S.O.L.I.D.

The Benefits and Potential of Using SOLID Principles

- **S** — Single Responsibility
- **O** — Open/Closed
- **L** — Liskov substitution
- **I** — Interface Segregation
- **D** — Dependency Inversion

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#java-advanced

Table of Contents

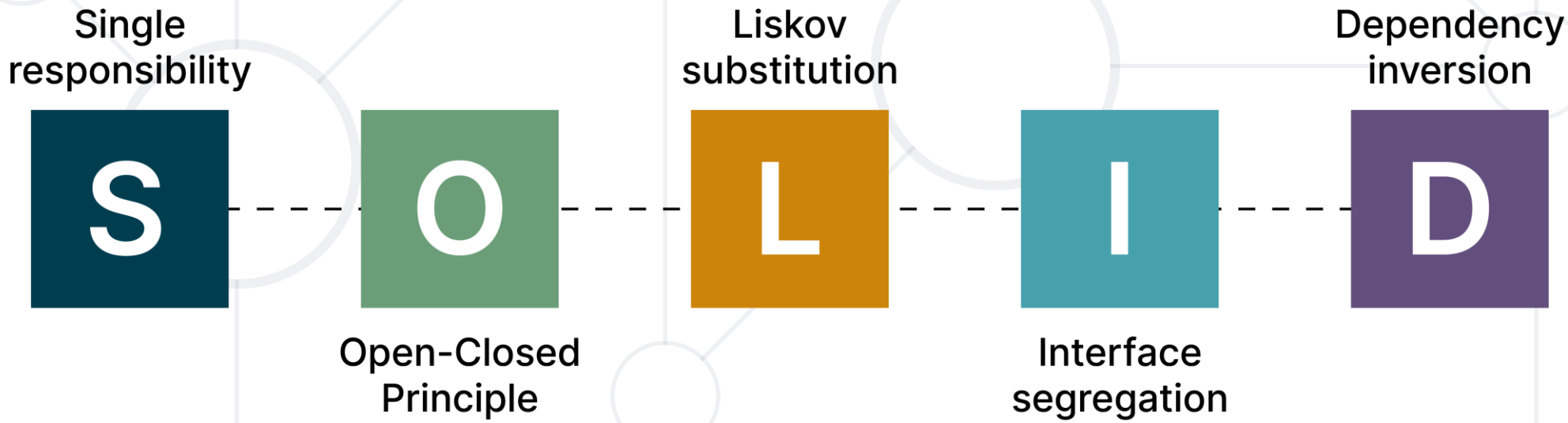
SOLID Principles

S.O.L.I.D.

- **S** – **Single responsibility principle** – class should only have one responsibility
- **O** – **Open–closed principle** – open for extension, but closed for modification
- **L** – **Liskov substitution principle** – objects should be replaceable with instances of their subtypes without altering the correctness of that program

S.O.L.I.D.

- I – Interface segregation principle – many specific interfaces are better than one general interface
- D – Dependency inversion principle – one should depend upon abstractions, not concretions



Single Responsibility

Single Responsibility Principle

Software University

- A class should **have only one responsibility**
  - Reduces **dependency** complexity
  - Each additional responsibility is an **axis to change the class**

```
public class HeroSettings {
   public static void changeName(Hero hero) {
      // Grant option to change
   }
}
```

Single Responsibility Principle

- Still, classes **can have multiple methods**
  - Each method should have a **single functionality** part of the class responsibility

```
public class HeroSettings {
   public static void changeName(Hero hero) {
      // Grant option to change name
   }
   public static void selectRole(Hero hero) {
      // Grant option to select role
   }
}
```

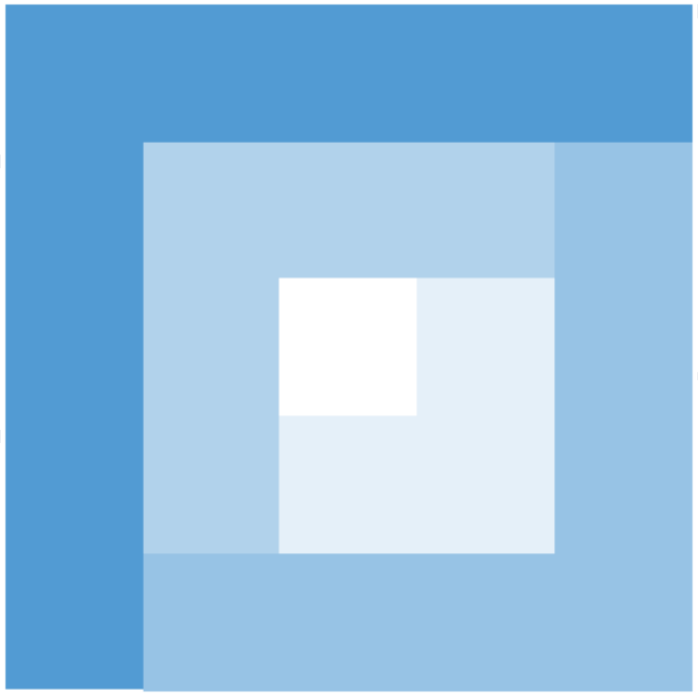
Open / Closed

What is Open/Closed?

