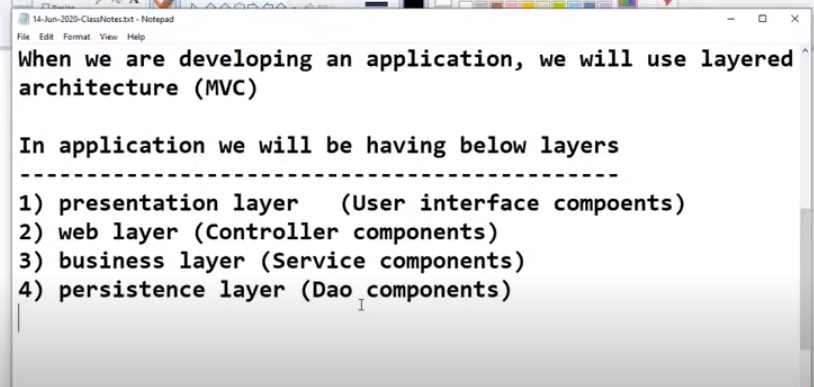
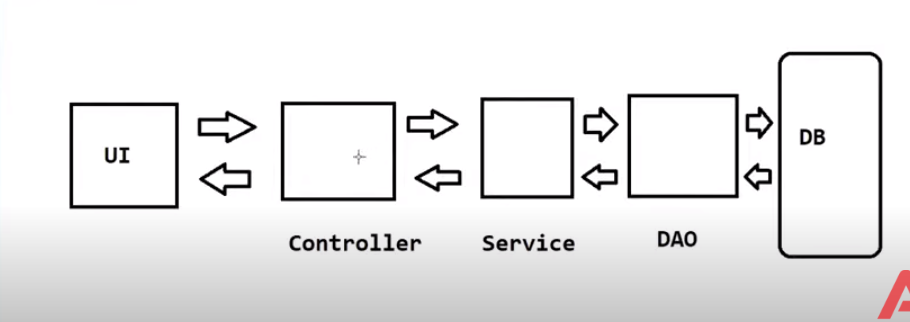
**JUnit and Mockito**

**Youtube link:**

**https://www.youtube.com/watch?v=6khYnHt513c&t=555s**



By using Junit, we can test Controller components, Service components, Dao components. We cannot test UI components using Junit.



The above is MVC architecture.

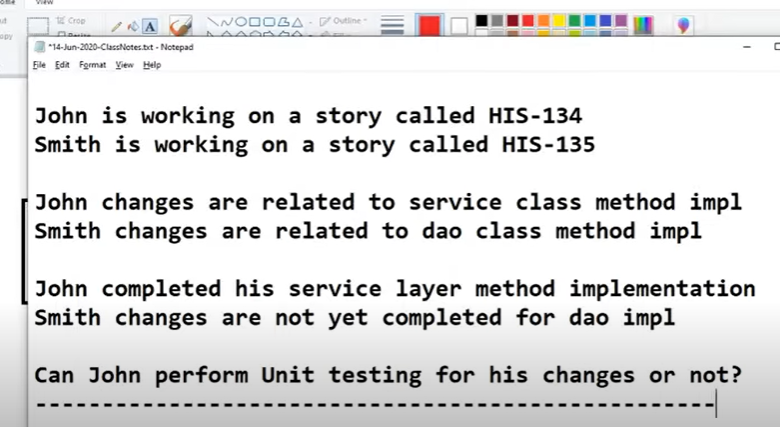
When we want to perform unit testing of components.

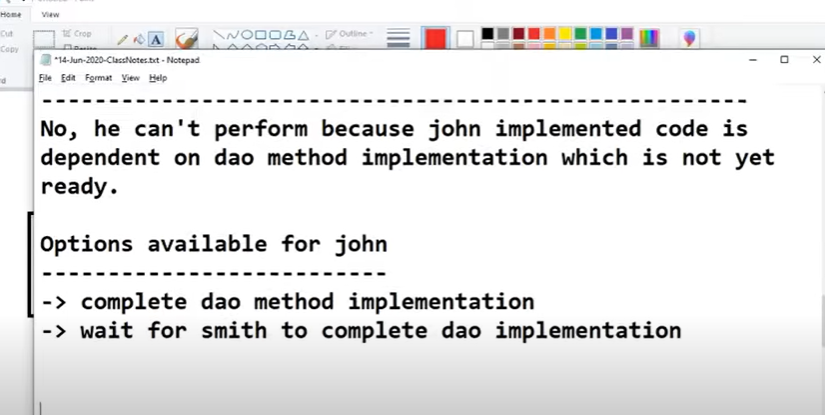
To perform Unit testing of Controller layer, we should have implementation of Service layer.

To perform Unit testing of Service layer, we should have implementation of DAO layer.

