به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر

آزمون نرم افزار

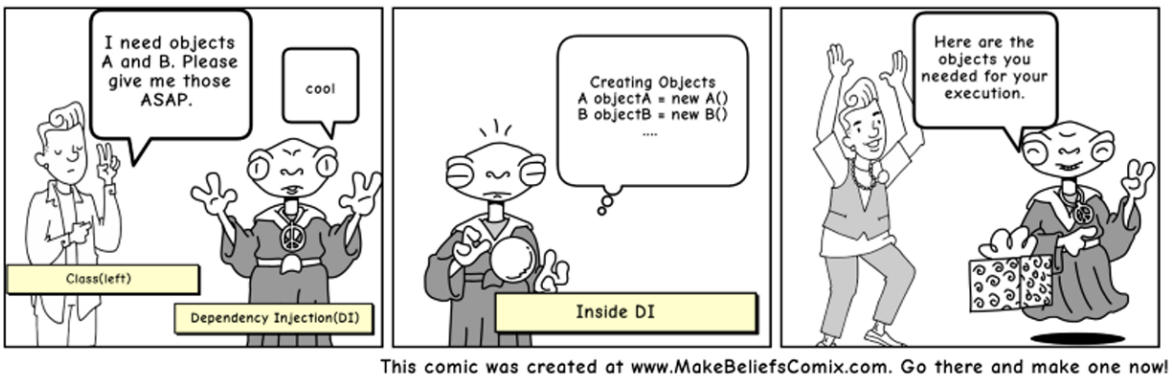
گزارش کار شماره 2

محمد پویا افشاري (810198577)
مصطفی ابراهیمی (810199575)

بخش دوم - گزارش کار

1. سوال اول:

1. Dependency Injection by Constructor
2. Dependency Injection by Setter
3. Dependency Injection by Field



- Dependency Injection by Constructor

در این روش dependency در یک کلاس به وسیله ی constructor آن inject میشود. این روش معمولا بسیار توصیه شده است چرا که mandatory dependency به این وسیله مشخص میشود. بنابر این به این وسیله dependencyهای یک کلاس روشن میشود و immutability بهبود پیدا میکند.

```java
public class UserService {
    private final UserRepository userRepository;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
}
```

- Dependency Injection by Setter

در این روش به کمک setter نیازمندی ها inject می شود. به کمک این حالت در بعد از construct شدن object میتوان انها را تغییر داد. معمولا وقتی به کار می روند که در طول lifecycle برنامه تغییر کنند یا dependency optional باشند.

```java
public class ShoppingCart {
    private PaymentGateway paymentGateway;

    public void setPaymentGateway(PaymentGateway paymentGateway) {
        this.paymentGateway = paymentGateway;
    }
}
```

- Dependency Injection by Field

در این حالت نیازمندی ها مستقیما به وسیله‌ی فیلد ها به کلاس داده میشود. از این حالت به عنوان ساده ترین روش injection یاد میشود. از مشکلات این روش میتوان اشاره به در دسترس بودن عمومی نیازمندی ها کرد که تست پذیری را سخت تر می‌کند. در حالي که این روش production code را تضعیف می‌کند ولی میتوان از ان برای prototype محصول در کنار روش های دیگر استفاده کرد.

```
public class OrderService {
    @Autowired
    private OrderRepository orderRepository;
}
```

2. سوال دوم:

a. به Imposter test که به عنوان "imposter object" یا "impersonator object" نیز شناخته می‌شود. تا حد امکان رفتار مشابه object واقعي دارند ولي با هدف isolation از DOC. از انجا که imposter ها رفتار کاملا نزدیک object هایی دارند که آنها را شبیه سازی میکنند و تقلید آنها هستند از این رو این نامگذاری انتخاب شده است.

b. اصولا تست doubleها برای نوشتن کد های بهتر به کمک ما می‌آیند به این وسیله که محیط isolated کد SUT را از DOC جدا می‌کنند بر طبق صورت طرح درس پنج Double مطرح شده است.

i. Dummy Objects: این Object ها فقط حکم Create شدن براي ادامه روند در طول تست را دارند و ساده ترین نوع دابل تعریف میشود.

ii. Stub: در این حالت test double به ازاي method callهای مختلف خروجی fix میدهد.

iii. Spy: نوعی از test double تعریف میشود که interaction را ذخیره کرده و بر اساس آن با SUT ارتباط میگیرد.تفاوت این نوع و نوع بعدي در ان است که spies به صورت wrapper بخشي از متدهاي مد نظر يك شي را در برميگيرد و رفتار اصلي object بافي مي‌ماند.