- Software entities (classes, modules, functions, etc.) should be
  - **open** for **extension**
  - **closed** for **modification**
- **Design** the code in a way that **new** functionality can be added with **minimum changes** in the **existing** code

Extensibility

- Implementation takes **future growth** into **consideration**
- New or **modified functionality** affects little or not at all the internal structure and data flow of the system



Reusability

- Software reusability refers to **design features** of a software **element** that enhance its **suitability** for **reuse**
- Modularity
- Low coupling
- High cohesion
- Coupling and Cohesion

OCP – Violations

- **Cascading changes** through modules
- Each change **requires re-testing**
- Logic **depends** on conditional statements

OCP – Solutions

- Inheritance / Abstraction
- Inheritance / Template Method pattern
- Composition / Strategy patterns

Liskov Substitution

What is Liskov Substitution?

Software University

- Derived types must be **completely substitutable** for their base types
- Reference to the base class can be replaced with a derived class without affecting the functionality of the program module
- Derived classes extend without replacing the functionality of old classes

LSP Relationship

- OOP Inheritance

```
Student IS-A Person
```

- Plus LSP

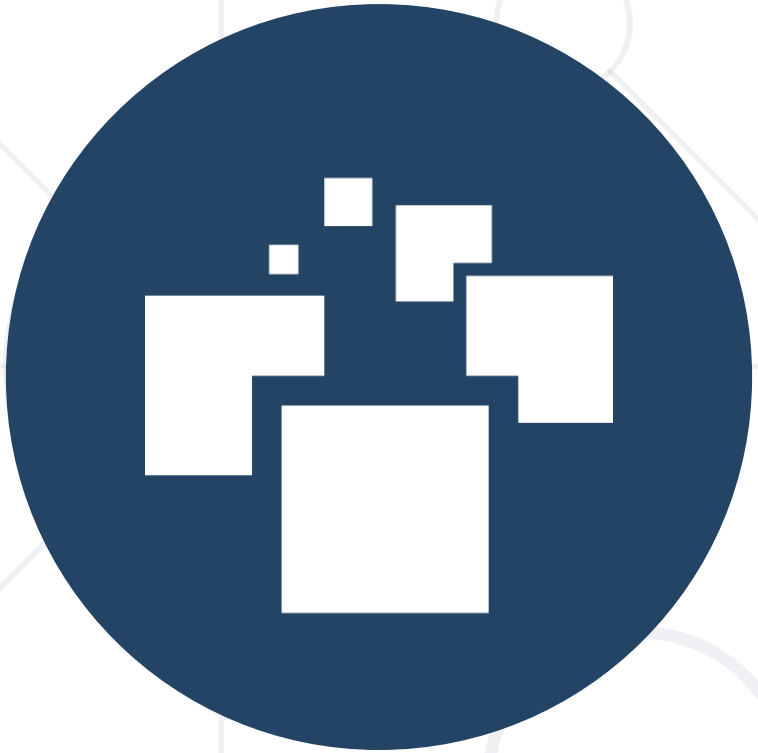
```
Student IS-SUBSTITUTED-FOR Person
```

OCP vs LSP

- Liskov Substitution Principle is just an **extension** of the Open-Closed Principle
- We must make sure that new derived classes are extending the base classes **without changing** their **behavior**

LSP – Violations and Solutions

- Violations
  - Type Checking
  - Overridden methods say "I am not implemented"
  - Base class depends on its subtypes
- Solutions
  - Refactoring in the **base class**



Interface Segregation

ISP – Interface Segregation Principle

- Clients should **not** be **forced** to **depend** on methods they do **not use**
- Segregate interfaces
  - Prefer **small**, **cohesive** interfaces
  - Divide "**fat**" interfaces into "**role**" interfaces

Fat Interfaces

- Classes whose interfaces are not cohesive have "fat" interfaces

```
public interface Worker {
  void work();
  void sleep();
}
```

Class Employee is OK

```
public class Robot implements Worker {
  public void work() {}
  public void sleep() {
    throw new UnsupportedOperationException();
  }
}
```

"Fat" Interfaces

Software University

- Having "**fat**" interfaces:
  - Classes have methods they do not use
  - Increased **coupling**
  - Reduced flexibility
  - Reduced maintainability

How to ISP?

