

## Capturando argumentos:

Argument captor é um recurso do mockito que permite deixar o uso dos mocks ainda mais dinâmicos.

Para ilustrar bem como **a captura de argumentos funcionam**, vamos criar uma **nova funcionalidade** para nosso sistema. Vamos **poder prorrogar uma locação**:

```
public void prorrogarLocacao(Locacao locacao, int dias) {  
    Locacao novaLocacao = new Locacao();  
    novaLocacao.setUsuario(locacao.getUsuario());  
    novaLocacao.setFilmes(locacao.getFilmes());  
    novaLocacao.setDataLocacao(new Date());  
    novaLocacao.setDataRetorno(DataUtils.obterDataComDiferencaDias(dias));  
    novaLocacao.setValor(locacao.getValor() * dias);  
    dao.salvar(novaLocacao);  
}
```

Esse método **recebe uma locacao por parâmetro e a quantidade de dias** que será prorrogada. O **método não tem retorno**, nós apenas **criamos uma nova locação baseada na locação recebida** e mudamos **apenas alguns atributos**, como a **data e valor e depois salvamos a locação**.

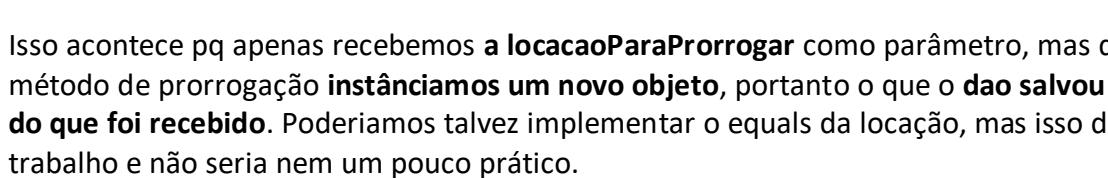
## O teste:

O cenário(Given está configurado) :

```
@Test  
public void deveProrrogarUmaLocacao() {  
    //GIVEN  
    Locacao locacaoParaProrrogar = LocacaoBuilder.umaLocacao().agora();  
  
    //WHEN  
    servicoDeLocacao.prorrogarLocacao(locacaoParaProrrogar, 3);  
  
    //THEN  
}
```

Agora é hora de **fazer a verificação**, mas **o método que prorroga uma locação é void** então não retorna a nova locação, **apenas a salva no final chamando o dao**. Se usarmos o verify para ver se nosso dao.salvar( ) foi acionado teremos o seguinte resultado:

```
//THEN
Mockito.verify(dao).salvar(locacaoParaProrrogar);
```



Isso acontece pq apenas recebemos a **locacaoParaProrrogar** como parâmetro, mas dentro do método de prorrogação **instanciamos um novo objeto**, portanto o que o **dao salvou é diferente do que foi recebido**. Poderíamos talvez implementar o equals da locação, mas isso daria muito trabalho e não seria nem um pouco prático.

Então o que faremos é **usar o ArgumentCaptor** para **capturar o que foi enviado para o dao.salvar()**. Antes da verificação do método de salvar, instanciamos um ArgumentCaptor e falamos para qual classe ele será usado:

```
//THEN
ArgumentCaptor<Locacao> argCapt = ArgumentCaptor.forClass(Locacao.class);
```

Agora pedimos para ele **capturar o que foi passado para o dao.salvar()**:

```
//THEN
ArgumentCaptor<Locacao> argCapt = ArgumentCaptor.forClass(Locacao.class);

Mockito.verify(dao).salvar(argCapt.capture());
```

Agora precisamos **receber o que foi capturado** pelo **argCapt.capture()**:

```
Locacao locacaoCapturada = argCapt.getValue();
```

Pronto, agora temos a locação que foi passada dentro do método, agora precisamos apenas fazer a checagem(Then) usando os valores que a locacaoCapturada capturou:

```
//THEN
erro.checkThat(locacaoCapturada.getValor(), is(36.00));
erro.checkThat(locacaoCapturada.getDataLocacao(), is(MatchersProprios.ehHoje()));
erro.checkThat(locacaoCapturada.getDataRetorno(),
    is(MatchersProprios.ehHojeComDiferencaDeDias(3)));
```