iv. Mock: Mock Object ها simulate شده‌ي رفتار يك object تعریف می‌شوند به این صورت که میتوانند expected behavior method را تعریف کرده و در mock object از ان استفاده کنند. گفته میشود که mock ترکیب spy , stub میباشد.

v. Fake Objects: معمولا ساده شده و بهینه شده‌ی یک کامپوننت کامل می‌باشد. نظیر database و یا front-end

3. سوال سوم:

دو شیوه‌ی اصلی مواجه با unit testها به شیوه های Mocking تست کردن و یا Classical می باشد.هر یک از این روش ها مشکلات و مزیت های خودش را دارد که در ادامه به شرح پرداخته شده است.

Classicist	Mockist
Like working with real objects	Prefer working with fake objects
State verification	Behavior verification
Use mocks to test collaborations	Use mocks all the time
Will hard code collaboration	Will mock collaborations
TDD from middle out	TDD from the outside in
Use ObjectMothers/Factories for test setup	Will mock only what they need for test collaboration
Test tend to be more coarse grained – approaching more integration style tests	Tests tend to be very fine grained – may miss integrations
Classists don’t couple tests to implementation	Mockists do
Classists don’t like thinking about implementation when writing tests	Mockists do
Don’t mind creating query methods to support testing	Mockists typically don’t have to
Classists style can encourage Asking Not Telling design	Mockists style encourages Tell Don’t Ask

- تست کلاسیک: این روش که با state-based هم شناخته میشود بر روی تغییرات state در صورت اجرای متد یا اپرند تمرکز دارد.
  - مزایا:
    - سادگی: نوشتن تست ها در این روش ساده است.
    - تاکید روی رفتار: تاکید در این حالت روی تست نوشتن بر روی state اولیه و نهایی است
    - Less coupling: تست های کلاسیک معمولا coupling کمتری دارند
  - معایب:
    - در دسترس نبودن componentها : در محیط های عملی واقعی و محیط های complex معمولا DOC ها ساخته نشده / در دسترس نبوده / کند و ... هستند و از این روش در عمل نمیتوان برای دسترسی SUT به آنها به خوبی و همه جا استفاده کرد.
    - محدود بودن behavioral verification: تست های از این دست معمولا برای interaction های complex به خوبی قادر پیاده سازی نیستند و بنابر این verify کردن آنها مشکل خواهد بود.
    - Miss collaborative issues: از انجا که در این حالت روی حالت نهایی state تمرکز بیشتر است امکان از دست دادن توجه به interaction با dependency متصور است.
- تست mock:
  تست mock که گاها با behavioral based testing نیز مطرح می‌شود تمرکز زیادی روی ارتباط با dependency ها دارد.در این روش mock object ها برای verify کردن SUT به کمک می‌آیند.
  - مزایا:
    - بررسی رفتار: در این حالت به دلیل بهره گیری از mocking میتوان behavioral تست ساده تری انجام داد.
    - تمرکز بر روی collaboration: در این حالت ارتباط بین object ها به سادگی برقرار شده و برای سیستم های پیچیده مناسب میشود.
  - معایب:
    - پیچیدگی: اگر component ها coupling زیادی داشته باشند یا به صورت testable ساخته نشده باشند (دارای متد های static یا فیلد private و ..) باشند تعریف mock در این حالت سخت میشود.
    - Overuse mock: استفاده بیش از حد از mock ها خطر "over-mocking" را افزایش داده و کار maintain کردن را سخت تر میکند.

- در استفاده هر یک از دو روش: مسلما دلیلی برای همیشه تست نوشتن کلا به روش mock وجود ندارد – در حالت mock test نیاز به ایجاد Fixture مناسب هر چند ساده تر هست و مشکلات گفته شده را دارد ولی از نظر من استفاده از هر یک از دو روش برتری هایی دارد و میتوان از ترکیب هر دو این روش ها استفاده کرد. استفاده از تست های mock باعث افزایش flexibility میشود در حالی که شکنندگی در تست های classical کمتر خواهد بود. بنابر این احتمالا من تا حد امکان classical را ترجیح میدهم و از mocking framework ها برای اجرای stubها استفاده میکنم. همینطور در اجرای تست ها نیازمند DOC که در ارتباط گرفتن با انها سربار زیادی دارم سمت اچرای mock مناسب میروم.