- Solutions to broken ISP
  - **Small** interfaces
  - **Cohesive** interfaces
  - Let the client **define** interfaces – "role" interfaces

Cohesive Interfaces

- Small and Cohesive "**Role**" Interfaces

```
public interface Worker {
   void work();
}
```

```
public interface Sleeper {
   void sleep();
}
```

```
public class Robot implements Worker {
   void work() {
      // Do some work…
   }
}
```

Dependency Inversion

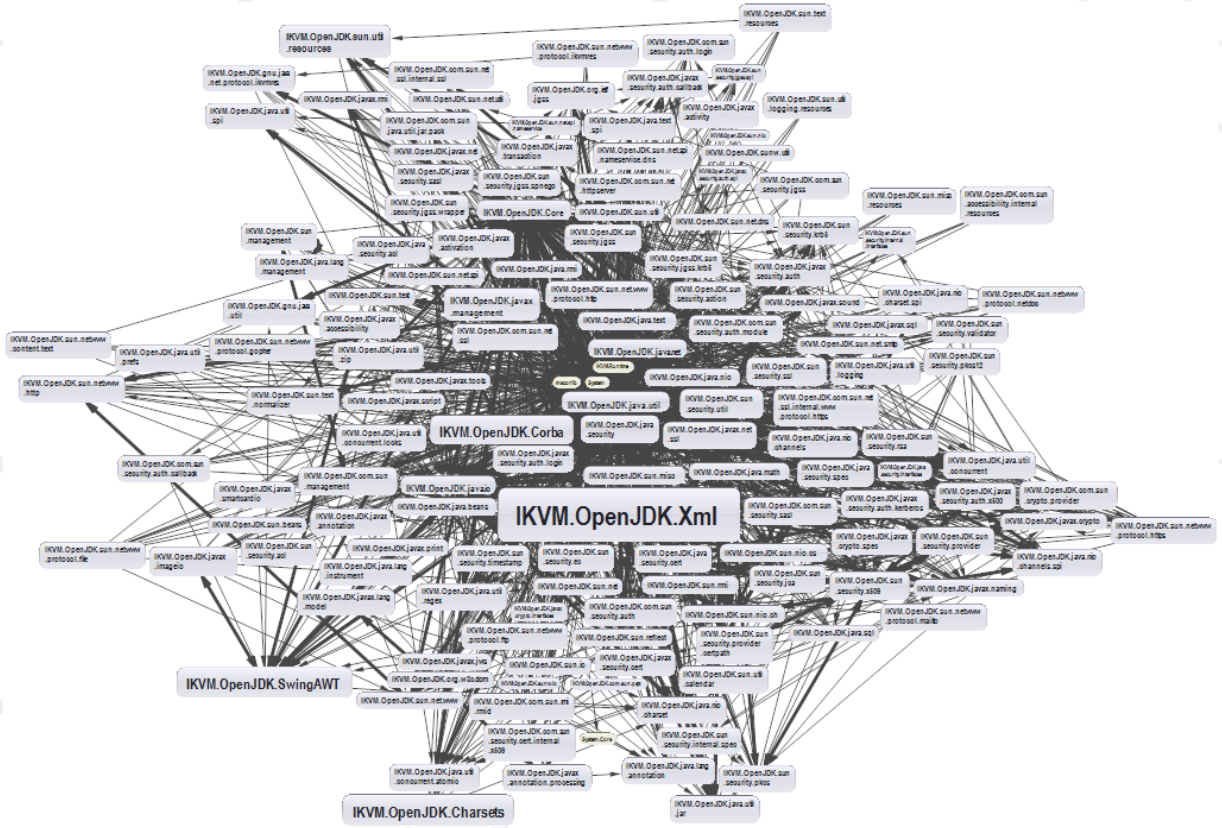
Dependency Inversion Principle (DIP)

Software University

- 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
- Goal: **decoupling between modules** through abstractions

