2025 — 2026

1TI – 1ACS



Back-End Development

Mocking

N. Jacobs, J. Pieck, E. Steegmans, B. Van Impe, S. Van Peborgh, V. Witters

CONTENTS

- Stubbing
- Mocking
  - Mockito

STUBBING

```
public class PonyRepositoryStub implements PonyRepository {

    @Override
    public <S extends Pony> List<S> saveAll(Iterable<S> entities) {
        return null;
    }

    @Override
    public Optional<Pony> findById(Long aLong) {
        return Optional.empty();
    }

    @Override
    public List<Pony> findAll() {
        return ponies;
    }

    @Override
    public Pony findByName(String name) {
        for (Pony pony : findAll()) {
            if (pony.getName().equals(name)) {
                return pony;
            }
        }
        return null;
    }

    // ...
}
```

- Remember how we wrote a dummy implementation (a stub) for our repository when unit testing with JPA ...

STUB — MOCK

- A stub is useful when we test a class that depends on another class
  - By stubbing the dependency we isolate the class that we want to test
  - But sometimes it is hard to write a stub...

- In this case we can use another technique called *mocking*
  - A *mock* serves the same purpose as a stub
  - But is constructed and used in an entirely different way

- We will use an external library called Mockito to manage our mocks
  - This library is already included by "spring-boot-starter-test"



Mocking

- Mockito

MOCKING

```
public class TestsWithMocking {

    @Test
    public void test() {
        PonyRepository repository = Mockito.mock(PonyRepository.class);

        // ...
    }
}
```

- A mock object
  - Can be created via “Mockito.mock(<class>.class)”
  - Is **a fake object** that mimics the behaviour of a real object in controlled ways

MOCKING

```
@Test
public void test() {
    PonyRepository repository = Mockito.mock(PonyRepository.class);
    Assertions.assertEquals(new ArrayList<>(), repository.findAll());
}
```

- A mock is a fake object of the requested type
  - It does nothing after being initialised
  - Every method called will adhere to the contract of the method in a minimal way

MOCKING

```
@Test
public void test() {
    PonyRepository repository = Mockito.mock(PonyRepository.class);

    List<Pony> ponies = new ArrayList<>();
    ponies.add(new Pony("Bella", 5));
    Mockito.when(repository.findAll()).thenReturn(ponies);

    Assertions.assertEquals(ponies, repository.findAll());
}
```

- A mock is programmable
  - You can specify what it needs to return when a method is called
  - *When* a method is called on the mock, what needs to be returned?

MOCKING

```
@Test
public void givenFindPonyByName_whenPonyExists_thenItIsReturned() {
    PonyRepository repository = Mockito.mock(PonyRepository.class);
    PonyService service = new PonyService(repository);

    // ...

    Pony foundPony = service.findPonyByName("Bella");
    Assertions.assertEquals("Bella", foundPony.getName());

}
```

- We can pass the mocked PonyRepository to the PonyService
- When testing a method of the PonyService, we can program the mock in the exact way that we need the PonyRepository to behave for our test

MOCKING

```
@Test
public void givenFindPonyByName_whenPonyExists_thenItIsReturned() {
    PonyRepository repository = Mockito.mock(PonyRepository.class);
    PonyService service = new PonyService(repository);

    Pony bella = new Pony("Bella", 18);
    Mockito.when(repository.findByName(Mockito.anyString())).thenReturn(bella);

    Pony foundPony = service.findPonyByName("Bella");
    Assertions.assertEquals("Bella", foundPony.getName());

}
```

- We can pass the mocked PonyRepository to the PonyService
- When testing a method of the PonyService, we can program the mock in the exact way that we need the PonyRepository to behave for our test

MOCKING

```
@Test
public void givenFindPonyByName_whenPonyExists_thenItIsReturned() {
    PonyRepository repository = Mockito.mock(PonyRepository.class);
    PonyService service = new PonyService(repository);

    Pony bella = new Pony("Bella", 18);
    Mockito.when(repository.findByName(Mockito.anyString())).thenReturn(bella);

    Pony foundPony = service.findPonyByName("Bella");
    Assertions.assertEquals("Bella", foundPony.getName());

}
```

- The mock will only trigger if the arguments match the *argument captors*
- In this example, the mocked method will return the pony “Bella” with every argument passed to it
  - If the service eventually calls the method with the argument “Bella”, the argument matches with the captor and the pony object is returned

