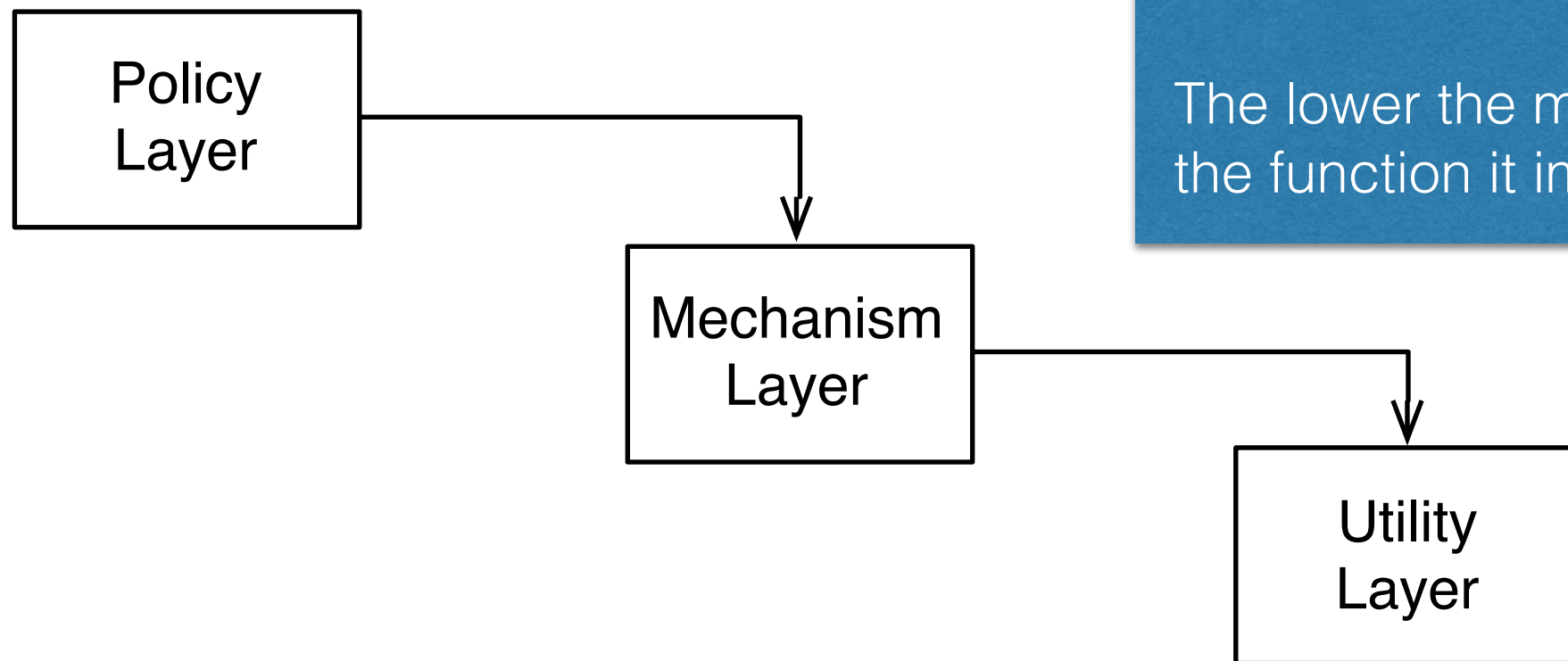
-Grady Booch

*„...all well-structured object-oriented architectures **have clearly defined layers**, with each layer providing some coherent set of services **through a well-defined and controlled interface**...“*

An Interpretation

The higher the module is positioned in a layered architecture, the more general the function it implements.

