Refactoring with Sandwich Pattern

Context

DOMAIN DRIVEN DESIGN EUROPE
HANDS-ON CONFERENCE
FEBRUARY 4-5 2021
ONLINE ONLY

This Talk is inspired by

VLADIMIR KHORIKOV

@vkhorikov

GABRIELE TONDI

@racingDeveloper

About us

SEPEHR NAMDAR

@SepNamdar

DOMAIN DRIVEN DESIGN IRAN

@DDD_Iran

HADI AHMADI

@HAhmadi15

Problem

- A Legacy Code
- Without Test
- Procedural
- With Anemic Domain Objects
- Add Some New Features

Goal

- **A Domain Centric Code**
- **With Rich Domain Objects**
- **With Tests**

Where to start ?

This slide is intentionally left blank

First of all : Learn the Domain

As a Human Resource

I want to find an available Recruiter

According to my Candidate Availabilities

***“Who can test”** my Candidate.**

Then : Ask questions

First of all : Learn the Domain

As a Human Resource

I want to find an available Recruiter

According to my Candidate Availabilities

“Who can test”* **my Candidate.**

Who can test : The Recruiter should cover all Candidate’s Skills.

Let’s have a look !

Java™

https://github.com/SepehrNamdar/sandwich-driven-development/

https://github.com/H-Ahmadi/DDDEU21_Sandwich_Driven_Development

Step 1 : Protect your code with tests

***Refactoring* is changing the code structure without changing its behaviour**

Bad Practice: Start *Refactoring* without a test coverage

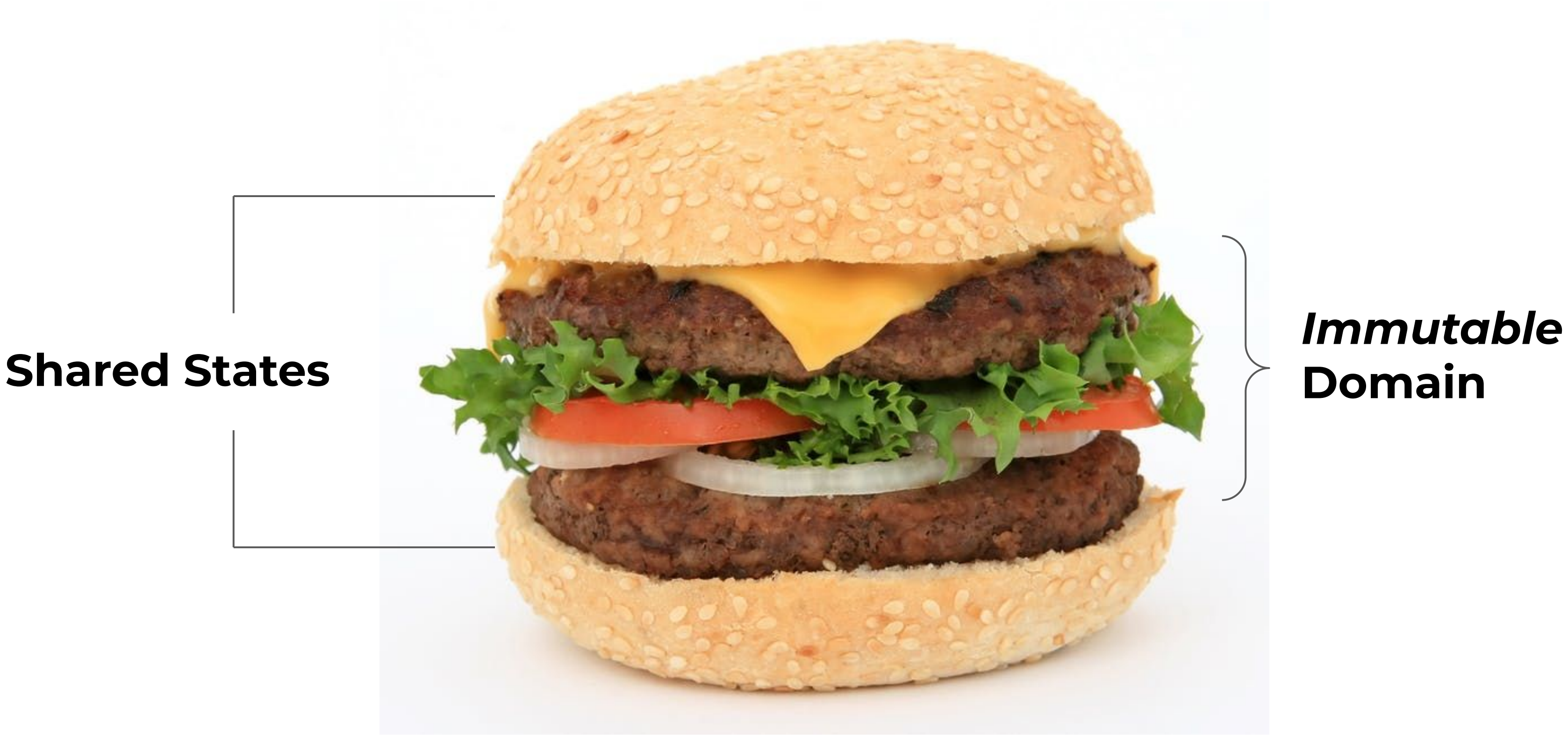
Writing tests is too long and boring

Step 1 : Protect your code with Approval Tests

- **Advantages:**
  - **Fast to write**
  - **Easy to learn**
  - **Multi Platform**
  - **Compact**
  - **Based on Golden Master**

- **Disadvantages:**
  - **Not enough by itself**

Step 2 : Apply the Sandwich Pattern



Step 3 : Make your Domain Model Rich

Step 4 : A new Business Rule

Recruiter Availabilities must be booked before plan the Interview

2 solutions for this Temporal Coupling

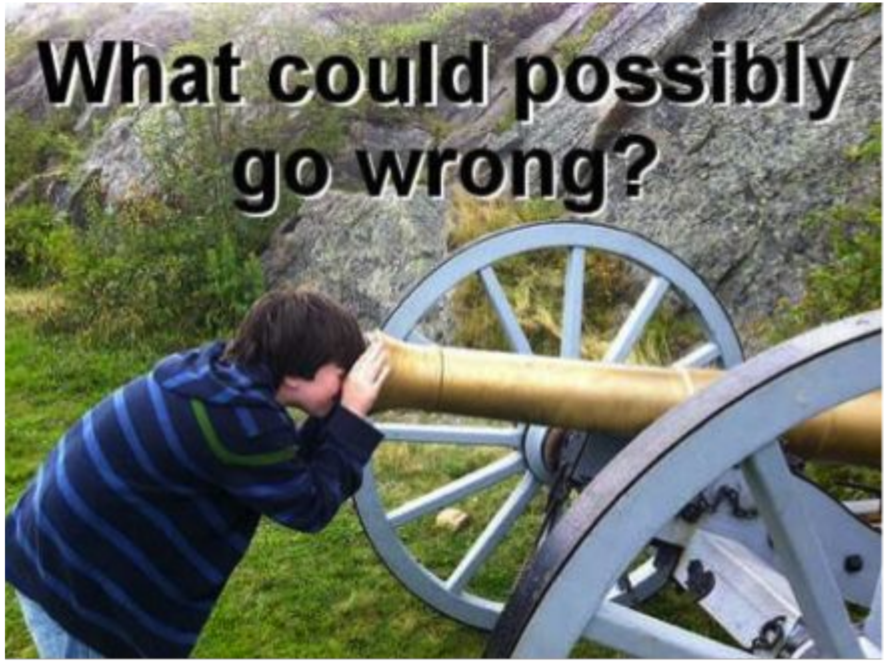
1. Use your tests (Best solution):

```
private RecruiterRepository recruiters = Mockito.mock(RecruiterRepository.class);
private InterviewRepository interviews = Mockito.mock(InterviewRepository .class);
InOrder inOrder = Mockito.inOrder(recruiters, interviews);
inOrder.verify(recruiters).bookAvailability(recruiter, interviewDate);
inOrder.verify(interviews).save(interview);
```

2. Force your method to return a value (Our choice):

```
Recruiter recruiter  = recruiters.bookAvailability( appropriateRecruiter , availability);
```

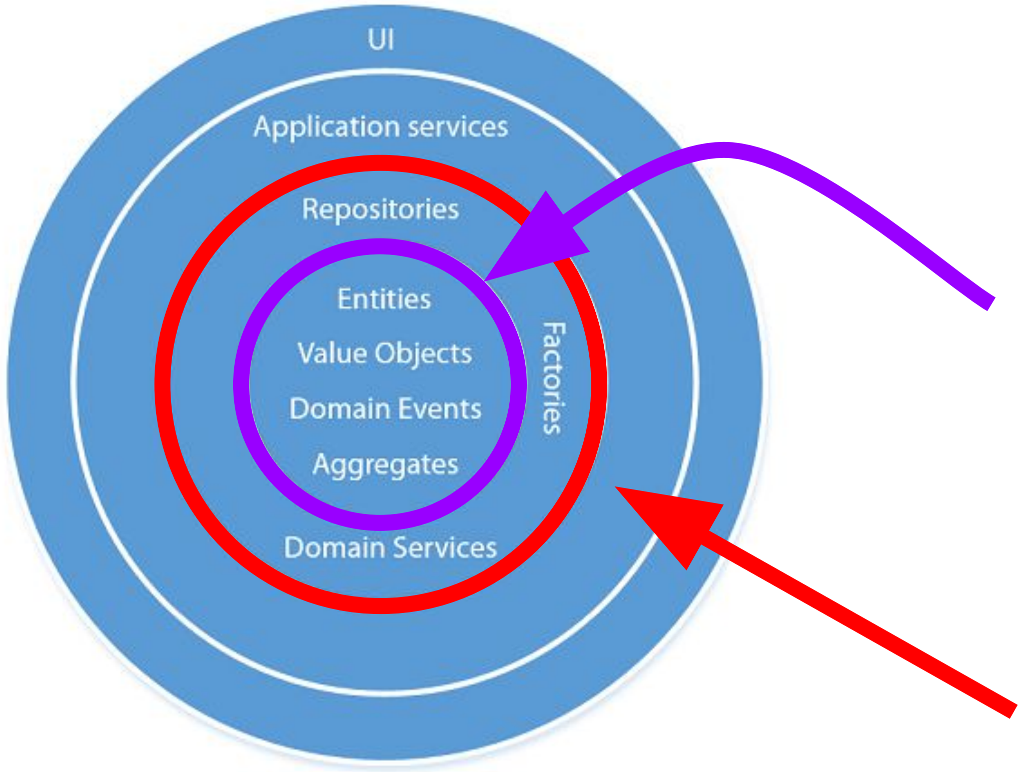
The second solution !



Domain Model Purity

- **Domain layer does not depend on any *external resource* or *framework***
- **Its objects know only about *primitive* or other *domain objects***

Step 5 : A Pure Domain Model



Use a Domain Service !

Pure part of Domain Model

Domain Model

Did that really helped ?

Domain Model Completeness

- **Domain layer contains all business rules and Domain Logic**

Step 6 : Domain Model Completeness

Gather all Business Rules into Domain Layer !

What Could Possibly Go Wrong ?

Conclusion

- **You can never have a Domain Model which is absolutely *Efficient*, *Complete* and *Pure* !**

Conclusion

- **Sandwich Pattern helps you to make your Domain Model *Pure* and *Complete***

Conclusion

- **Sandwich Pattern is not the most *Efficient***

Questions ?

@SepNamdar @DDD_Iran @HAhmadi15

@SepNamdar @DDD_Iran @HAhmadi15

Resources

- https://enterprisecraftsmanship.com/posts/domain-model-purity-lazy-loading/
- https://enterprisecraftsmanship.com/posts/domain-vs-application-services/
- https://enterprisecraftsmanship.com/posts/temporal-coupling-and-immutability/
- https://vimeo.com/107963074
- https://fr.slideshare.net/JAXLondon2014/crafted-design-sandro-mancuso
- http://videos.ncrafts.io/video/221024483