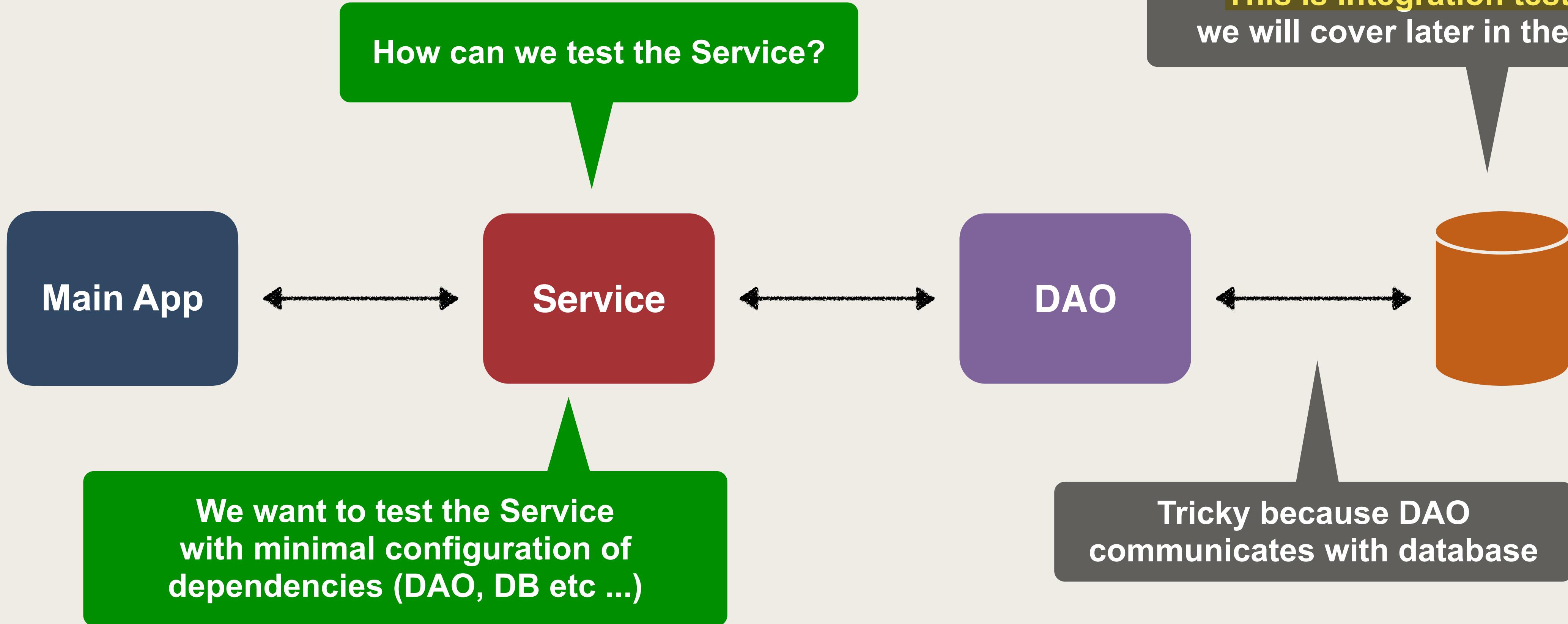


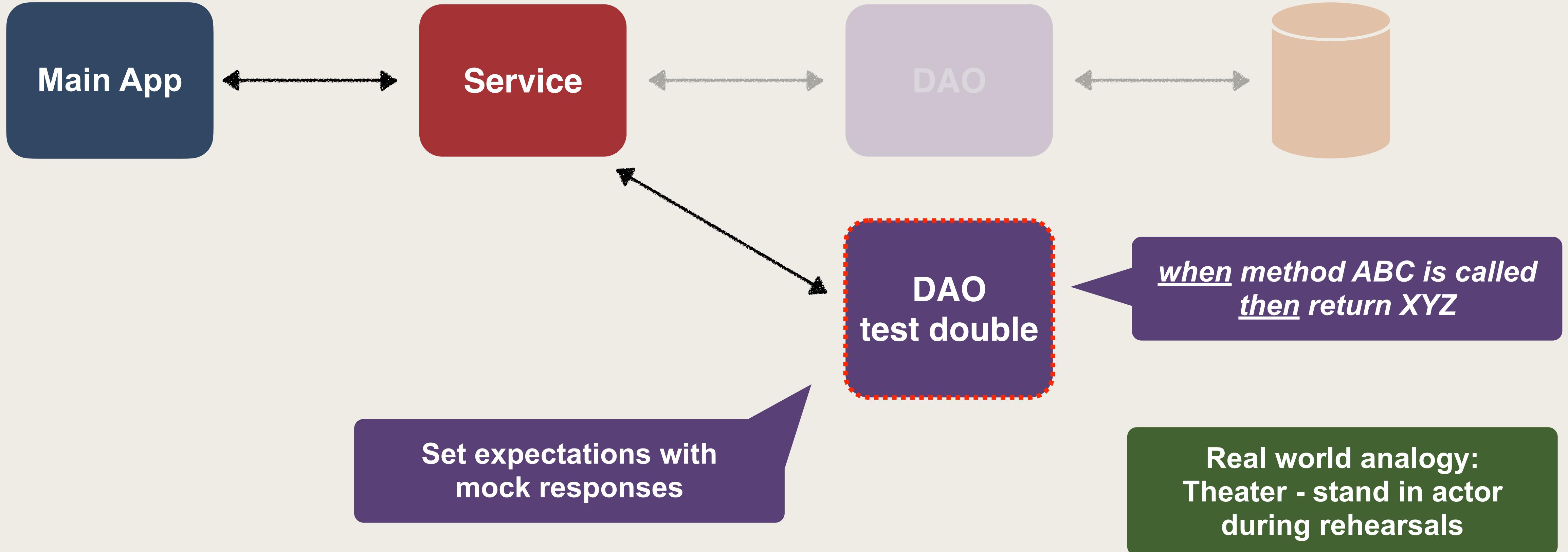
Mocks with Mockito and Spring Boot



Typical Application Architecture



Using a Test Double



**The technique of using test doubles
is known as "mocking"**

Benefits of Mocking

- Allows us to test a given class **in isolation**
- Test interaction between given class and its dependencies
- Minimizes configuration / availability of dependencies
- For example DAO, DB, REST API etc
 - We can mock the DAO to give a response
 - We can mock a REST API to give a response

Real world analogy:
Theater - stand in actor
during rehearsals

Mocking Frameworks

- The Java ecosystem includes a number of Mocking frameworks
- The Mocking frameworks provide following features:
 - Minimize hand-coding of mocks ... leverage annotations
 - Set expectations for mock responses
 - Verify the calls to methods including the number of calls
 - Programmatic support for throwing exceptions

Mocking Frameworks



Name	Website
Mockito	site.mockito.org
EasyMock	www.easymock.org
JMockit	jmockit.github.io
...	