To perform Unit testing of DAO layer, we should have implementation of DB

**Scenario1**



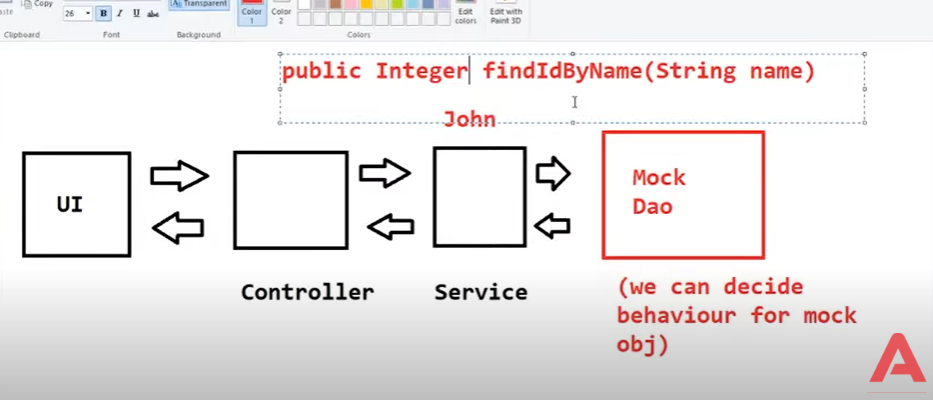


But in real time, one developer does not wait for another developer to do his testing.

Unit testing is testing your components independently.

John has to use DAO layer methods to test his service layer methods.But since DAO layer implementation is not finished, John has to use Mocking.

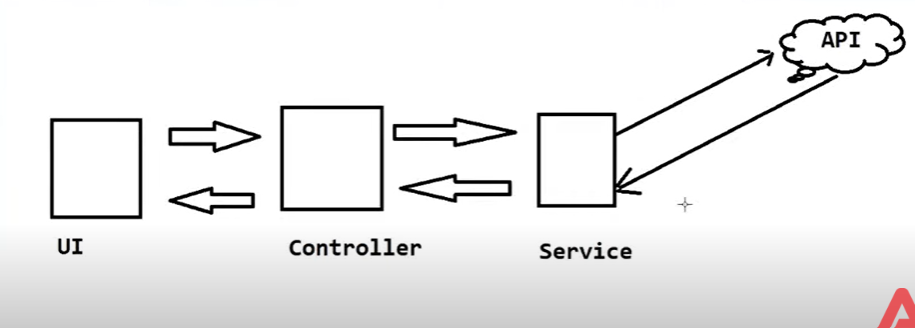
**Mocking:** Mocking is the process of creating Substitute object for Real objects.Mocking is the process of imitating something.



**Scenario 2**

Sometimes, even when DAO layer implementation is available, the developer does not want to test components that does not belong to him. He wants to test only his components in isolated environment.Then in that case he goes for Mocking.

**Scenario 3**



Sometimes the Service layer wants to communicate with an API(service developed by third-party) instead of DAO layer.This is called Web Service communication.The external Web service is developed by some other company.

Imagine in this example, Web service is a weather forecasting service and it is down one day.One day the web service is not working.The service layer logic requires response from external web service.

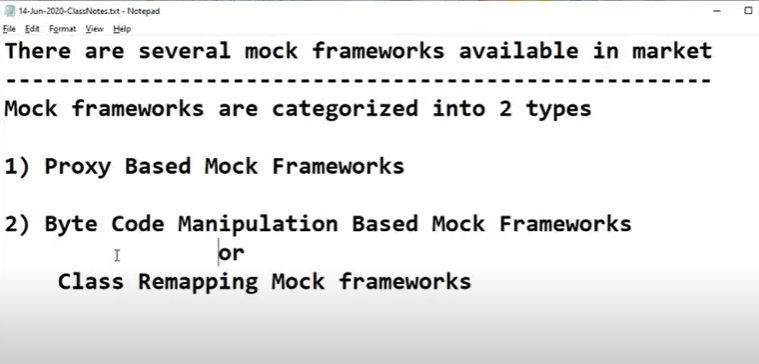


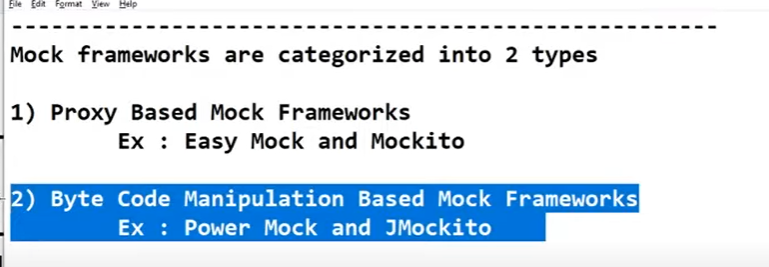
Imagine, in our Service layer implementation, we have written 10 lines of code.At line no 2, we are calling External API. Lines from 3 to 10 gets executed only if line 2 gets executed successfully.

In the real time scenario, we do not have to wait for that external API service response to test our Service layer logic.In this scenario, we go for Mocking.

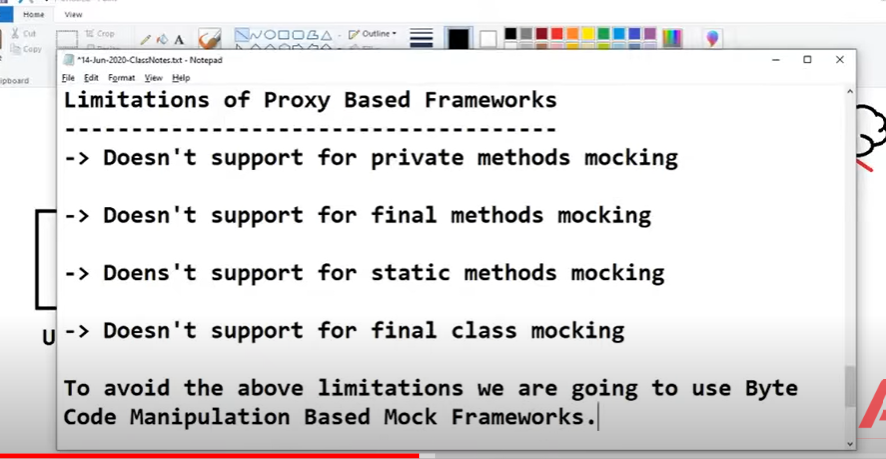
The process of creating Dummy objects by using Mock framework is called Mocking.

Mock frameworks are categorized into 2 types.





In the industry, we are using Byte Code Manipulation based Mock frameworks because they are having advantage over Proxy based Mock frameworks.

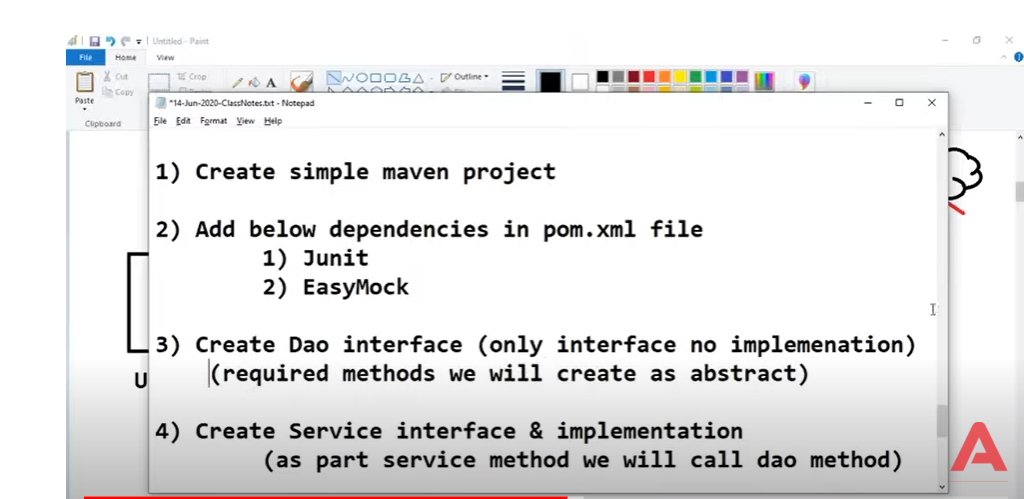


**Steps to develop Application using Junit and EasyMock**

**Junit --Unit Testing Framework.**

**The most common version used in Projects is Junit5 and some projects use Junit4**

**EasyMock --- To perform Mocking**



**Step5)** Create Unit test class for Service class using Junit

Dependencies

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>my-project</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 dependency -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.8.2</version>

<scope>test</scope>

</dependency>

<!-- EasyMock dependency -->

<dependency>

<groupId>org.easymock</groupId>

<artifactId>easymock</artifactId>

<version>4.3</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.2</version>

<configuration>

<includes>

<include>\*\*/\*Tests.java</include>

<include>\*\*/\*Test.java</include>

</includes>

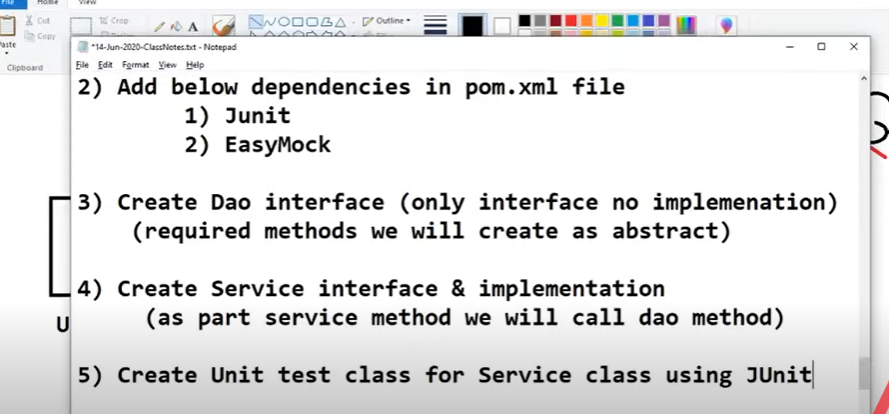
</configuration>

</plugin>

</plugins>

</build>

</project>



To represent a method as a Unit testing method, we use annotation called @Test