软件工程

从OO到Mocking测试

Spring 2022, SWUFE

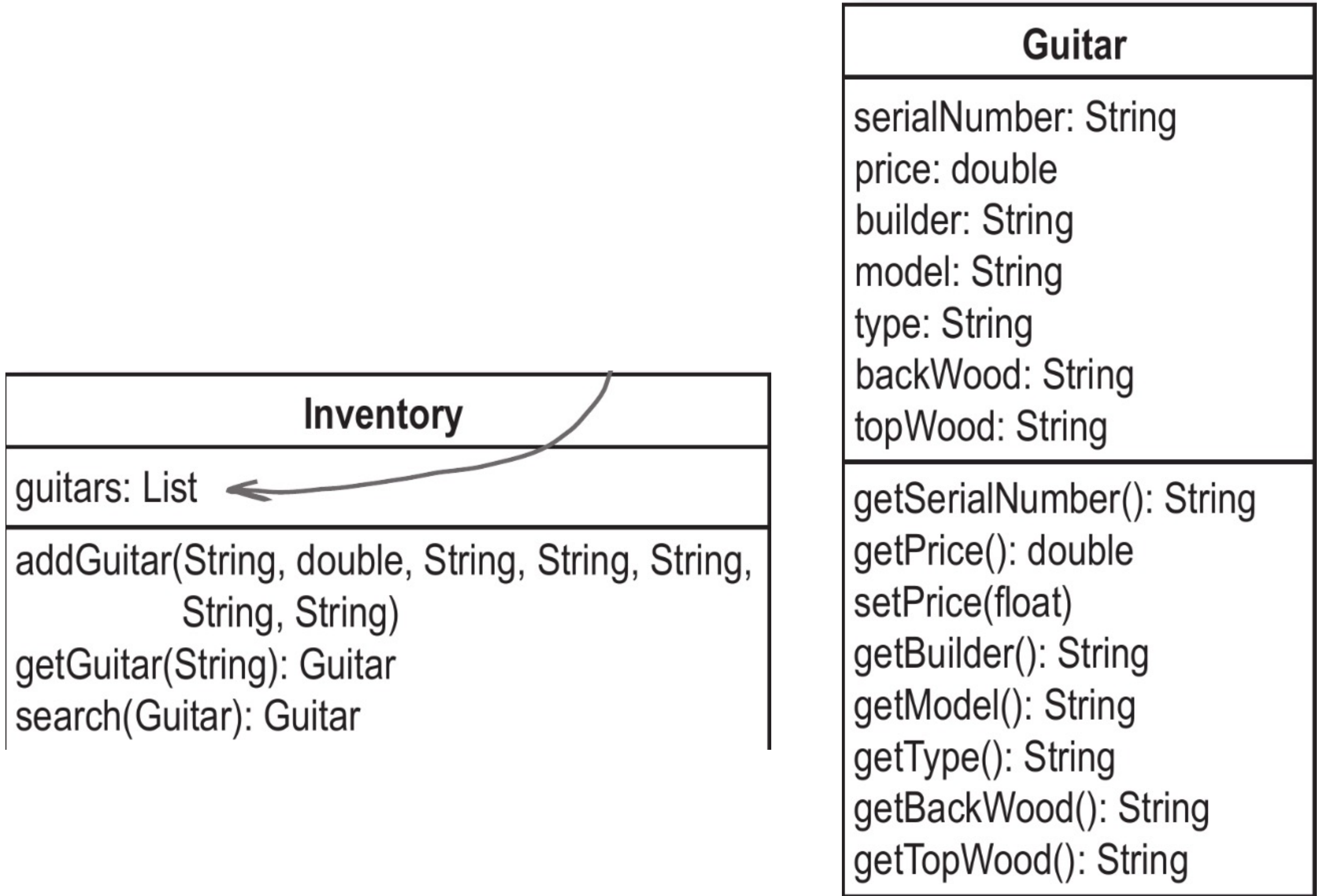
复习

实现用户需求

实现灵活性

实现易维护和代码复用

- 单一职责
- 面向接口编程
- 封装可能改变的模块
- 低耦合
- 开闭原则
- …


Inventory
guitars: List
addGuitar(String, double, String, String, String, String, String)
getGuitar(String): Guitar
search(Guitar): Guitar
Guitar
serialNumber: String
price: double
builder: String
model: String
type: String
backWood: String
topWood: String
getSerialNumber(): String
getPrice(): double
setPrice(float)
getBuilder(): String
getModel(): String
getType(): String
getBackWood(): String
getTopWood(): String