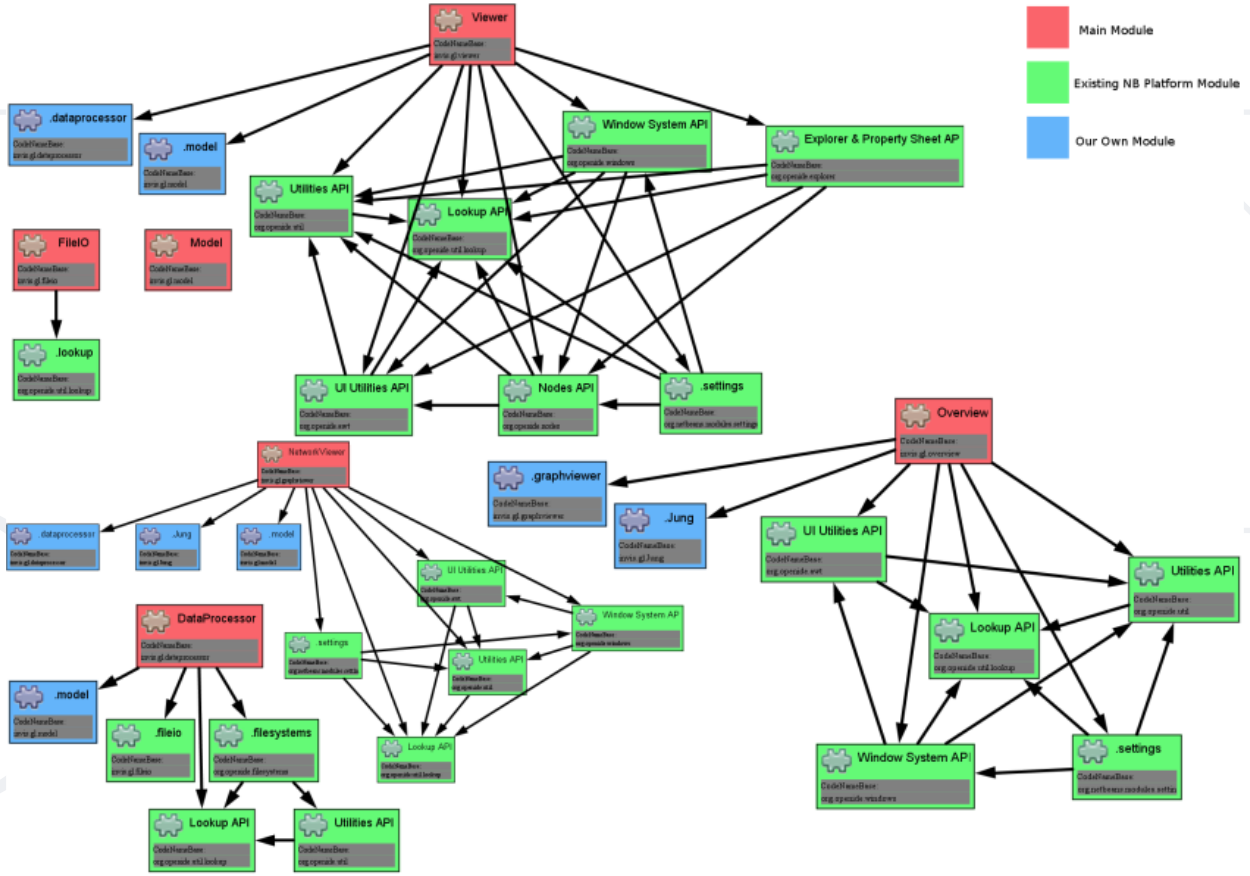
Dependencies and Coupling (1)

- What happens when modules **depend directly** on **other modules**



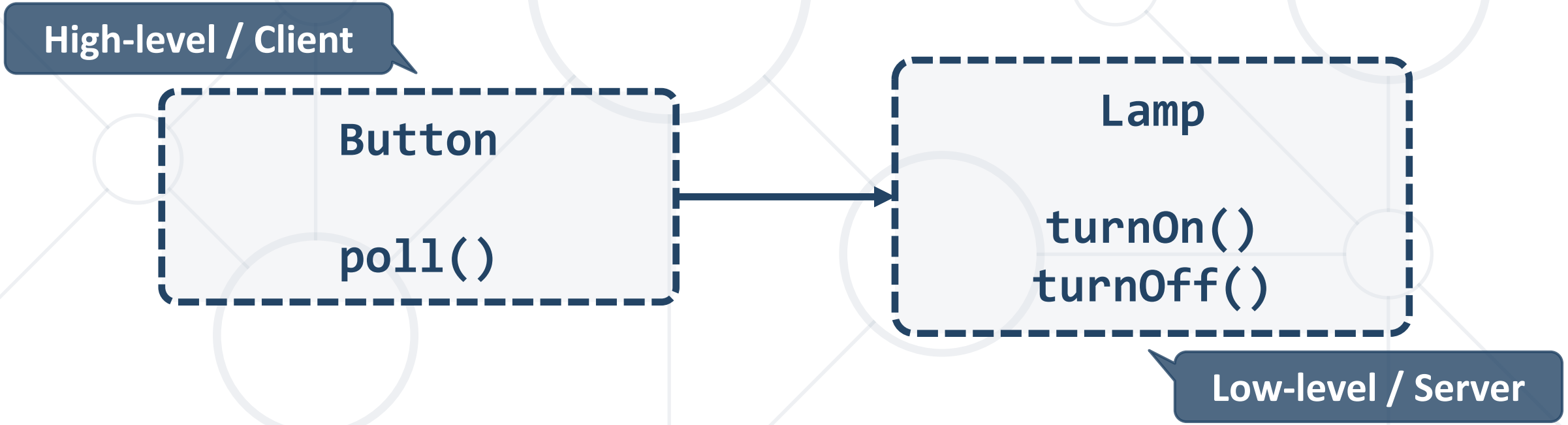
Dependencies and Coupling (2)