MOCKING

```
@Test
public void givenFindPonyByName_whenPonyDoesNotExist_thenNullIsReturned() {
    PonyRepository repository = Mockito.mock(PonyRepository.class);
    PonyService service = new PonyService(repository);

    Mockito.when(repository.findByName(Mockito.anyString())).thenReturn(null);

    Pony foundPony = service.findPonyByName("Bella");
    Assertions.assertEquals(null, foundPony);
}
```

- We use the mock to fake the dependency of the PonyService
- We can program the mock differently for every test to make sure that we cover all cases of our PonyService code

MOCKING

```
@Test
public void givenANewPony_whenSavingThatPony_thenThePonyIsStoredInTheDatabase() {
    PonyRepository ponyRepository = Mockito.mock(PonyRepository.class);
    OwnerRepository ownerRepository = Mockito.mock(OwnerRepository.class);
    PonyService service = new PonyService(ponyRepository, ownerRepository);

    Pony pony = new Pony("Little Rainbow", 5);
    Mockito.when(ponyRepository.save(pony)).thenReturn(pony);

    service.addPony(pony);

    Mockito.verify(ponyRepository).save(pony);
}
```

- You can even use the mock to verify that certain methods were called as expected and use this to verify behaviour
  - Note that this should be used sparingly, since this test will stop working if the implementation changes

MOCKING

```java
@ExtendWith(MockitoExtension.class)
public class PonyServiceTest {

    @Test
    public void givenFindPonyByName_whenPonyExists_thenItIsReturned() {
        PonyRepository ponyRepository = Mockito.mock(PonyRepository.class);
        OwnerRepository ownerRepository = Mockito.mock(OwnerRepository.class);
        PonyService service = new PonyService(ponyRepository, ownerRepository);

        // ...
    }

    @Test
    public void givenFindPonyByName_whenPonyDoesNotExist_thenNullIsReturned() {
        PonyRepository ponyRepository = Mockito.mock(PonyRepository.class);
        OwnerRepository ownerRepository = Mockito.mock(OwnerRepository.class);
        PonyService service = new PonyService(ponyRepository, ownerRepository);

        // ...
    }
}
```



```java
@ExtendWith(MockitoExtension.class)
public class PonyServiceTest {

    @Mock
    private PonyRepository ponyRepository;

    @Mock
    private OwnerRepository ownerRepository;

    @InjectMocks
    private PonyService service;

    @Test
    public void givenFindPonyByName_whenPonyExists_thenItIsReturned() {
        // ...
    }

    @Test
    public void givenFindPonyByName_whenPonyDoesNotExist_thenNullIsReturned() {
        // ...
    }
}
```

- The mock setup can be simplified if all tests in the test-file instantiate the same object to test with identical mocks as dependencies

MOCKING

```
@ExtendWith(MockitoExtension.class)
public class PonyServiceTest {

    @Mock
    private PonyRepository ponyRepository;

    @Mock
    private OwnerRepository ownerRepository;

    @InjectMocks
    private PonyService service;

    //…

}
```

- Mocking support is enabled by writing @ExtendWith(MockitoExtension.class) above the test class
- @Mock these mock objects simulate the behavior of real objects but allow us to define their behavior and responses during testing.
- @InjectMocks is used to inject mocked dependencies into the class under test. It injects mock objects into the fields marked with @Mock annotations in the test class.

MOCKING

```
@Test
public void givenFindPonyByName_whenPonyExists_thenItIsReturned() {
    PonyRepository repository = Mockito.mock(PonyRepository.class);
    PonyService service = new PonyService(repository);

    Pony bella = new Pony("Bella", 18);
    Mockito.when(repository.findByName(Mockito.anyString())).thenReturn(bella);

    Pony foundPony = service.findPonyByName("Bella");
    Assertions.assertEquals("Bella", foundPony.getName());

}
```



With use of @Mock and @InjectMocks

```
@Test
public void givenFindPonyByName_whenPonyExists_thenItIsReturned() {
    Pony bella = new Pony("Bella", 18);
    Mockito.when(repository.findByName(Mockito.anyString())).thenReturn(bella);

    Pony foundPony = service.findPonyByName("Bella");
    Assertions.assertEquals("Bella", foundPony.getName());

}
```

MOCKING

- There are many things that you can program on a mock
  - when(listMock.add(anyString())).thenReturn(false);
  - when(listMock.add(anyString())).thenThrow(IllegalStateException.class);
  - when(listMock.add(anyString()))
    .thenReturn(false)
    .thenThrow(IllegalStateException.class);
  - verify(ponyRepositoryMock).save(pony)
  - …

- Refer to the Mockito documentation for a comprehensive overview

EXERCISES @CLASS

- exercise1
- exercise2



EXERCISES @HOME

- exercise3