1. 从一段代码说起

lazy initialization

```
public Service getService() {
        if (service == null)
            service = new MyServiceImpl(...);
        return service;
}
```

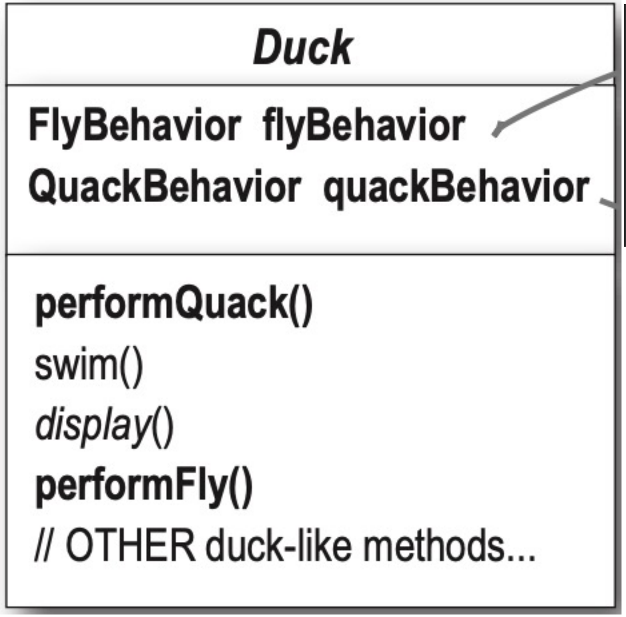
1.1 面向接口编程

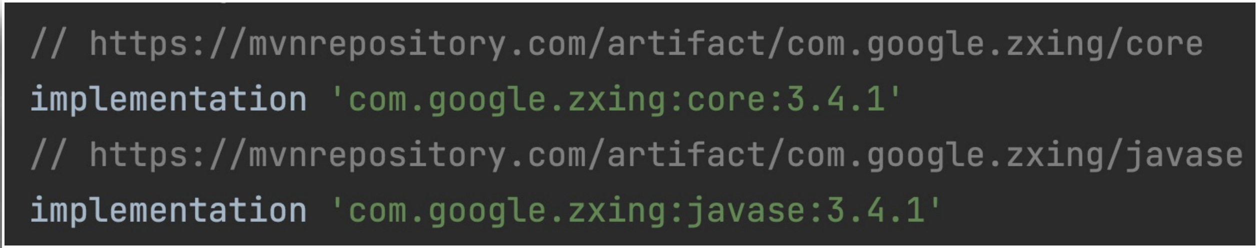
```
public Service getService() {
        if (service == null)
            service = new MyServiceImpl(...);
        return service;
}
```

- 具体的对象可能特别复杂，不利于实现/测试
- 依赖具体的实现，缺乏灵活性

理解“依赖”（dependency）

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.29</version>
</dependency>
```





