Mocks – Java

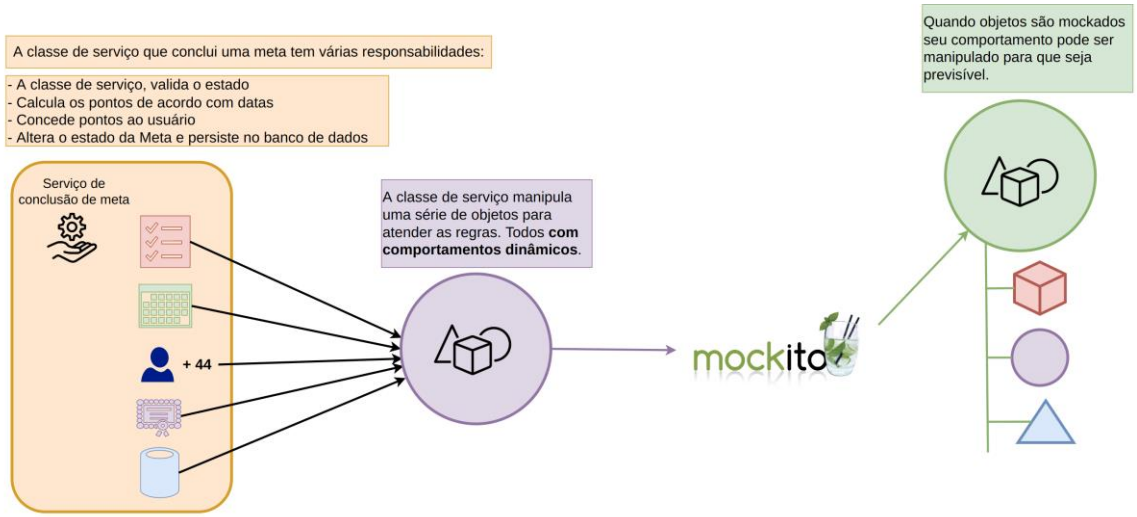
Para que testes **sejam realmente unitários** é necessário **garantir que nenhuma interferência "externa" possa agir no método**, entenda interferência externa como: *uma busca no banco de dados, uma classe atrelada que complementa uma lógica maior e etc.*

Pra impedir a interferência externa usamos o **mock**, a ideia de **mockar** é justamente **simular/fingir/imitar** um comportamento de uma classe/método **sem UTILIZAR A CLASSE REAL, O MÉTODO REAL** e etc...

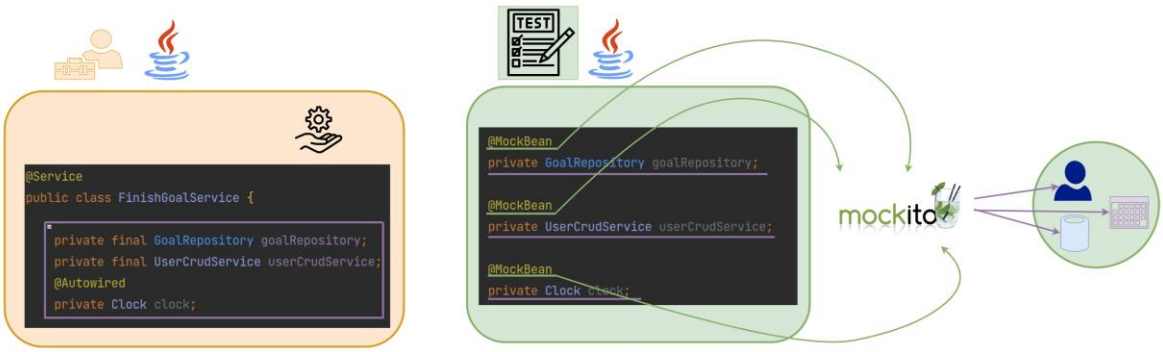
Objetos mockados – Java

Mockito é um framework para mock de teste. O mock é feito em **objetos específicos que são necessários para a conclusão de um teste**, então a ideia é **mockar o objeto e manipular seus retornos quando forem chamados**, sem utilizar a chamada real.

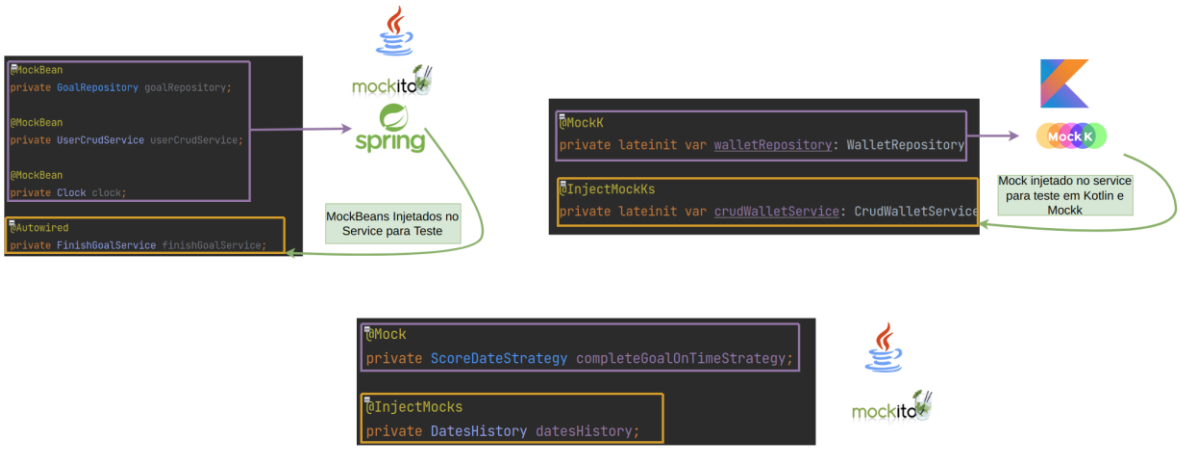
Por exemplo, uma **classe de serviço de conclusão de meta** precisa fazer diversas operações dinâmicas, como dar ***pontos baseado em data de conclusão***, como eu posso manter meus testes sempre controlados para evitar que fatores externos interfiram na execução deles(como datas):



Os **objetos de dependência** da **Classe de serviço** agora **ficam sendo Mocks**:



Os **mocks precisam ser injetados** na **classe alvo** do teste:



E **seus comportamentos podem ser alterados quando for necessário**, fazendo com que **o resultado seja sempre previsível:**

Capturando argumentos – Java

Talvez seja necessário **capturar um argumento que esteja dentro do método**. Por exemplo, um **método imutável cria um novo objeto a partir daquele passado por parâmetro** e **persiste ele no banco de dados.**

Como eu poderia fazer para **recuperar esse argumento/objeto criado dentro do escopo do método** para comparar seus valores e fazer o teste? Da pra capturar ele:

O que o mockito NÃO FAZ – Java

Mockar construtores de objetos, alterar comportamento de métodos “static” e privados.