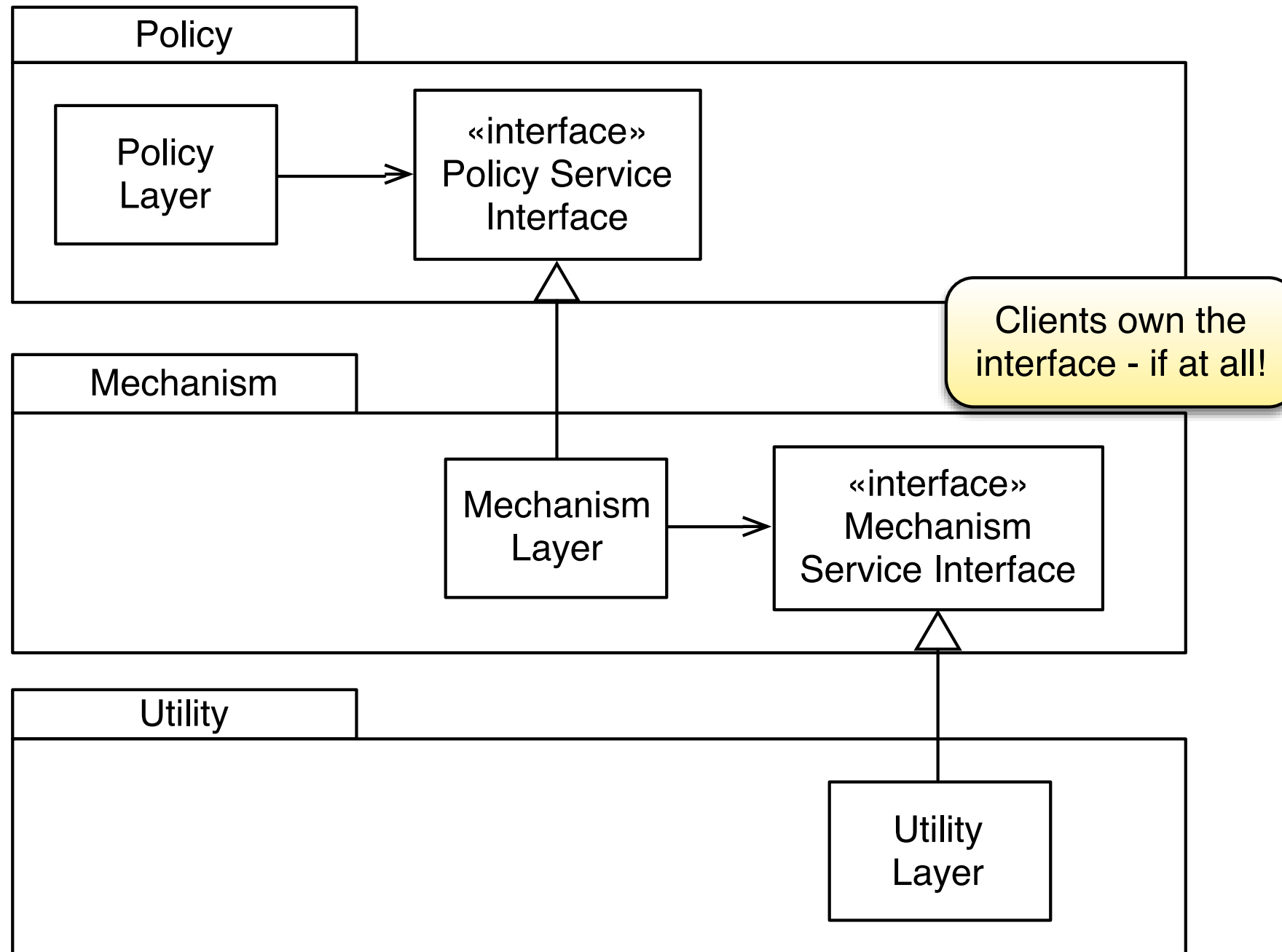
The lower the module, the more detailed the function it implements.



-Grady Booch

Layers and Dependencies

Inverted Layer Dependencies



Naïve Heuristic for Ensuring DIP

DO NOT DEPEND ON A CONCRETE CLASS.

All relationships in a program should terminate on an abstract class or an interface.

- No class should hold a reference to a concrete class.
- No class should derive from a concrete class.
- No method should override an implemented method of any of its base classes.

Dependency-Inversion Principle

- Traditional structural programming creates a dependency structure in which policies depend on details.
High level modules should not depend on low level modules. Both should depend on abstractions.
(Policies become vulnerable to changes in the details.)
- Object-orientation enables to invert the dependency:
Abstractions should not depend on details. Details should depend on abstractions.
 - Policy and details depend on abstractions.
 - Service interfaces are owned by their clients.
 - Inversion of dependency is the hallmark of good object-oriented design.
(Implies an inversion of interface ownership.)

–Agile Software Development; Robert C. Martin; Prentice Hall, 2003