```
// https://mvnrepository.com/artifact/com.google.zxing/core
implementation 'com.google.zxing:core:3.4.1'
// https://mvnrepository.com/artifact/com.google.zxing/javase
implementation 'com.google.zxing:javase:3.4.1'
```

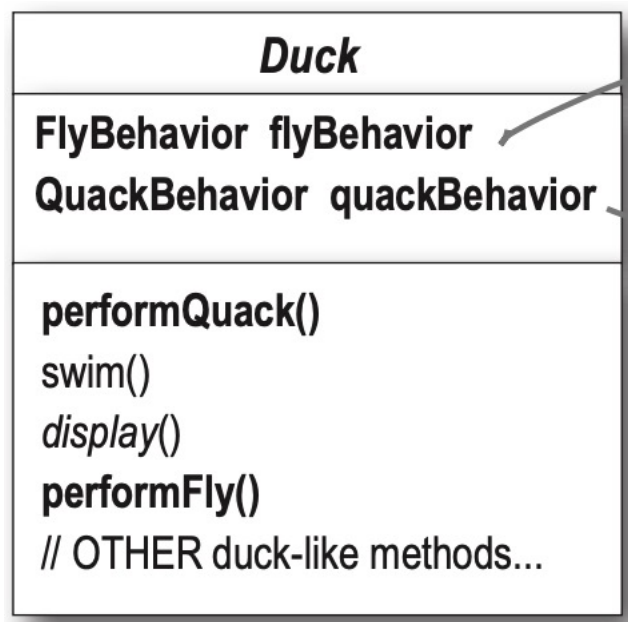
练习

- 使用Zxing这个依赖，生成一段文字的二维码。

参考https://github.com/ChenZhongPu/java-ee-swufe/tree/master/ch3/qrcode

DI

1.2 依赖注入（dependency injection）



Duck 依赖 FlyBehavior，那么如何给 flyBehavior 赋值呢？

显然，我们不能在 Duck 类中使用类似 flyBehavior = new FlyWithWings();

能否更进一步，即“创建依赖”完全独立于代码？

比如配置一个 web 应用的数据库连接，代码只需要关注抽象的 DataSource (DataConnection)，而不需要关心它如何被创建？

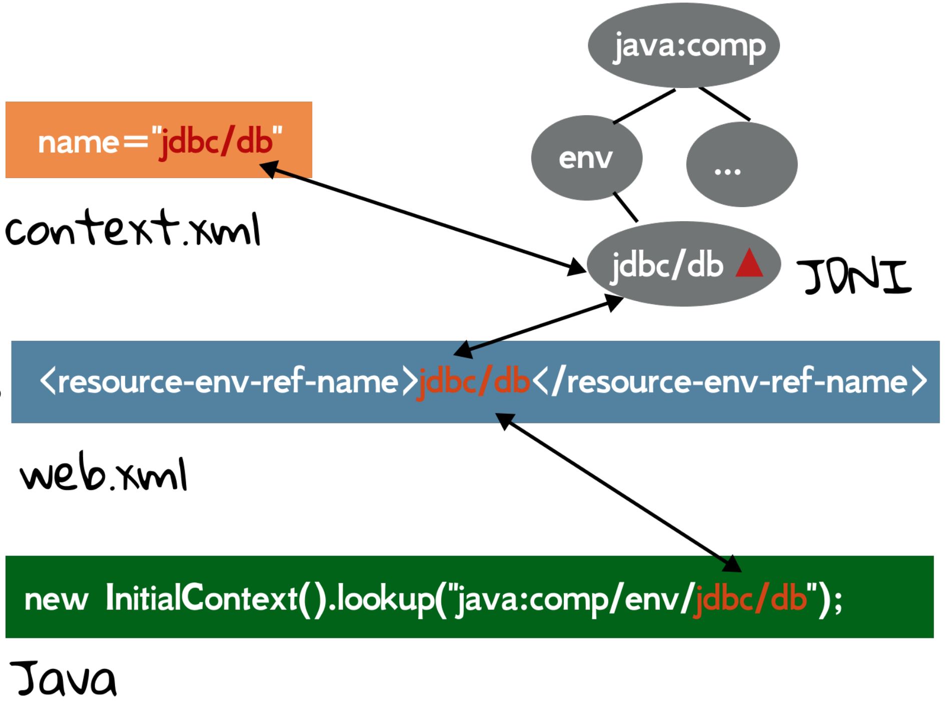
```xml
<Context>
    <Resource
            name="jdbc/db" type="javax.sql.DataSource"
            maxActive="30" maxIdle="10" maxWait="10000"
            url="jdbc:sqlite:/home/zhongpu/github/java-ee-swufe/ch6/test.db"
            driverClassName="org.sqlite.JDBC"
    />
</Context>
```