We will use Mockito since
Spring Boot has built-in support for Mockito

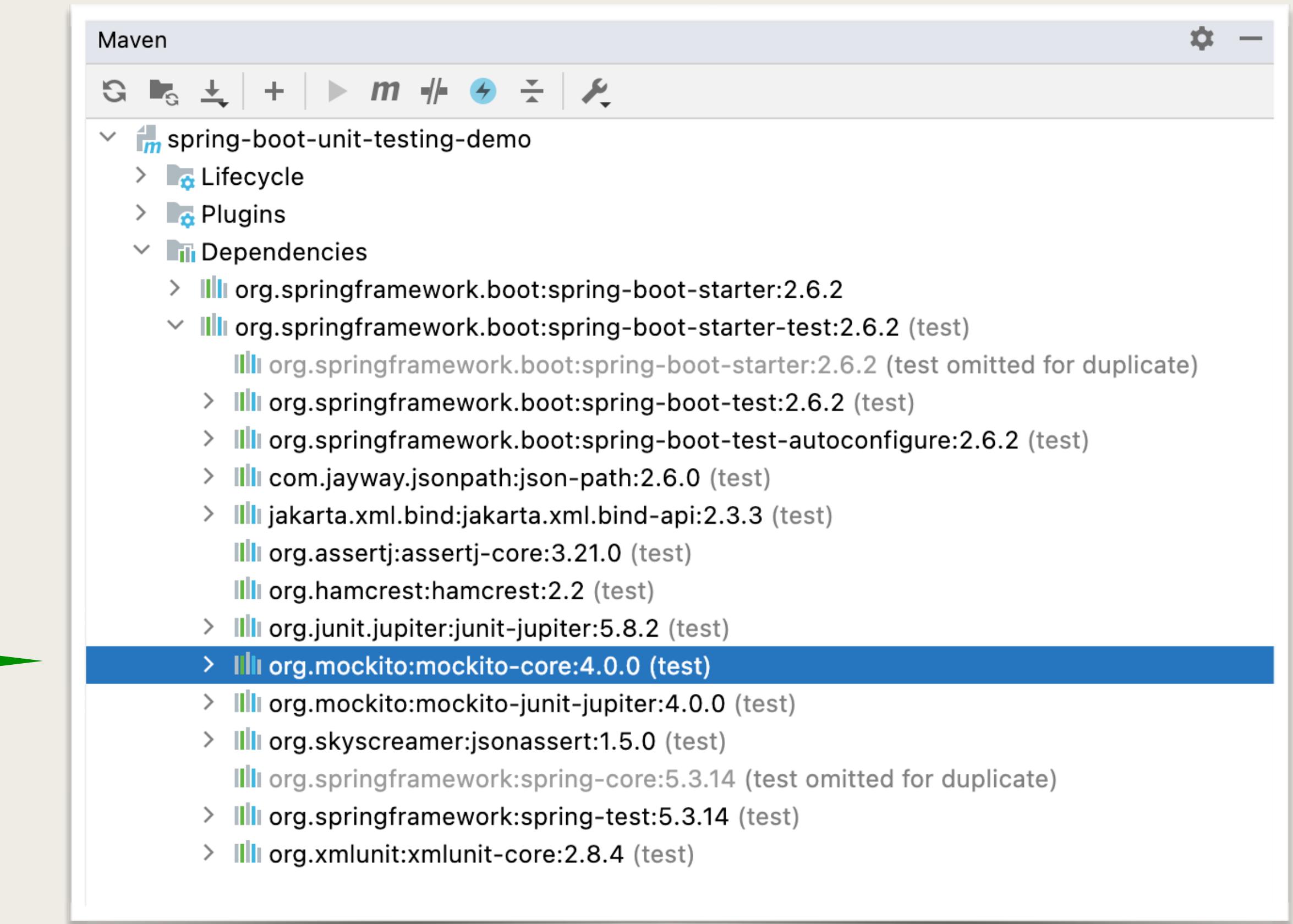
Spring Boot Starter - Transitive Dependency for Mockito

pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

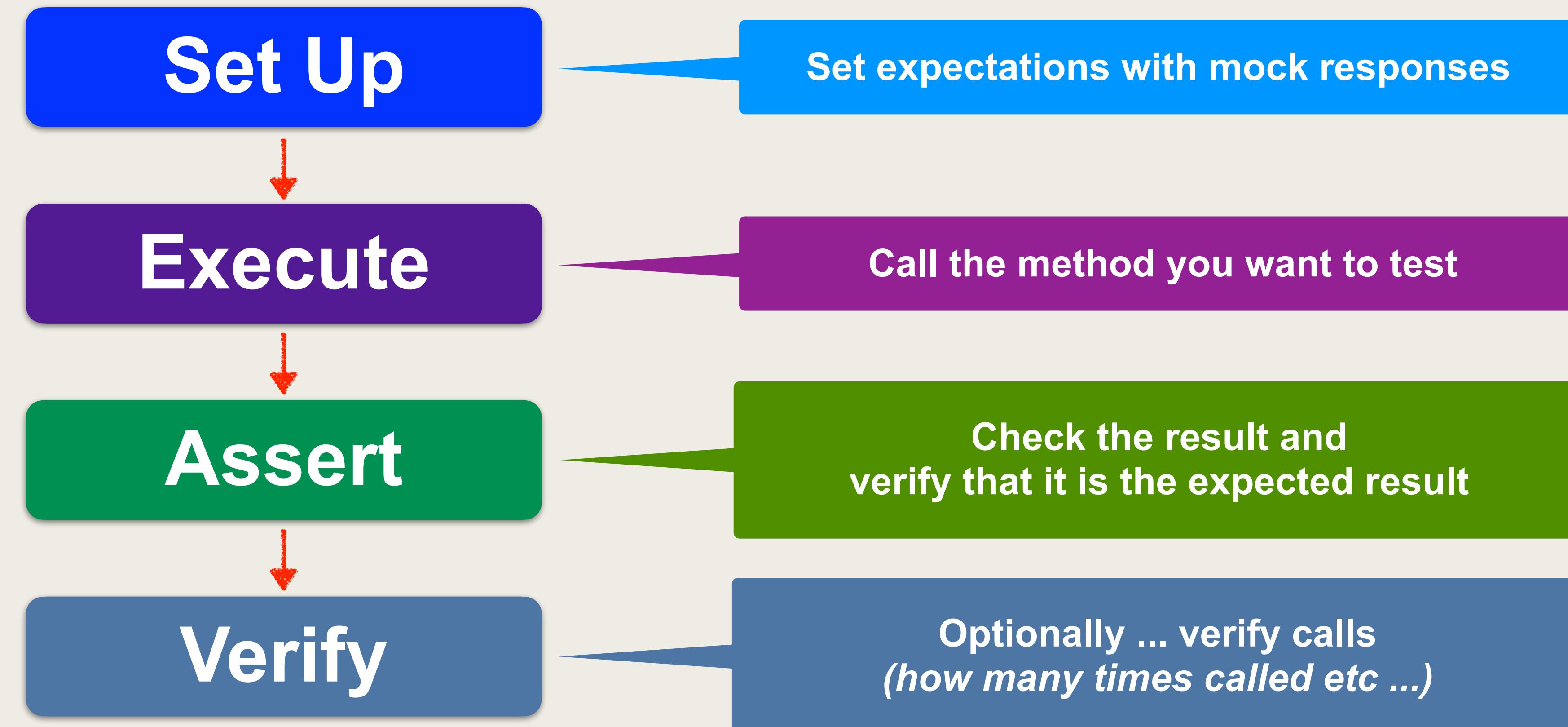
Starter includes a transitive dependency on Mockito

We get it for free :-)