- The goal is to **depend on abstractions**



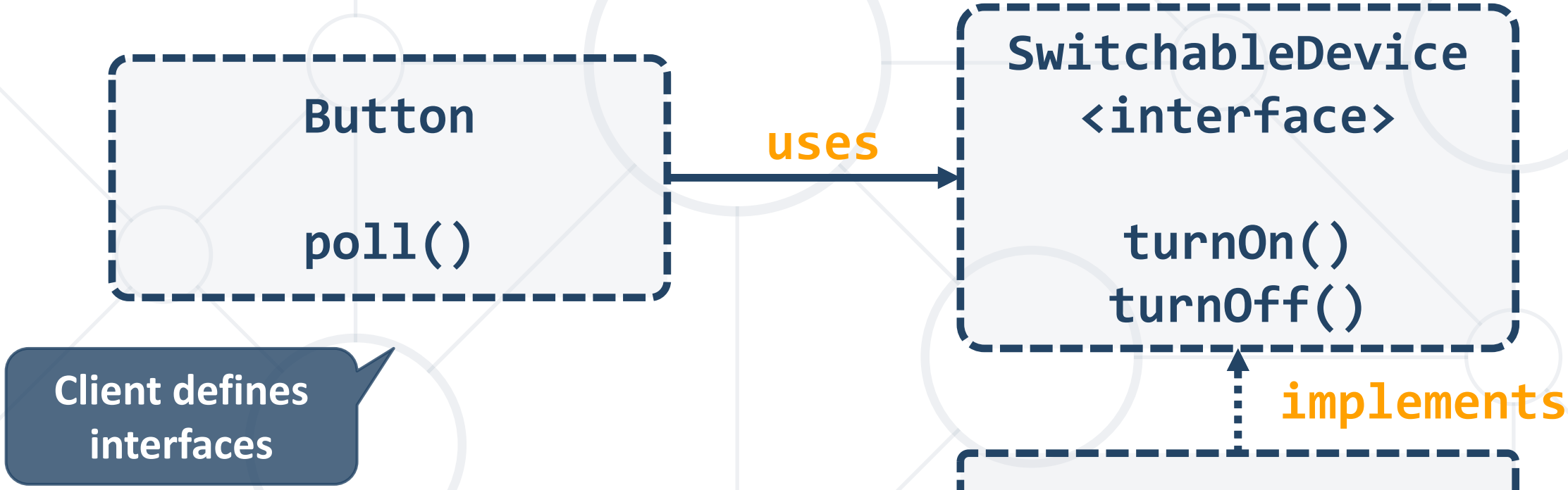
The Problem

- Button → Lamp Example – **Robert Martin**
- Button **depends on** Lamp



Dependency Inversion Solution

- Find the abstraction independent of details



Dependency Examples

- A **dependency** is an external component / system:
  - Framework
  - Third party library
  - Database
  - File system
  - Email
  - Web service
  - System resource (e.g. clock)
  - Configuration
  - The `new` keyword
  - Static method
  - Global function
  - Random generator
  - System.in / System.out

How to DIP? (1)

- **Constructor injection** - dependencies are passed through **constructors**
  - **Pros**
    - Classes **self-documenting** requirements
    - Works well without a container
    - Always **valid state**
  - **Cons**
    - Many parameters
    - Some methods may not need everything

Constructor Injection – Example

```
public class Copy {
  private Reader reader;
  private Writer writer;
  public Copy(Reader reader, Writer writer) {
    this.reader = reader;
    this.writer = writer;
  }
  public void copyAll() {}
}
```

How to DIP? (2)

- **Setter Injection** - dependencies are passed through **setters**
  - **Pros**
    - Can be changed anytime
    - Very **flexible**
  - **Cons**
    - Possible **invalid state** of the object
    - Less intuitive

Setter Injection – Example

```
public class Copy {
  private Reader reader;
  private Writer writer;
  public void setReader(Reader reader) {}
  public void setWriter(Writer writer) {}
  public void copyAll() {}
}
```

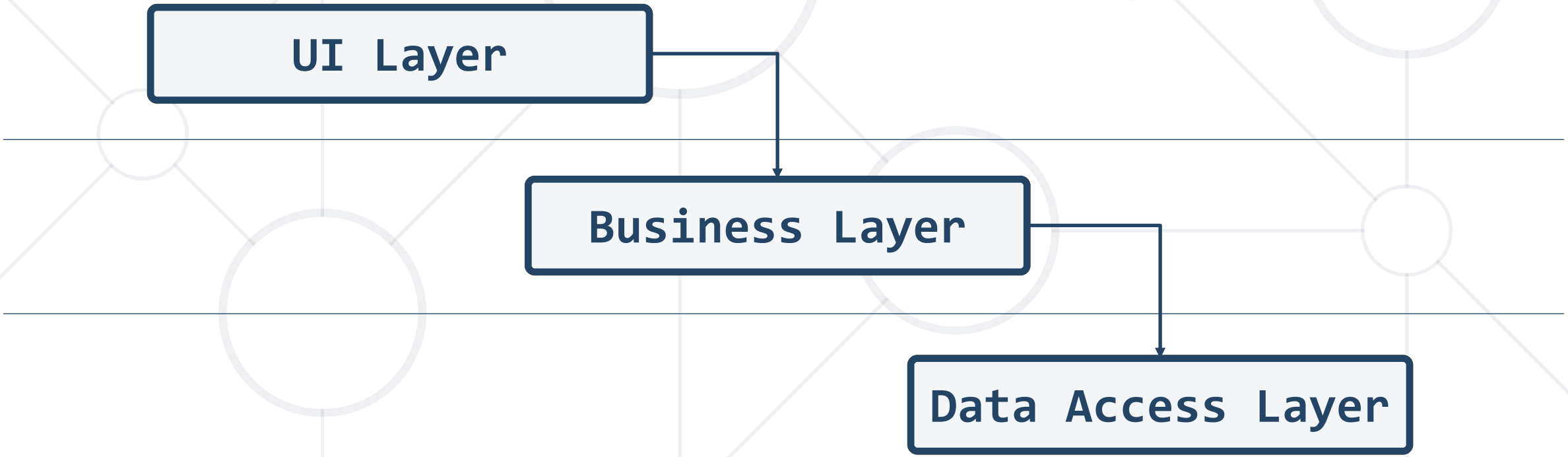
How to DIP? (3)

