Dependency Injection with Gin and Dagger2

Dr. Lofi Dewanto
https://lofidewanto.blogspot.de

Agenda

- Dependency Injection
- Gin and Dagger2

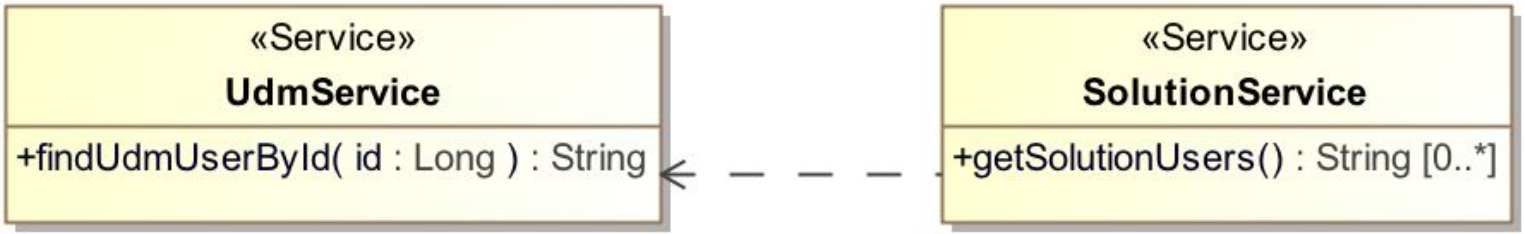
Dependency Injection

General Problem: Class Dependency

- General problem: **class dependency**
- Implementation of class A has a **direct dependency** to implementation class B.
- Implementation class B **cannot be exchanged** for other implementation.
- **Unit testing for class A impossible**, as soon as we have to mock the implementation of class B.

Simple Example

- SolutionService implementation **depends** on UdmService (User Data Management) implementation.
- UdmService implementation has **an access to database** and it should not be tested if you want to test SolutionService implementation.



Goal: Classes Independence

- Goal: **classes are not tightly coupled** and we can exchange the implementation of the classes.
- Solution: **Dependency Injection**
  - Do not use “new”
    - Create a direct dependency with the created object
    - Not testable
  - No need to implement Factory class manually
  - Centralize the dependencies of all classes

Dependency Injection

Wikipedia:
In software engineering, **dependency injection (DI)** is a technique whereby one object (or static method) supplies the dependencies of another object. A dependency is an object that can be used (a service). An injection is the passing of a dependency to a dependent object (a client) that would use it. The service is made part of the client's state.[1] Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.

Dependency Injection and Inversion of Control

- Special case of Inversion of Control (IoC)
  A software architecture with this design inverts control as compared to traditional procedural programming: in traditional programming, the custom code that expresses the purpose of the program calls into reusable libraries to take care of generic tasks, but with inversion of control, it is the framework that calls into the custom, or task-specific, code.

- **Don’t call me, I call you - Hollywood Principle**

Solution and Demo

Demo <SolutionService, UdmService>

- Layer 0: primitive with **new**
- Layer 1: primitive with **Factory** class
- Layer 3: DI **without framework**
- Layer 4: DI **with framework**

https://github.com/lofidewanto/dep-injection-examples

Pay attention: advantages and disadvantages of each layer!

Gin and Dagger2

Criteria for DI Framework

- Java **Code generation** (Gin, Dagger2)
- Standard with **JSR 330** (Guice, Gin, Dagger2, JBoss Weld, Apache OpenWebBeans, Caucho Candi)
- Can be used in different Java environments JavaSE, JavaEE, Web browser GWT, Android (Dagger2)

- Java **Reflection** (Spring, JBoss Weld, Apache OpenWebBeans, Caucho Candi, Guice)
- **Non JSR 330** standard (Spring in default mode with @Autowired instead @Inject)
- Only usable in JavaSE and JavaEE (Spring, JBoss Weld, Apache OpenWebBeans, Caucho Candi, Guice)

Gin and Dagger2

- Gin: https://github.com/gwtplus/google-gin/wiki
- Dagger2: https://google.github.io/dagger

Dependency Injection:Task

- Implement the *SolutionService* and *UdmService* with **Gin** or **Dagger2**

- Result: https://github.com/lofidewanto/dep-injection-examples/tree/master/di-inject-constructor-gwt-gin

References

- https://en.wikipedia.org/wiki/Dependency_injection
- https://github.com/google/guice/wiki/Motivation