```java
DataSource dataSource = (DataSource) new InitialContext().lookup("java:comp/env/jdbc/db");
Connection conn = dataSource.getConnection();
```

1.3 JNDI

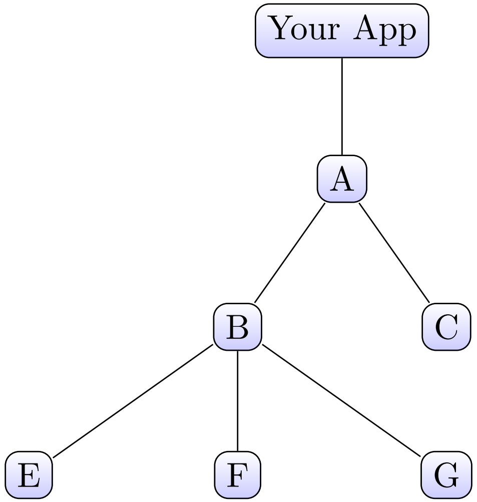
JNDI (Java Naming and Directory Interface) 可以看成是 DI 的部分实现。

思考：虽然我们不再关心 how to create the dependency，但是 *it must be created by someone else*，那么是谁？

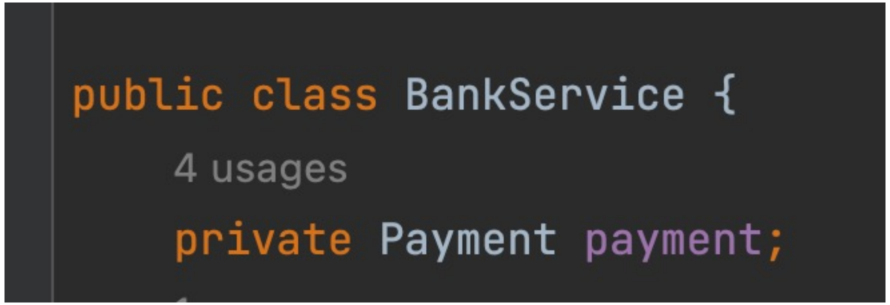

java:comp
env
...
jdbc/db
JDNI
name="jdbc/db"
context.xml
<resource-env-ref-name>jdbc/db</resource-env-ref-name>
web.xml
new InitialContext().lookup("java:comp/env/jdbc/db");
Java

1.4 DI（续）

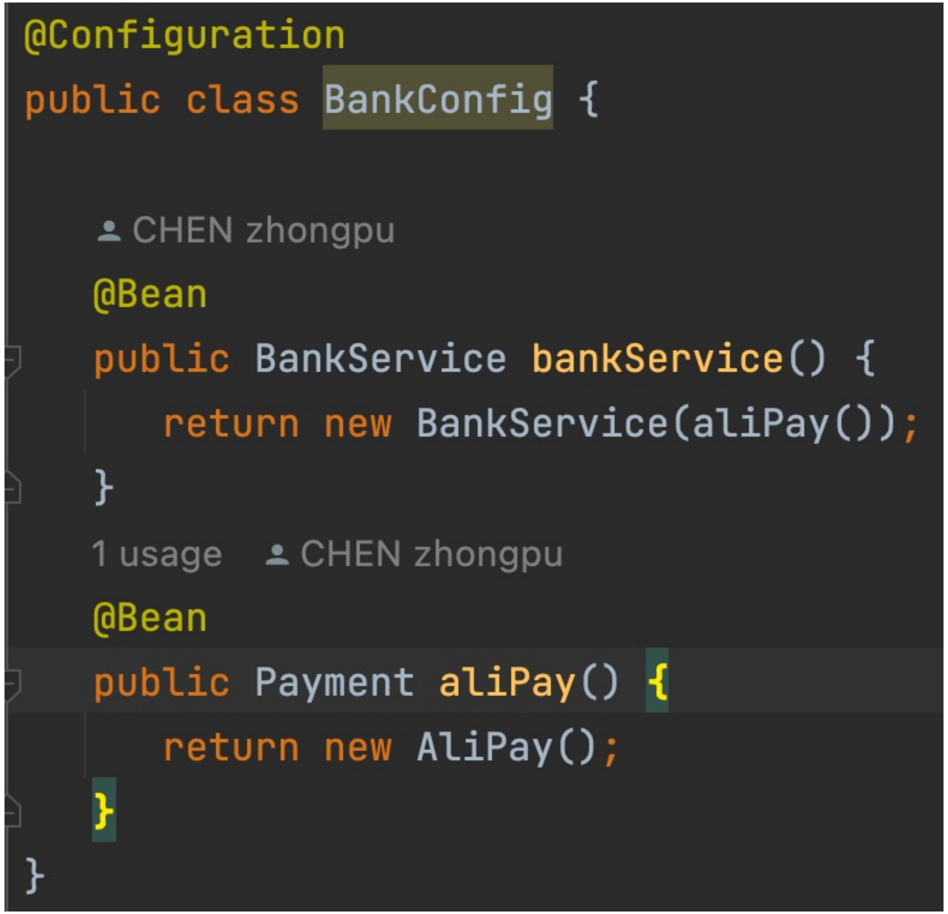
JNDI (Java Naming and Directory Interface) 可以看成是 DI 的**部分实现**。



能否自动装配？



```
public class BankService {
    4 usages
    private Payment payment;
```



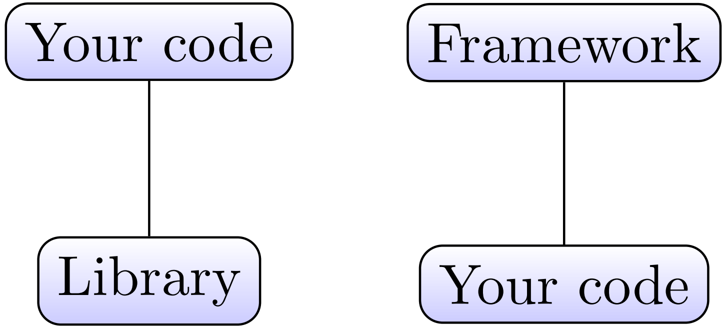
```
@Configuration
public class BankConfig {