- **Parameter injection** - dependencies are passed through **method parameters**
  - **Pros**
    - No change in rest of the class
    - Very flexible
  - **Cons**
    - Many parameters
    - Breaks the method signature

```
public class Copy {
  public void copyAll(Reader reader, Writer writer) {}
}
```

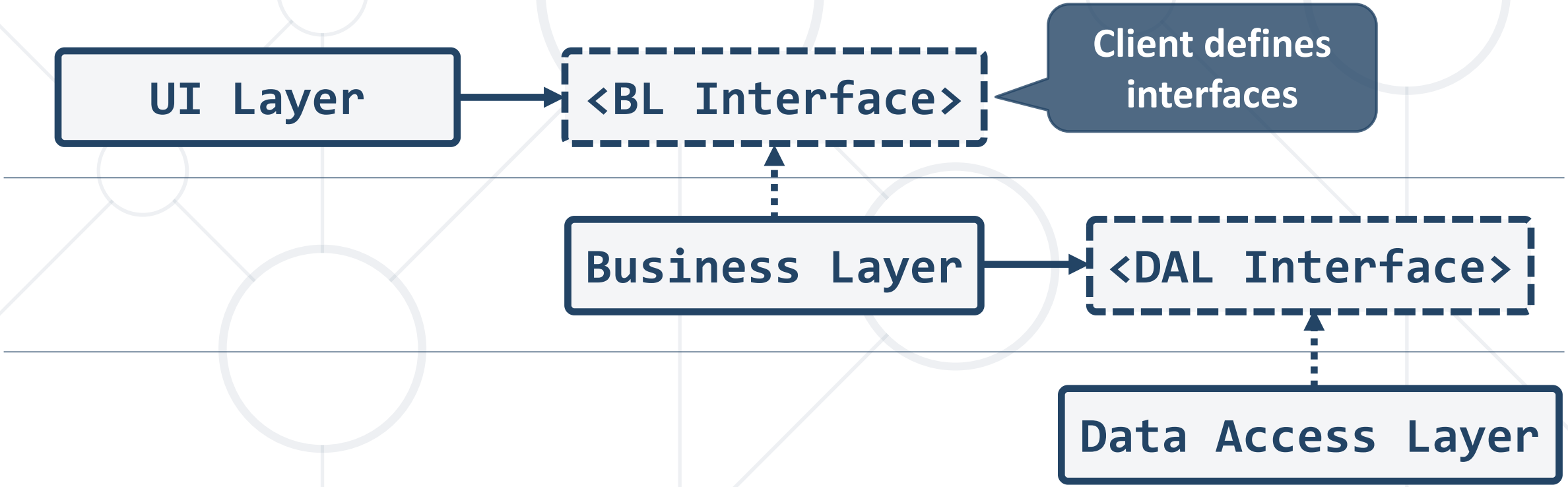
Layering (1)

- Traditional programming
  - **High-level** modules use **low-level** modules



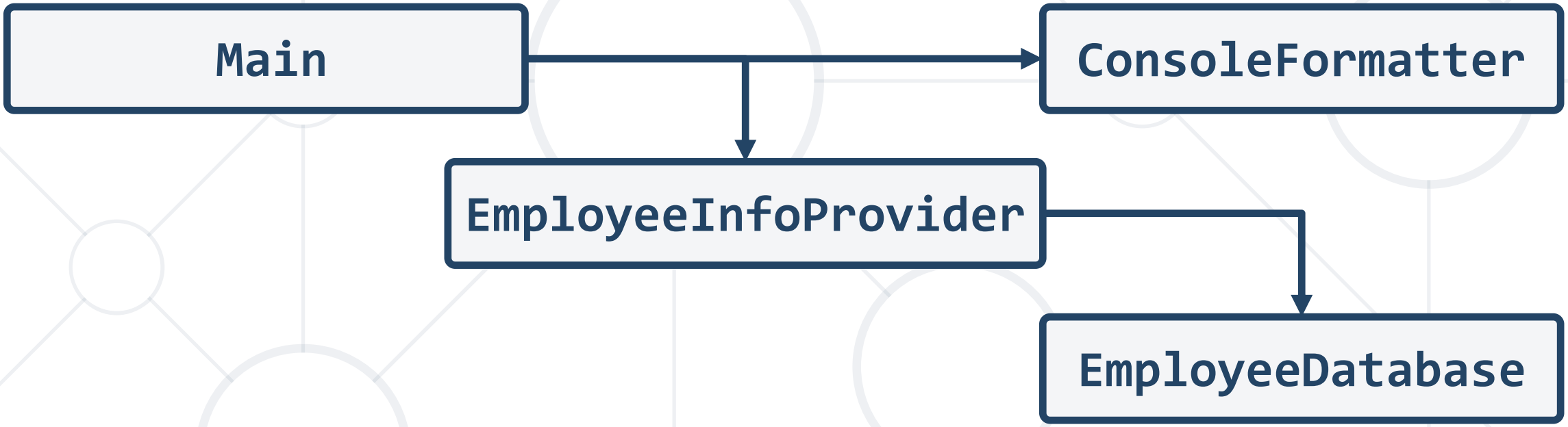
Layering (2)

