Usando o verify do Mockito:

Além de poder definir os comportamentos dinamicamente, **os mocks permitem se a interação com o objetos mockados foram realizados conforme o esperado.**

Vamos aprender isso através de uma nova funcionalidade, o sistema vai informar por email os usuários com locações atrasadas. Para isso **vamos tunar nossa interface de DAO**, adicionando um **método que vai retornar uma lista de locações pendentes**:

```
public List<Locacao> obterLocacoesPendentes();
```

Claro que é um **método sem corpo**, afinal de contas vem de **uma interface** e portanto ele é **abstrato** e quem deve **implementa-la é a classe responsável por fazer essas consultas**, nós estamos apenas a **utilizando para verificar nossa lógica.**

Também vamos criar **uma nova interface de serviço**, que será a **EmailService**, ela **quem deverá notificar o usuário**, então colocamos o método abstrato de notificar o usuario **recebendo como parâmetro um usuario**:

```
public interface EmailService {

    public void notificarAtraso(Usuario usuario);
}
```

Perfeito, agora a gente pode criar o **método na LocacaoService** que simplesmente irá pegar **uma lista de locações pendentes**(vinda do dao) e para **cada uma delas** acionará a **interface de email** que receberá o **usuário da locação** e **enviará a notificação**:

```
public void notificarAtrasos() {

    List<Locacao> locacoes = dao.obterLocacoesPendentes();

    for(Locacao locacao: locacoes) {
        emailService.notificarAtraso(locacao.getUsuario());
    }
}
```

O teste:

Vamos criar o teste, **primeiro montamos o cenário (Given),** precisamos de **uma lista de locações onde a data de retorno esteja atrasada**. Então **precisamos** de um **usuarioComLocacaoAtrasada** e uma lista de **locacoesPendentes com esse usuario,** usamos os builders para criar o cenário de forma bem legível e ja **informamos ao mockito o que o objeto**

mockado deve retornar quando **chamarmos o método de obterLocacoesPendentes():**

```
public void deveEnviarEmailAUsuarioComLocacaoAtrasada() {
    //GIVEN/Dado que
    Usuario usuarioComLocacaoAtrasada = UsuarioBuilder.umUsuario().agora();
    List<Locacao> locacoesPendentes = Arrays.asList(
            LocacaoBuilder.umLocacao()
                .comUsuario(usuarioComLocacaoAtrasada)
                .comDataRetorno(DataUtils.obterDataComDiferencaDias(-2))
            .agora());

    Mockito.when(dao.obterLocacoesPendentes()).thenReturn(locacoesPendentes);
```

Como nosso método de **notificarAtraso( )** ira receber **essa lista e iterar ela mandando um email para cada usuario da locacao** não seria interessante retornar a lista de usuario que mandamos o email, isso sairia do escopo do teste.

Nosso **resultado esperado é somente que o email seja enviado,** para checar o envio usamos o **Mockito com o método verify().** Então, **quando eu chamar o serviceDeLocacao.notificarAtrasos()** o **Mockito irá verificar se o emailService** notificou o atraso para o **usuarioComLocacaoAtrasada**:

```
    //WHEN/Quando
    servicoDeLocacao.notificarAtrasos();

    //THEN/Então
    Mockito.verify(emailService).notificarAtraso(usuarioComLocacaoAtrasada);
}
```

O escopo do **verify( )** acaba apenas na variável de instância do EmailService, depois disso temos o método dessa variável junto com o parâmetro. Ou seja, ele é diferente da **gravação de expectativa** quado **usamos o when( ).** Para deixar mais claro, podemos até usar o verify no teste de "**naoDeveAlugarFilmeParaNegativadoSPC()":**

```
168         //WHEN/QUANDO
169         servicoDeLocacao.alugarFilme(caraTentandoAlugar, listaDeFilmes);
170
171         //verificação
172         Mockito.verify(spcService).possuiNegativacao(caraTentandoAlugar);
173     }
```

uns: 1/1 Errors: 0 Failures: 0

Legal, mas temos um problema, se eu instânciar um outro usuário para saber se o teste não esta dando falso positivo, então nós temos um falso positivo:

```
Usuario caraTentandoAlugar = UsuarioBuilder.umUsuario().agora();
Usuario usuario2 = UsuarioBuilder.umUsuario().agora();
List<Filme> listaDeFilmes = Arrays.asList(
        FilmeBuilder.umFilme().agora(),
        FilmeBuilder.umFilme().agora());

Mockito.when(spcService.possuiNegativacao(caraTentandoAlugar)).thenReturn(true);

//WHEN/QUANDO
servicoDeLocacao.alugarFilme(caraTentandoAlugar, listaDeFilmes);

//verificaçaõ
Mockito.verify(spcService).possuiNegativacao(usuario2);
```

,212 seconds

Errors: 0 Failures: 0

Isso acontece **pela maneira que estamos checando a Exception**(nesse caso a maneira elegante), se eu **quisesse fazer a verificação de que o método do spcService foi de fato chamado** eu teria que usar **a maneira robusta de se tratar uma exceção em um teste**:

```
    //WHEN/QUANDO
    try {
        servicoDeLocacao.alugarFilme(caraTentandoAlugar, listaDeFilmes);
        Assert.fail();

        //verificaçaõ

    } catch (LocadoraException e) {
        Assert.assertThat(e.getMessage(), is("Usuário esta negativado no SPC"));
    }

    Mockito.verify(spcService).possuiNegativacao(caraTentandoAlugar);
}
```

1/1 Errors: 0 Failures: 0

Refatorando para o **método robusto conseguimos fazer a verificação de que o serviço do SPC foi de fato chamado** para o **caraTentandoAlugar**, se **não tivessemos mudado** o parâmetro que estava **usuario2(aquele para forçar o erro do teste)** então teriamos **falhado.**

Temos que ter cautela ao usar o verify, analisar onde realmente importa sua utilização.