    CHEN zhongpu
    @Bean
    public BankService bankService() {
        return new BankService(aliPay());
    }
    1 usage   CHEN zhongpu
    @Bean
    public Payment aliPay() {
        return new AliPay();
    }
}
```

https://github.com/ChenZhongPu/swufe-se/tree/main/week12/spring

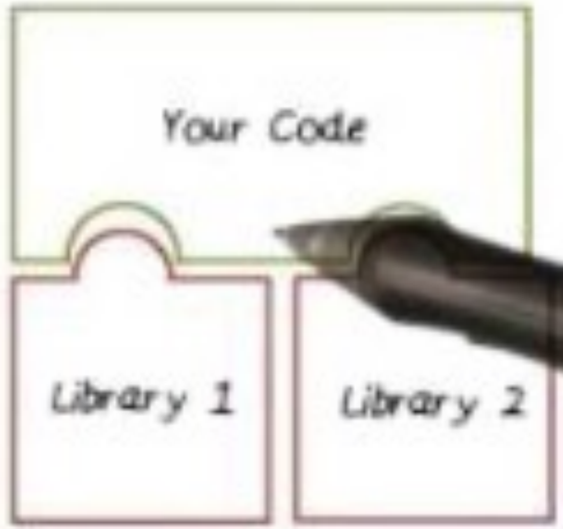
```
BankService service =
context.getBean("bankService", BankService.class);
```

1.5 From DI to IoC

IoC (Inversion of control)，强调的是**控制权的反转**，不仅仅是依赖，还包括 callback，listener 等：**让容器帮你做事情**（包括创建对象）



Frameworks and Inversion of Control



2. Mocking测试

他是成龙的替身，为什么成龙说自己从来不...
twoeggz.com

演员和替身们的合照，成龙的替身原来是他！
sohu.com

成龙从来不用替身？别开玩笑了，他替身多...
twoeggz.com

成龙的替身，吴京的替身，甄子丹的...
xuehua.us

他是成龙的"替身"，拍戏多年满身伤痕，成龙...
sohu.com

他是成龙最中意的替身，成龙送他一辆法拉利，如今...
m.tyule.com

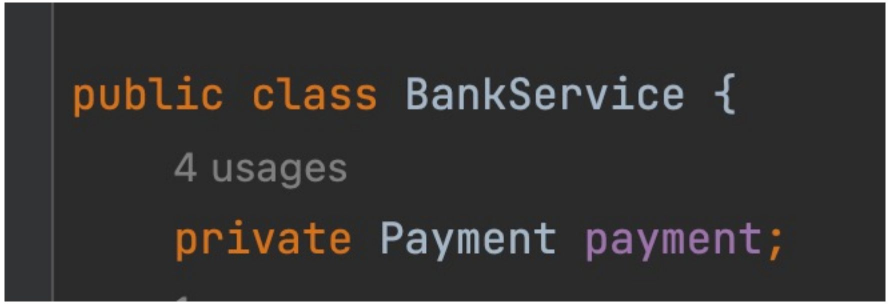
```
public Service getService() {
        if (service == null)
            service = new MyServiceImpl(...);
        return service;
}
```

- 具体的对象可能特别复杂，不利于实现/测试
- 依赖具体的实现，缺乏灵活性

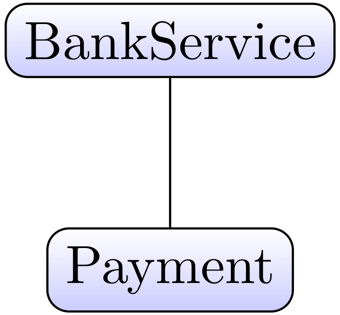
2.1 背景

你有什么解决方案？

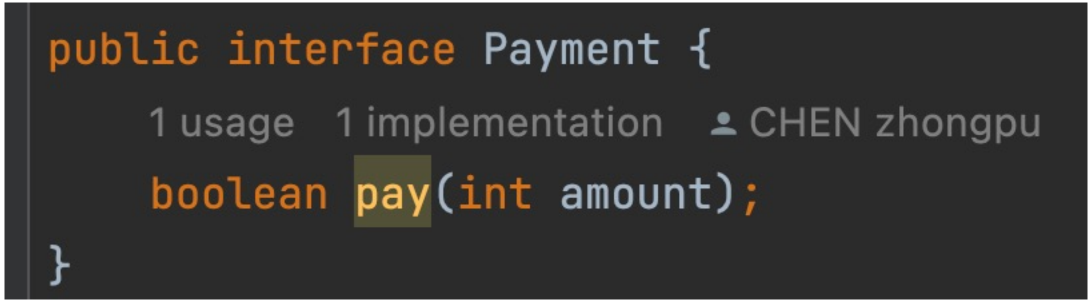
在开发阶段，有时很难使用生产环境的类。