- Dependency Inversion Layering
  - **High** and **low-level** modules **depend on abstractions**



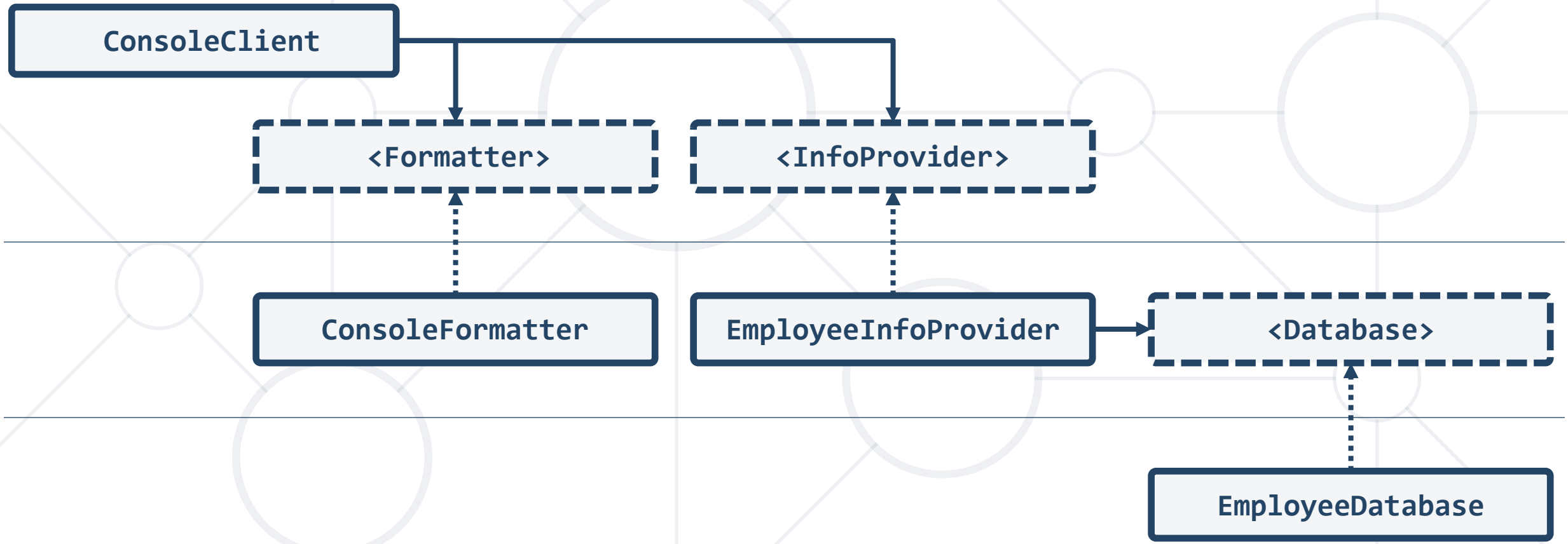
Problem: Employee Info

- You are given some classes



- Refactor the code so that it conforms to DIP

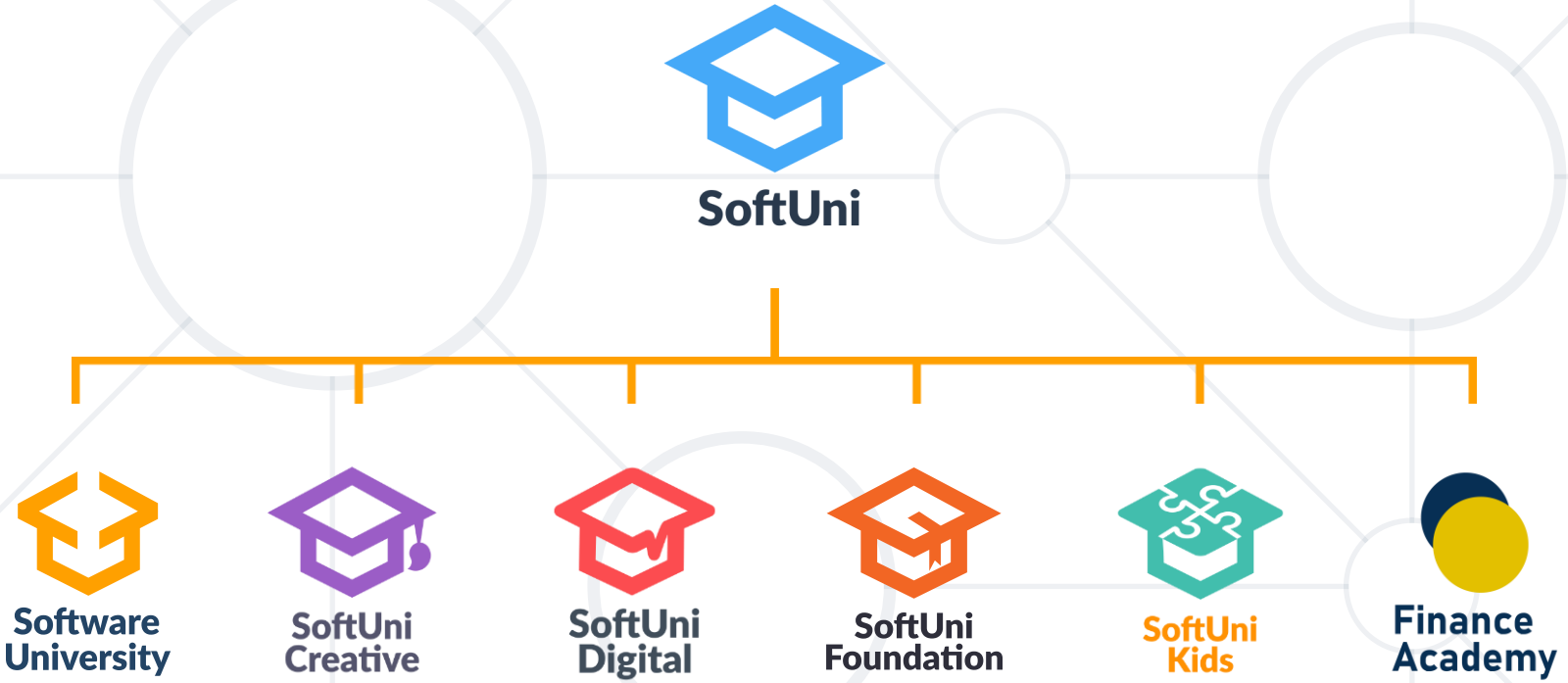
