mockito

O que é e para que serve?

- Mockito é um framework de testes para aplicações, seu principal objetivo é simular a instância de classes e comportamentos de métodos.
- Ao mockar uma dependência, podemos fazer com que a classe que será testada apenas simule o método testado e suas dependências.

Habilitando as anotações

Para que os testes possam rodar com o Mockito, a classe de teste deve estar configurada com a anotação **@RunWith** que faz referência a classe ***MockitoJUnitRunner.class*** conforme exemplo:

```
@RunWith(MockitoJUnitRunner.class)
public class EmployeeServiceTest {}
```

- Outra maneira, antes do teste:

```
@Before
public void setup() {
        MockitoAnnotations.initMocks(this);
}
```

Resumo das principais funções

- **Mock:** cria uma instância de uma classe (mockada). Se um método for chamado, ele será apenas simulado.
- **Spy:** cria uma instância de uma classe onde pode ser mockada ou chamar métodos desta classe. É uma alternativa ao InjectMocks (quando é preciso mockar métodos da própria classe que está sendo testada);
- **InjectMocks:** cria uma instância e injeta as dependências necessárias (que serão anotadas com @Mock);
- **Verify:** verifica a quantidade de vezes e quais parâmetros foram utilizados para acessar determinado método;
- **When:** após a criação do mock, pode ser configurado ações de chamada e de retorno;
- **Matchers:** permite verificações por meio de matchers de argumentos (anyObject(), anyString(), etc).

Mock

- Mock é um objeto criado em tempo de execução que devolve respostas pré-configuradas. São uteis em situações em que o código que desejamos testar possui dependências de outros objetos (outras classes da aplicação, interfaces, etc), e os detalhes dessas dependências não são importantes para o teste em questão. Queremos testar somente o código sem depender do comportamento de outros objetos ou recursos da infraestrutura.
- Criando um mock:

```
@Mock
private EmployeeRepository employeeRepository;
```

Spy

- Um Spy é essencialmente um objeto que “engloba” um instância real do tipo “espionado”, de modo que podemos utilizar normalmente esse objeto com o seu comportamento verdadeiro (é uma diferença fundamental em relação ao mock, que apenas sabe fazer o que lhe é “ensinado”).
- Com o Spy podemos utilizar tanto o comportamento real do objeto quanto configurar (mockar) os métodos que forem necessários para o nosso teste.

InjectMock

- O atributo anotado com @InjectMock será instanciado pelo Mockito, e todos os atributos anotados com @Mock e @Spy que serão considerados dependências desse objeto.
- Exemplo de uso:

```
@InjectMock
private EmployeeService employeeService;
```

Verify

- Como o próprio nome sugere, o verify verifica se aquele método, daquele mock, foi chamado com aqueles argumentos, se não, é lançada uma exceção (quebrando o teste). Essa exceção tem uma mensagem bastante explicativa informando como a chamada do método realmente ocorreu.
- Por padrão o verify confirma se o método em questão foi invocado apenas uma vez
- Exemplo de uso:

```
verify(service/repository).metodo(argumento);
```

When

- Através deste método é possível simular chamadas a recursos externos a classe, como acesso a banco de dados por exemplo, porém sem afetar o mesmo (que não sofre alterações) e sem preocupar-se em como funcionará a consulta.
- O método When diz essencialmente ao Mockito:

when(service.método(argumento))thenReturn(resultado)

Quando o **método** da **service** for invocado com um **argumento**, então devolva o **resultado**.

O método When oferece ainda alguns recursos adicionais, as Matchers.

Matchers

- Argument matchers permitem maior flexibilização em situações onde os argumentos do método mockado não são importantes no teste, ou simplesmente onde não é possível prever os valores.
- Exemplo de uso:

```
when(service.método(anyString()))thenReturn(resultado)
```

Neste caso estamos dizendo: quando o método for invocado com uma String qualquer, devolva o resultado.

Alguns exemplos de matchers:

anyInt(), any(Class), isNull(), notNull(), same()

Para que o Mockito não serve?

- Em cenários de testes, quando não for necessário fazer nenhum mock de dependências, ou seja, quando a classe a ser testada não depende de nenhuma outra classe ou repositório por exemplo.

Hands On

Obrigado!