```
public class BankService {
    4 usages
    private Payment payment;
```



2.2 初步方案



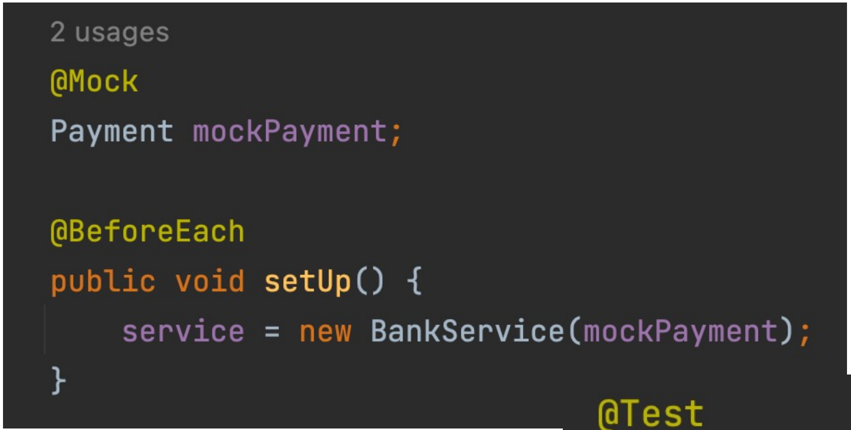
```
public interface Payment {
        1 usage   1 implementation   CHEN zhongpu
    boolean pay(int amount);
}
```

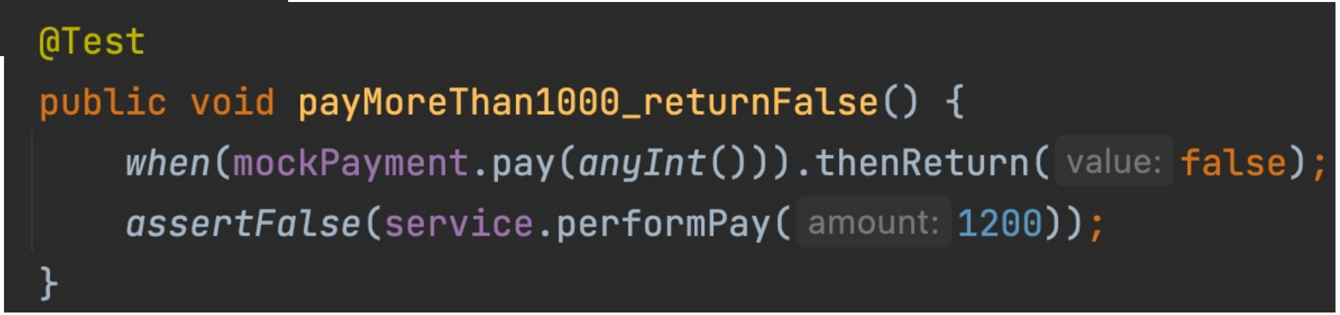
写一个 FakePayment，作为**替身**，用于测试。

- 提前创建测试，TDD（测试驱动开发）
- 团队可以并行工作
- 为无法访问的资源编写测试

如果测试仅关注对象的行为，可以使用考虑使用Mock测试框架

2.3 Mockito


```
2 usages
@Mock
Payment mockPayment;

@BeforeEach
public void setUp() {
    service = new BankService(mockPayment);
}
```


```
@Test
public void payMoreThan1000_returnFalse() {
    when(mockPayment.pay(anyInt())).thenReturn( value: false);
    assertFalse(service.performPay( amount: 1200));
}
```

https://site.mockito.org/

https://github.com/ChenZhongPu/swufe-se/tree/main/week12/mock

小结

- 从 OO 设计原则到 DI
- Mocking 测试