spring®

How it helps to keep your code clean.

Symptoms of poor design

- **Rigidity** – Hard to change
- **Fragility** – Easy to break
- **Immobility** – Hard to reuse
- **Viscosity** – Easier to hack than doing it right
- **Needless complexity** – Over engineered
- **Needless repetition** - DRY
- **Opacity** – High WTF / min rate

SOLID principles

- **S**ingle **R**esponsibility **P**rinciple
- **O**pen **C**losed **P**rinciple
- **L**iskov **S**ubstitution **P**rinciple
- **I**nterface **S**egregation **P**rinciple
- **D**ependency **I**nversion **P**rinciple

Single Responsibility

- A class should have only one reason to change

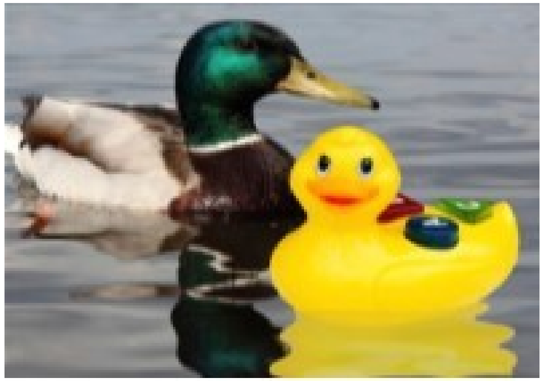
  Separate your classes

Open Closed Principle

- Open for **extension**
  closed for **modification**

Liskov Substitution Principle

- Each class must be able to be substituted by their sub-classes.



if it looks like a duck,
quacks like a duck,
but needs batteries -
you probably have the
wrong abstraction

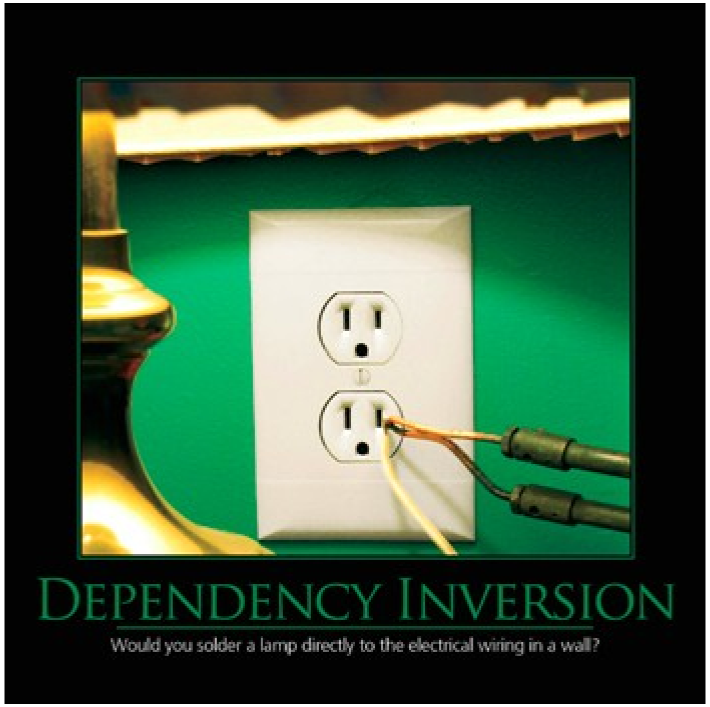
Interface Segregation Principle

- Clients should not be forced to depend upon interfaces that they do not use.

Not bad!
DRINK + EAT
EAT
DRINK

Dependency Inversion Principle

- High-level modules should not depend on low-level modules. Both should depend on abstractions (e.g. interfaces).
- Abstractions should not depend on details. Details (concrete implementations) should depend on abstractions.


DEPENDENCY INVERSION
Would you solder a lamp directly to the electrical wiring in a wall?

What about instantiation?

Inversion of Control

- Vanilla Java & Design Patterns
  - Factory
  - Service Locator
  - Template Method
  - Strategy

- Framework
  - Dependency Injection

+ Less boilerplate

What is Spring?

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.

A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

Spring Core

- Dependency Injection
- Data binding
- Validation
- Type conversion
- AOP
- Events
- Resources
- I18n
- SpEL

Dependency injection

- Constructor injection

```java
@Service
public class MyService {

    private final Dependency dependency;

    @Autowired
    public MyService(final Dependency dependency) {
        this.dependency = dependency;
    }

    public void doSomething() {
        dependency.doSomething();
    }
}
```

- Field injection

```java
@Service
public class MyService {

    @Autowired
    private Dependency dependency;

    public void doSomething() {
        dependency.doSomething();
    }
}
```

- Setter injection

```java
@Service
public class MyService {

    private Dependency dependency;

    @Autowired
    public void setDependency(final Dependency dependency) {
        this.dependency = dependency;
    }

    public void doSomething() {
        dependency.doSomething();
    }
}
```

IoC container & Beans

- Application Context
- Managed Beans
- Configuration (annotation, Java, XML)

Demo

- Dependency Injection
- Values

Unit Testing

- MockitoExtension
- Inject mocks
- Avoid Test Unfriendly Constructs (TUC) at Test Unfriendly Features (TUF)

Demo

- Unit Testing

Type Conversion

- ConversionService
- Implicit type conversion at RestController-s
- Use it with care

Demo

- Conversion

AOP

- Orthogonal features e.g. Logging, Error handling, Sanitizing input, etc… Can be added placed upon your methods in a non-invasive way.
- Advices
  - Before
  - AfterReturn
  - Around
- Pointcuts
  - Annotation
  - Class/method name
  - Argument list

Demo

- AOP
- Integration testing

Coming Soon

- Advanced Spring
  - Spring Boot
  - Spring Data (Cosmos)
  - Spring Functional Web Framework