Solution: Employee Info



Summary

- **SOLID** principles make the software:
  - Understandable
  - Flexible
  - Maintainable

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

Postbank
Решения за твоето утре

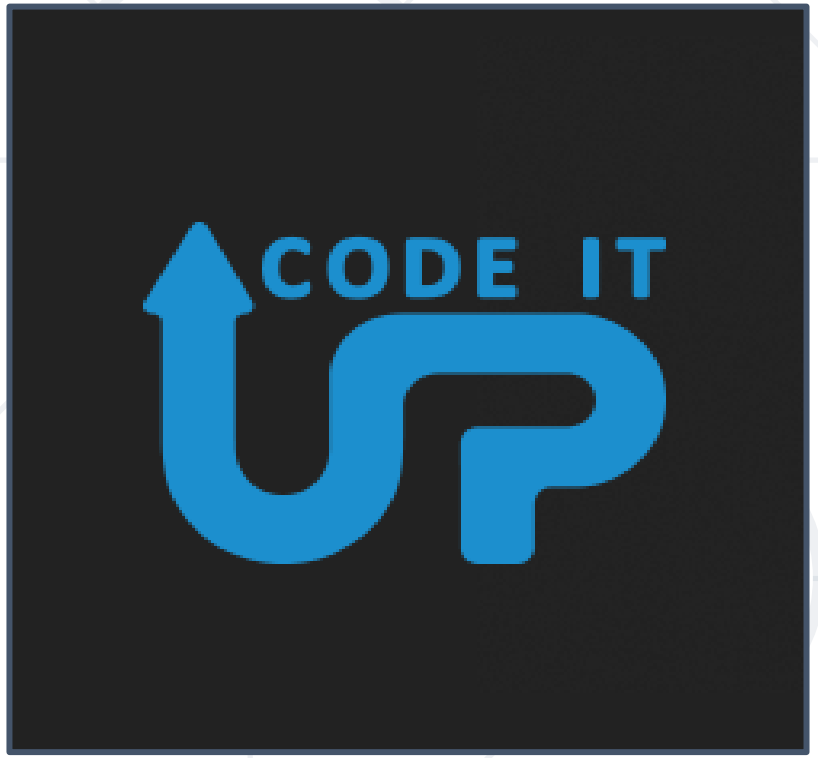
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg