Test Driven Development (TDD)

Learning concepts

We LEARN more when we LEARN TOGETHER

Learning concepts

We LEARN more when we LEARN TOGETHER

The heart of learning is to focus on the concepts, which is more important than focusing on the details 🥰

Learning concepts

We LEARN more when we LEARN TOGETHER

The heart of learning is to focus on the concepts, which is more important than focusing on the details 🥰

I can **EXPLAIN** it to you, but I can't **UNDERSTAND** it for you 💪

Agenda

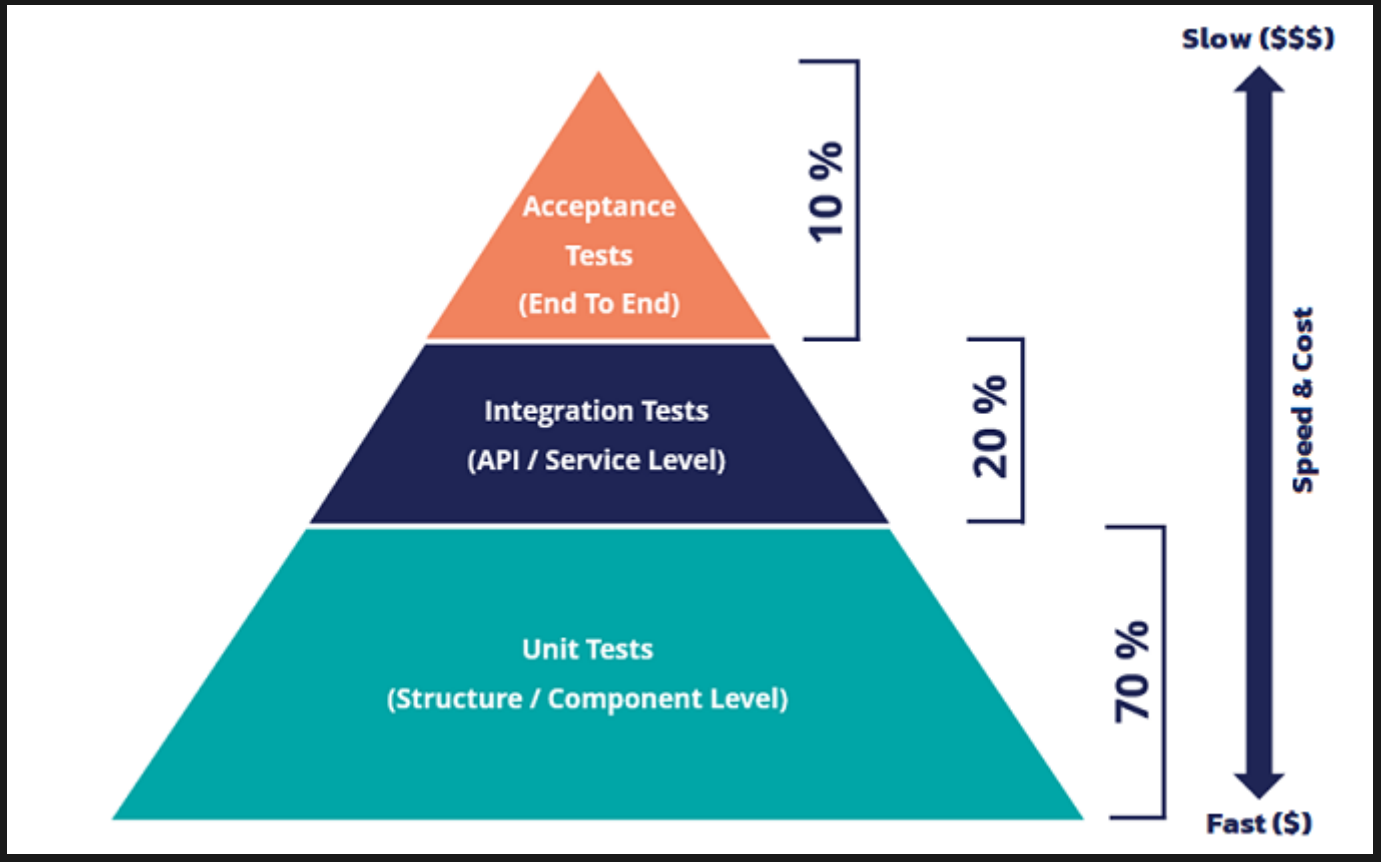
1. The World of Software Testing
2. Code Refactoring
3. Isolated Test Environments
4. High Level Testing
5. Workshop

Isolated

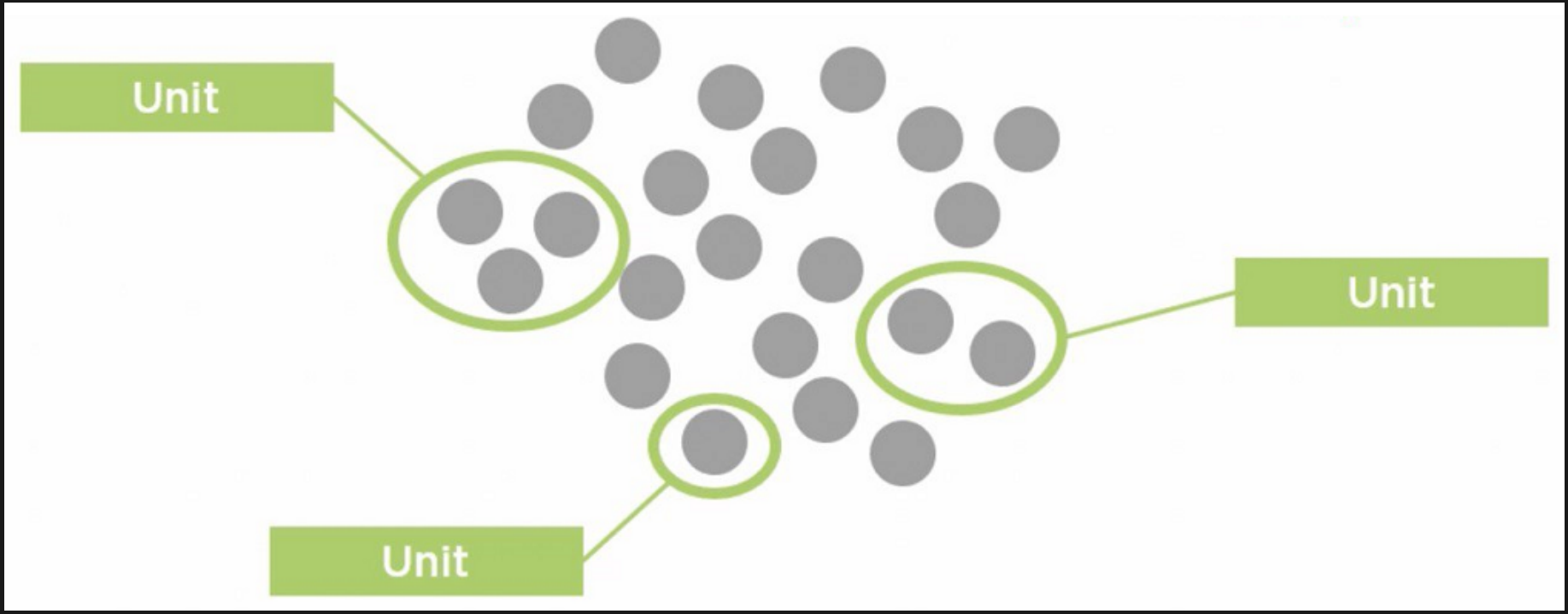
Test Environments

Unit Testing & Integration Testing

The Practical Test Pyramid

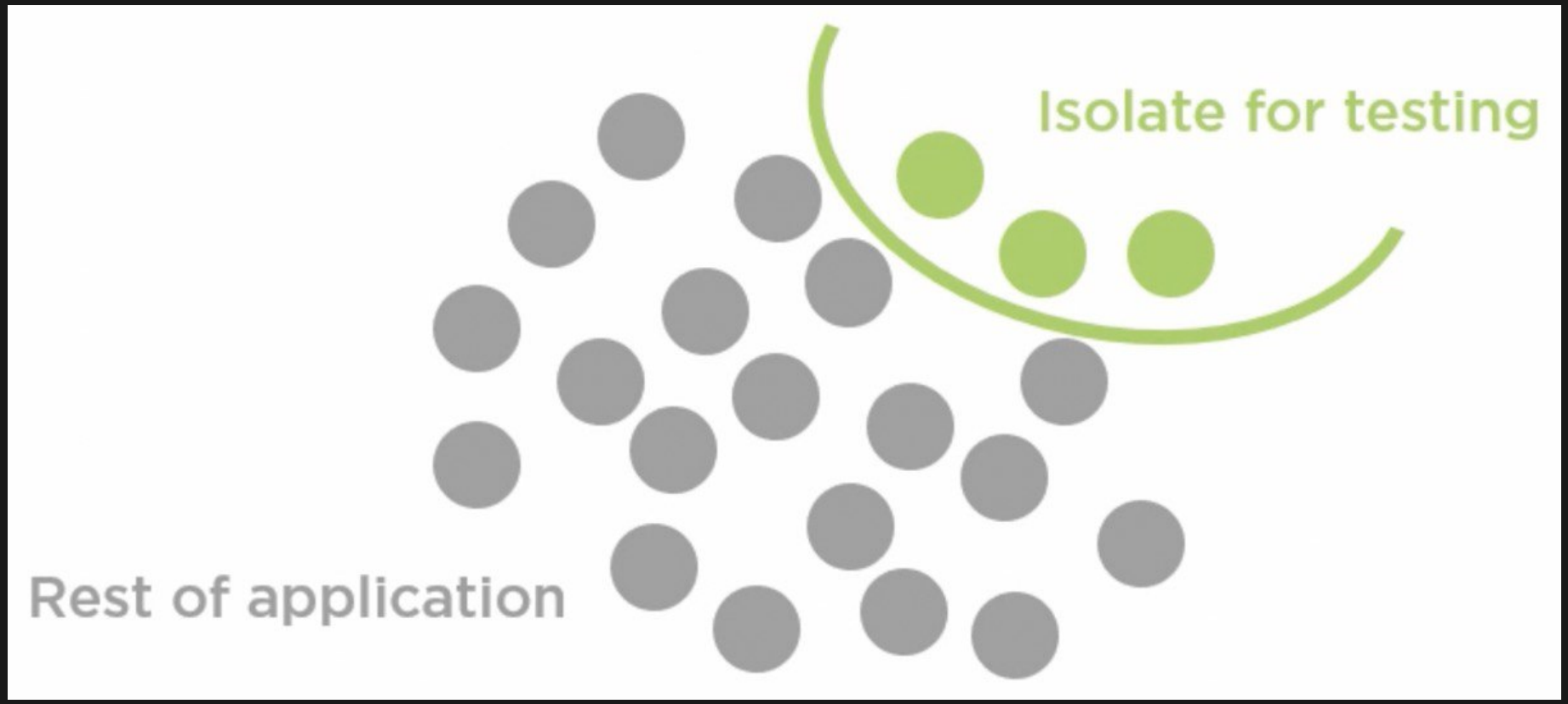


Unit Testing



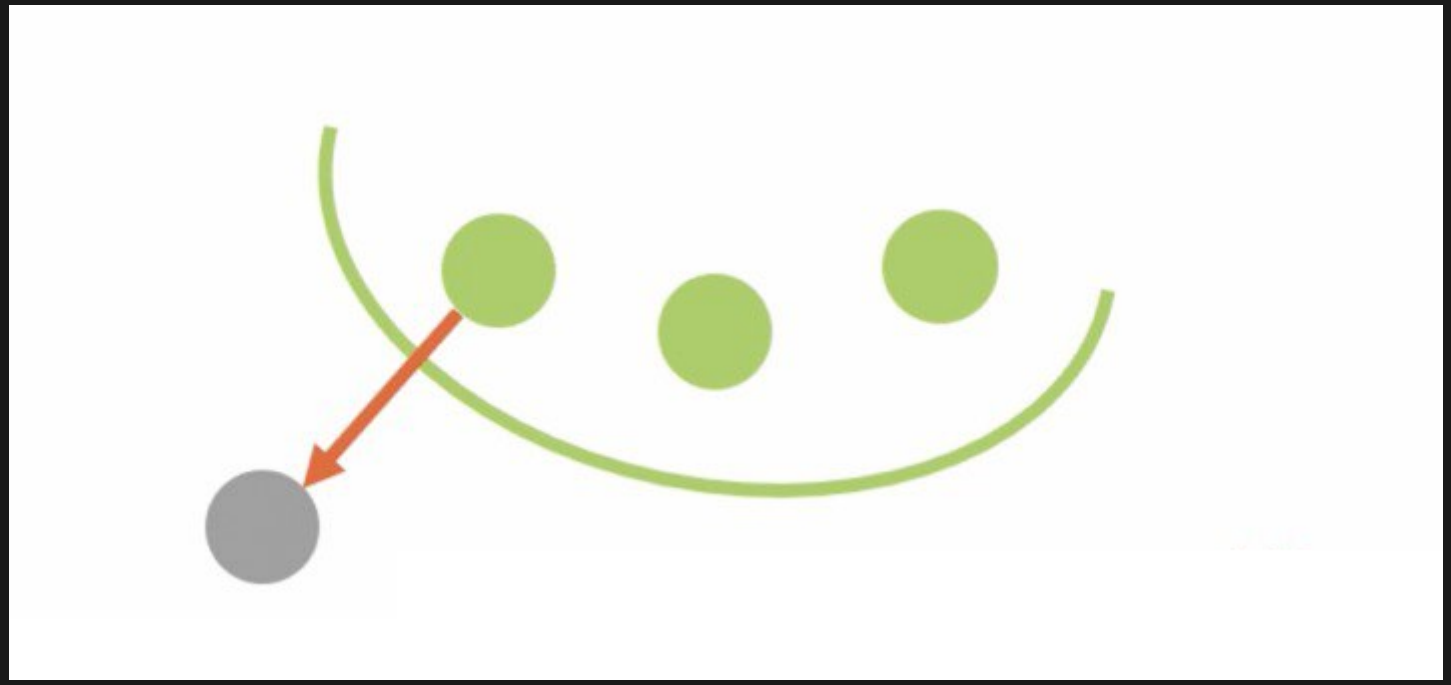
🎯 ทดสอบของต่างๆให้มีพฤติกรรมตามที่เดฟคาดหวัง

Isolate for testing



💡 แยกของที่จะเทสออกมา เพื่อป้องเทสหลุดเข้า production source

Isolate for testing



บางสถานะการณ์ของที่จะเทสไปพันกับเรื่องอื่นโดยเลี่ยงไม่ได้ เช่น ต่อ database, เรียกใช้งาน module อื่น ซึ่งส่งผลให้คุมผลลัพท์ไม่ได้

 Test Double

> “It is a generic term for any case where you replace a production object for testing purposes.” — Martin Fowler

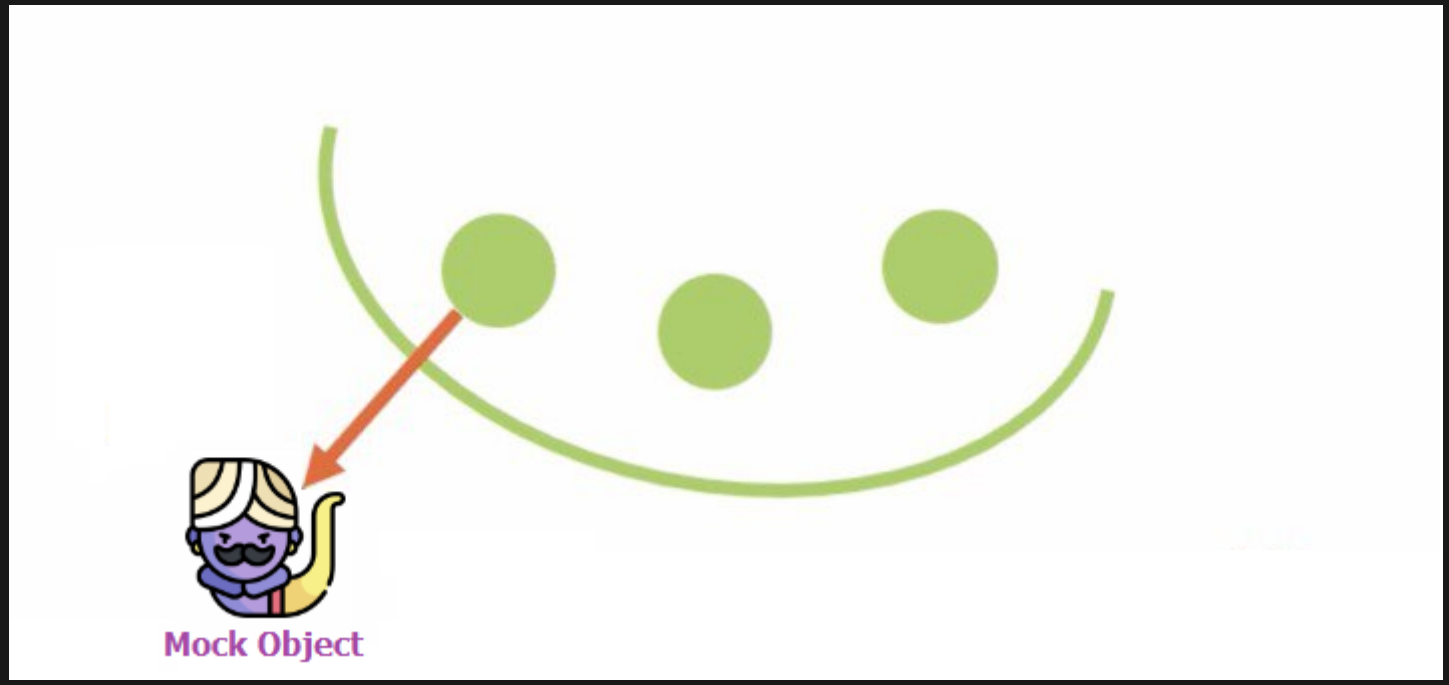
1. `Dummy` objects are passed around but never actually used. Usually they are just used to fill parameter lists.
2. `Fake` objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an InMemoryTestDatabase is a good example).
3. `Stubs` provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test.
4. `Spies` are stubs that also record some information based on how they were called. One form of this might be an email service that records how many messages it was sent.
5. `Mocks` are pre-programmed with expectations which form a specification of the calls they are expected to receive. They can throw an exception if they receive a call they don't expect and are checked during verification to ensure they got all the calls they were expecting.

Mock Object

“mock objects are simulated objects that mimic the behavior of real objects in controlled ways, most often as part of a software testing initiative. A programmer typically creates a mock object to test the behavior of some other object, in much the same way that a car designer uses a crash test dummy to simulate the dynamic behavior of a human in vehicle impacts.” — Wikipedia

Object ที่เดฟสามารถควบคุมพฤติกรรมได้ เอาไว้ใช้ในการทดสอบ

Isolate for testing



ใช้ Mock object กับสิ่งที่เป็น dependency กับของที่เราต้องการทดสอบ เพื่อ
ใช้ในการควบคุมพฤติกรรม และ การตรวจสอบการทำงานต่างๆ

Demo D04

Mock Object

https://github.com/moq/moq4

Moq

Setup & Verify

```
// ตั้งค่าการทำงานของ Mock object
var mock = new Mock<ICalculator>();
mock.Setup(it => it.Add(3, 4)).Returns(7);

// เรียกใช้ Mock object
ICalculator calculator = mock.Object;
var result = calculator.Add(3, 4);

// ตรวจสอบความถูกต้อง
Assert.Equal(7, result);
mock.Verify(it => it.Add(3, 4), Times.AtMostOnce(
```

Moq

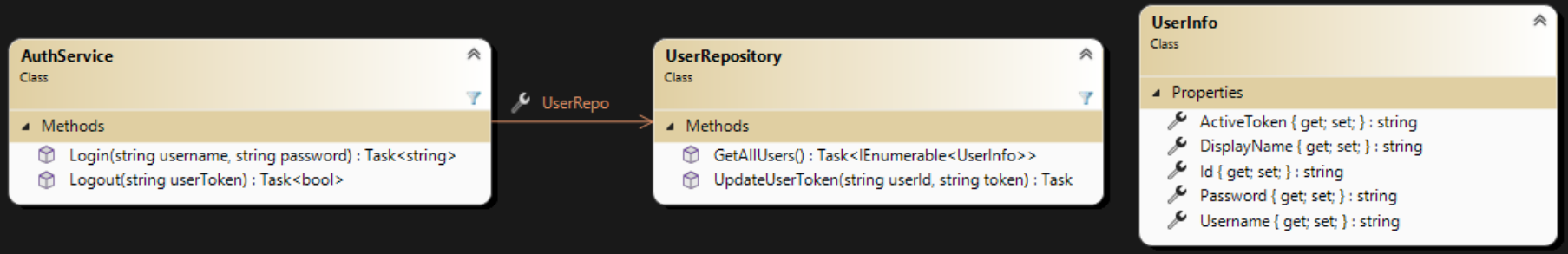
`Loose` / `Strict` behaviours

```
// Loose behaviour (default)
var looseMock = new Mock<ICalculator>(MockBehavior.Loose);
ICalculator calculator = looseMock.Object;
var result = calculator.Add(3, 4); // result: 0
```

```
// Strict behaviour
var strictMock = new Mock<ICalculator>(MockBehavior.Strict)
ICalculator calculator = strictMock.Object;
var result = calculator.Add(3, 4); // Throw MockException
```

Challenge 11

Authentication Service

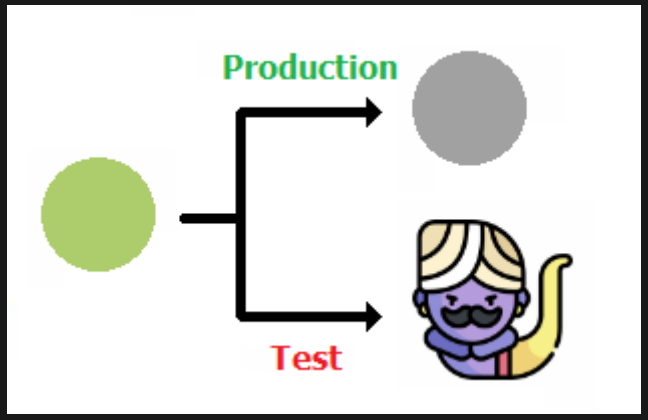


ประโยชน์จากการใช้

Mock Object

1. ควบคุมสภาพแวดล้อม และ พฤติกรรมทุกอย่างได้
2. เร็วกว่าการทำงานจริง เช่น ต่อ database
3. ทำซ้ำได้เรื่อยๆไม่มีข้อจำกัด เช่น ค่าใช้จ่าย

🤔 จะแยกไม่ให้ปนกันยังไง ?



Environment	Expected
Production	ทำงานกับโค้ดที่เขียนไว้จริงๆ
Test	ทำงานกับ Mock object

Testable Code

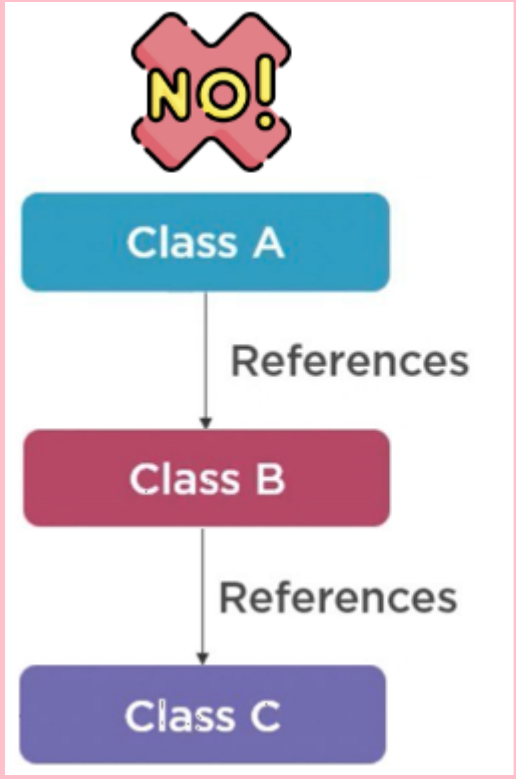
ไม่ได้เกิดขึ้นเอง มันต้องเกิดจาก

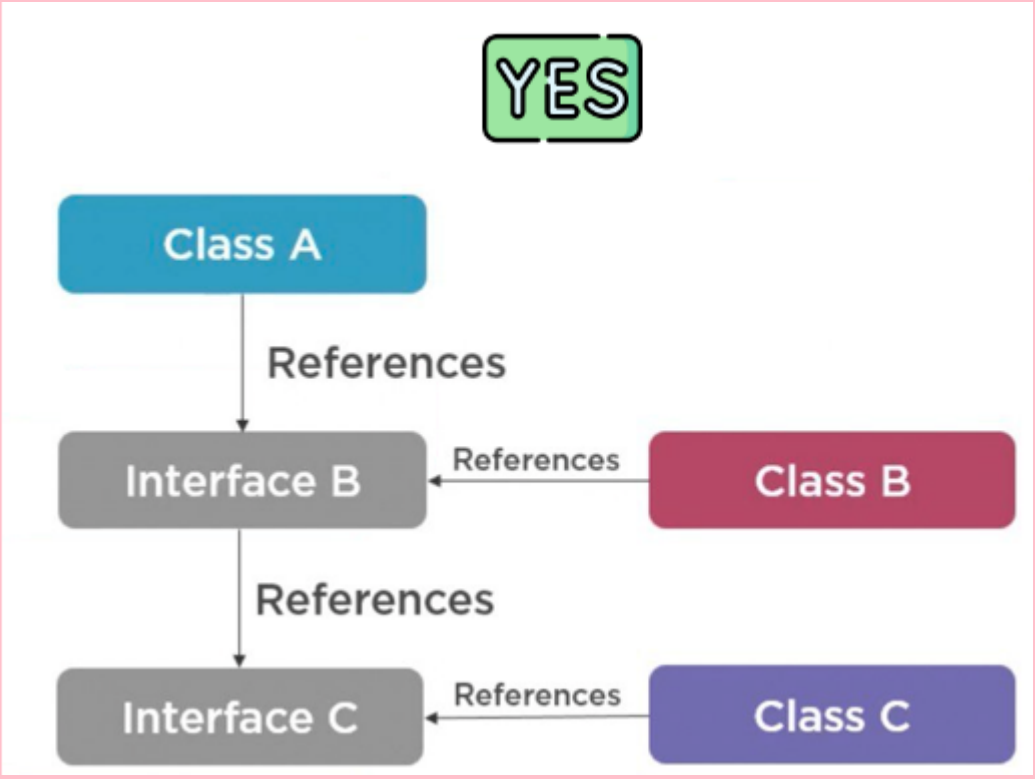
ความตั้งใจ - ตั้งแต่กระบวนการออกแบบ

Design Principles

Dependency Inversion Principle

1. High-level modules SHOULD NOT depend on low-level modules. **Both should depend on abstractions.**
2. Abstractions SHOULD NOT depend on details. **Details should depend on abstractions.**





Design Principles

Dependency Inversion Principle

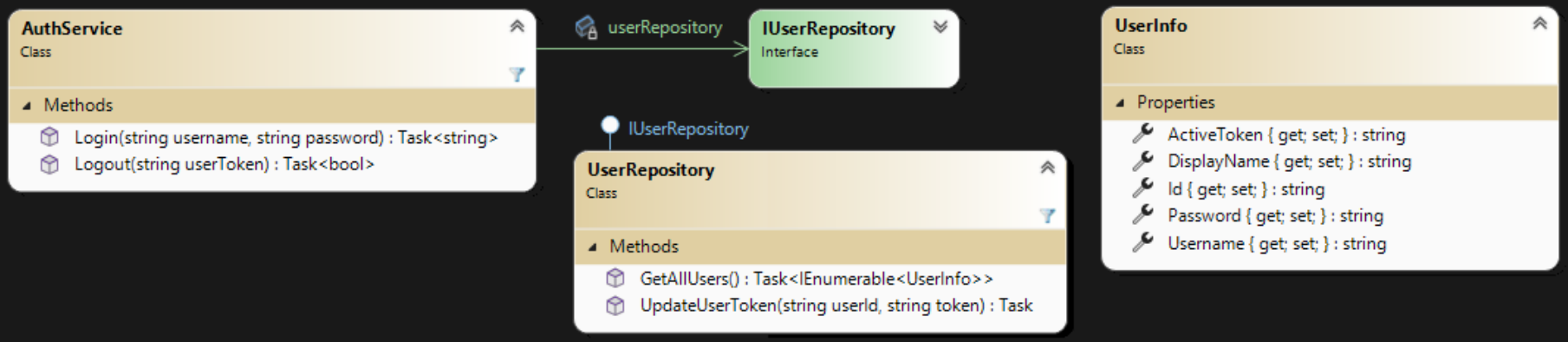
หัวใจหลักของ OOP คือการมองของต่างๆเป็น `Components` เพื่อแบ่งหน้าที่ในการรับผิดชอบออกจากกัน แล้วให้ควบคุม `Communications` ที่เกิดขึ้นจากการทำงานร่วมกันของ Components เหล่านั้น

ถอดเปลี่ยนชิ้นได้

Challenge 11 (Part 2)

Refactoring

Dependency Inversion Principle

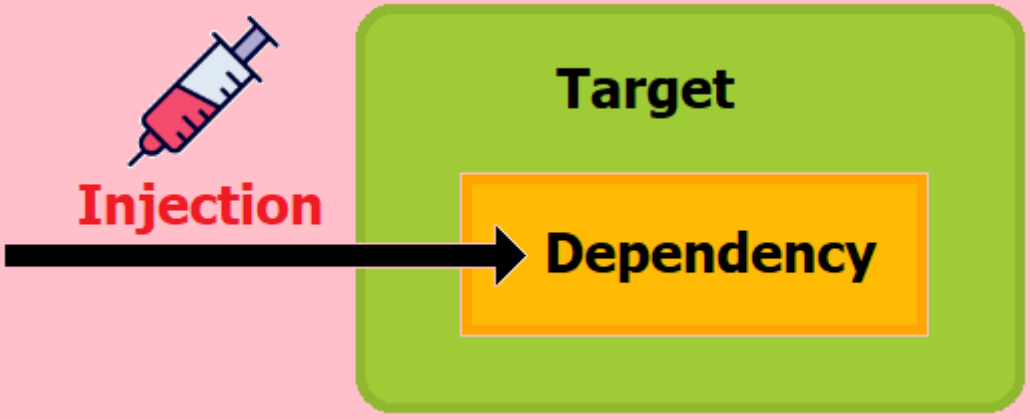


Design Pattern

Dependency Injection

"It is a design pattern in which an object receives other objects that it depends on. Dependency injection aims to **separate the concerns of constructing objects** and using them, **leading to loosely coupled programs**." — Wikipedia

ใช้ DI เพื่อช่วยในการแยก Test กับ Production ออกจากกัน



Demo `D05`

Dependency Injection

`(built-in) ASP.NET MVC` `(custom) SimpleInjector`

Demo D06

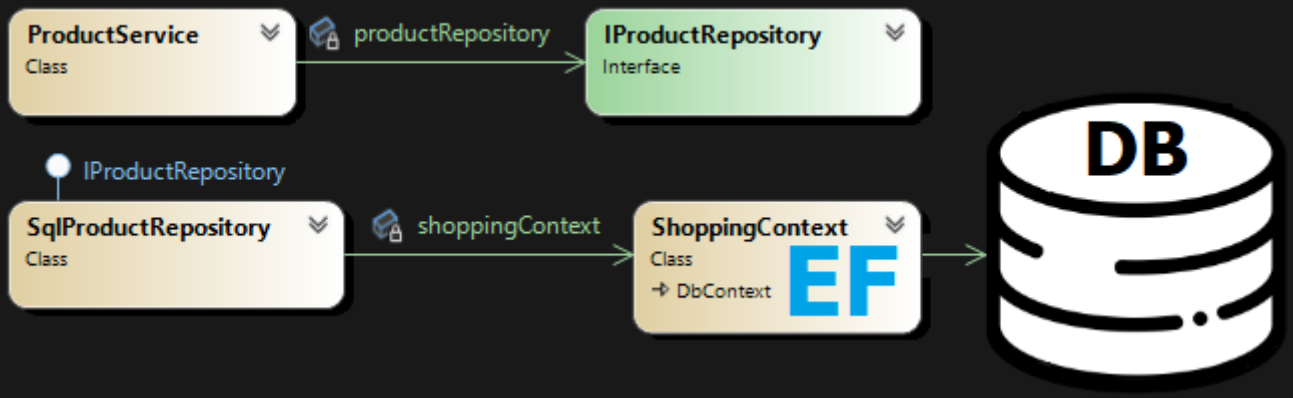
AutoFixture

AutoFixture makes it easier for developers to do Test-Driven Development by automating non-relevant Test Fixture Setup, allowing the Test Developer to focus on the essentials of each test case.

Testing with

Entity Framework

🤔 กรณีต่อ DB จะเทสยังไง?



☝ Mock Object


ProductService
Class
productRepository
IProductRepository
Interface
IProductRepository
SqlProductRepository
Class
shoppingContext
ShoppingContext
Class
DbContext
EF
DB

✌ Mock Object


ProductService
Class
productRepository
IProductRepository
Interface
IProductRepository
SqlProductRepository
Class
shoppingContext
ShoppingContext
Class
DbContext
DB

Mock = โบกมือลา

Database Magic

`DEFAULT` `CHECK` `NOT NULL` `UNIQUE KEY` `PRIMARY KEY` `FOREIGN KEY`

`AUTO INCREMENT` `INDEX` `DBMS stuff`

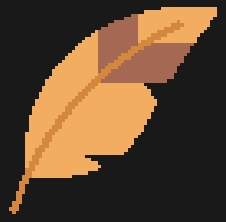
👌 Temporary Database

 Database

จะเทส DB ยังไง ?

Lightweight & In-memory

Database



SQLite

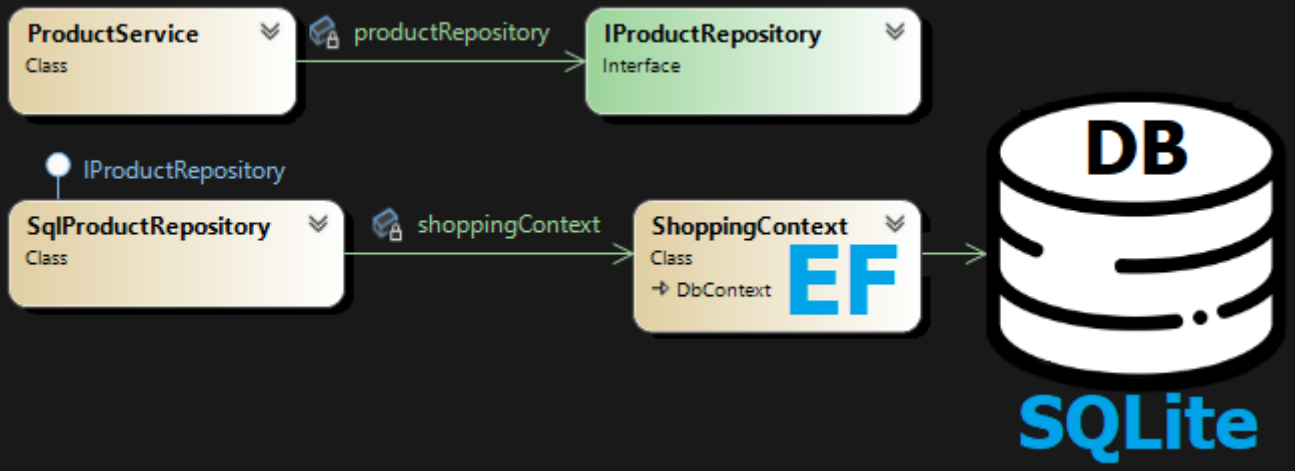
Microsoft.EntityFrameworkCore.Sqlite

`Small` `Fast` `Self-contained` `High-reliability` `Full-featured`

`Maintained by Microsoft` `Nuget`

EF Database Providers

👍 SQLite


ProductService
Class
productRepository
IProductRepository
Interface
IProductRepository
SqlProductRepository
Class
shoppingContext
ShoppingContext
Class
DbContext
EF
DB
SQLite

Demo D07

SQLite

💡 หลักการออกแบบ

`Programming to an Interface`

- Avoid using concrete type and new keyword
- Dependency Injection (IoC)
- Reducing Coupling
- Factory Pattern, Builder Pattern

Summary

1. Isolated Test Environments
2. Test Double → `Mock Object` → `Moq`
3. Testable Code → `เกิดจากความตั้งใจ`
4. Design Principle → `Dependency Inversion`
5. Design Pattern → `Dependency Injection`
6. Tools
    - AutoFixture
    - SQLite