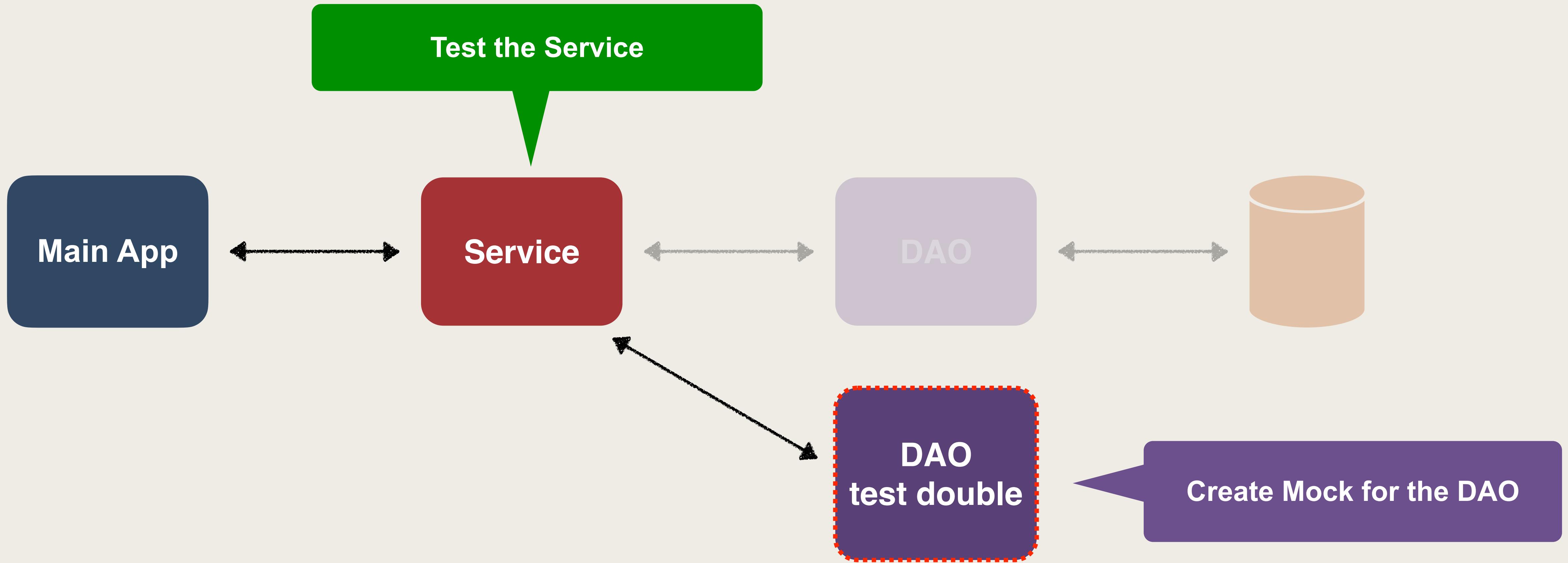


Unit testing with Mocks

- Unit tests with Mocks have the following structure



Testing Plan

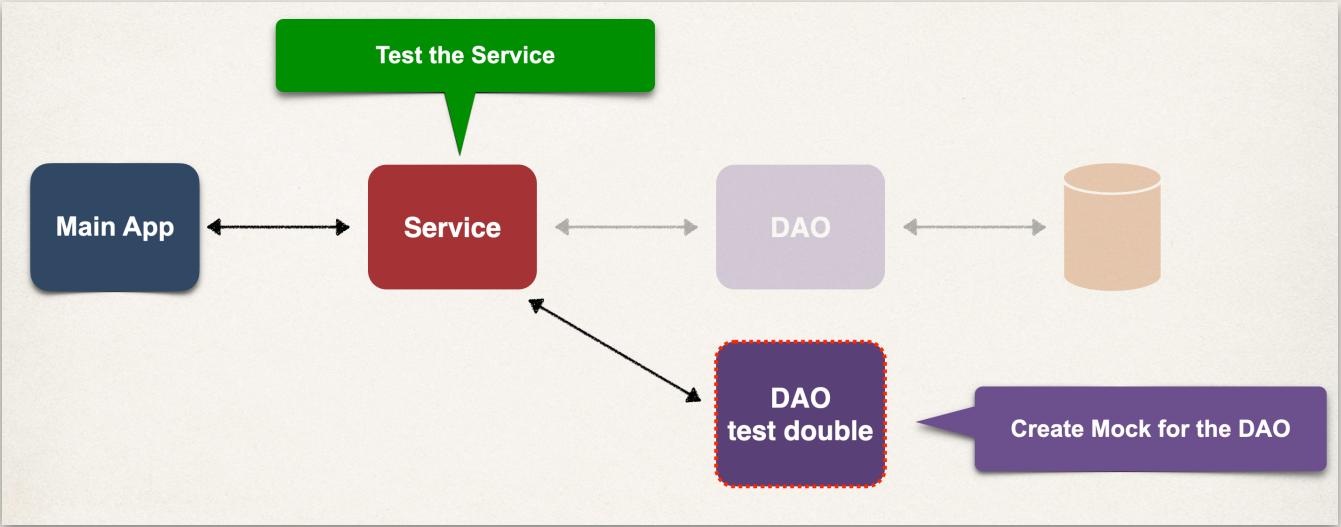


Review code for Service and DAO

ApplicationService.java

```
public class ApplicationService {  
  
    @Autowired  
    private ApplicationDao applicationDao;  
  
    public double addGradeResultsForSingleClass(List<Double> grades) { ... }  
  
    public double findGradePointAverage (List<Double> grades ) { ... }  
  
    public Object checkNull(Object obj) { ... }  
  
}
```

Dependency



ApplicationDao.java

```
public class ApplicationDao {  
  
    public double addGradeResultsForSingleClass(List<Double> grades) { ... }  
  
    public double findGradePointAverage (List<Double> grades ) { ... }  
  
    public Object checkNull(Object obj) { ... }  
  
}
```