JUnit: Implementando testes unitários em Java

Um pouco de XP

XP é um apelido carinhoso de uma metodologia ágil de desenvolvimento designada Extreme Programming, com foco em agilidade de equipes e qualidade de projetos, apoiada em valores como simplicidade, comunicação, feedback e coragem.

XP é uma metodologia muito comportamental, onde prima mudanças de atitudes e práticas. Sua principal mudança está na máxima integração entre pessoas e, principalmente, estimulando uma participação maior do cliente. Portando, literalmente, temos que “FAZER O PROGRAMA COM O CLIENTE”.

Práticas de XP

XP sugere um conjunto de boas práticas que melhoram o planejamento, execução, e gerenciamento de seu projeto de software.

Essas práticas (Ver na **figura 1**) melhoram sua eficiência e eficácia, diminuindo o retrabalho, garantindo dessa forma a qualidade do seu projeto.

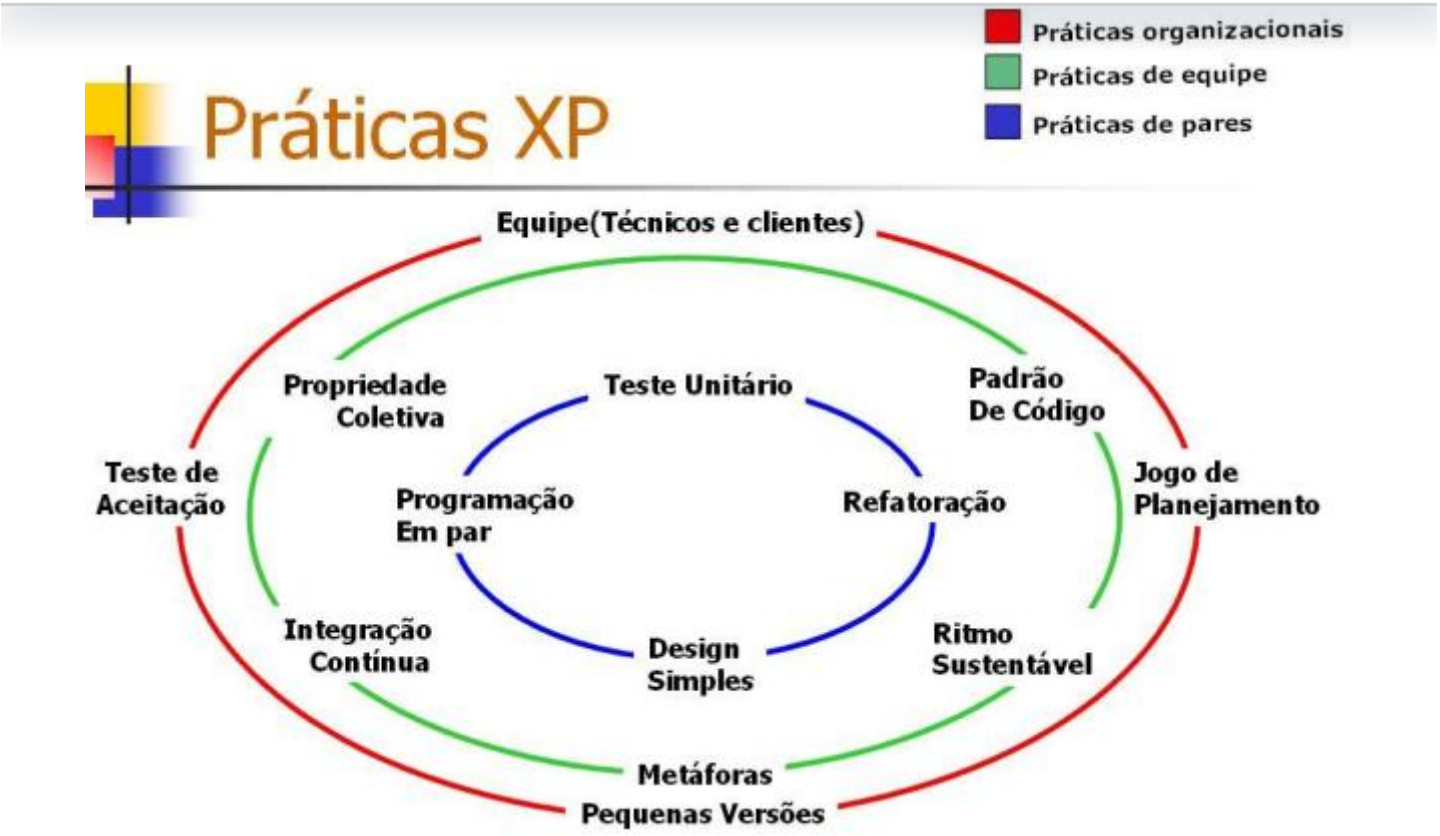


Figura 1. Exemplo de práticas XP a nível organizacional, de equipes e de pares.

Curso relacionado: O que é JUnit?

TDD - Test Driven Development

O conceito de Desenvolvimento Guiado por Testes define que antes de criarmos um código novo (classe), devemos escrever um teste (classe de test case) para ele. Essa prática traz vários benefícios às equipes de desenvolvimento e inclusive estes testes serão usados como métrica em todo o tempo de vida do projeto. Veja

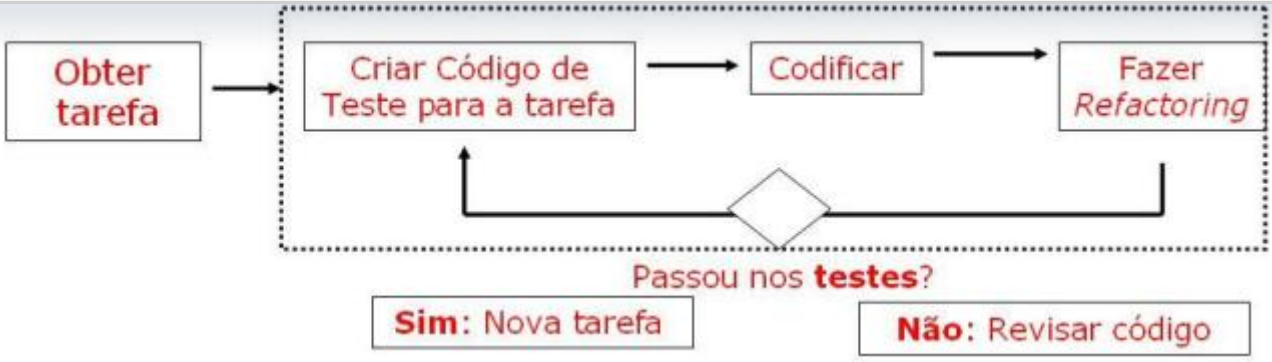


Figura 2. Processo de testes unitários dentro de seu projeto.

O que são Testes Unitários?

Imagine por exemplo, se um avião só fosse testado após a conclusão de sua construção, com certeza isso seria um verdadeiro desastre, é nesse ponto que a engenharia aeronáutica é uma boa referência em processos de construções de projetos de software, principalmente em sistemas de missão crítica, pois durante a construção e montagem de um avião todos os seus componentes são testados isoladamente até a exaustão, e depois cada etapa de integração também é devidamente testada e homologada.

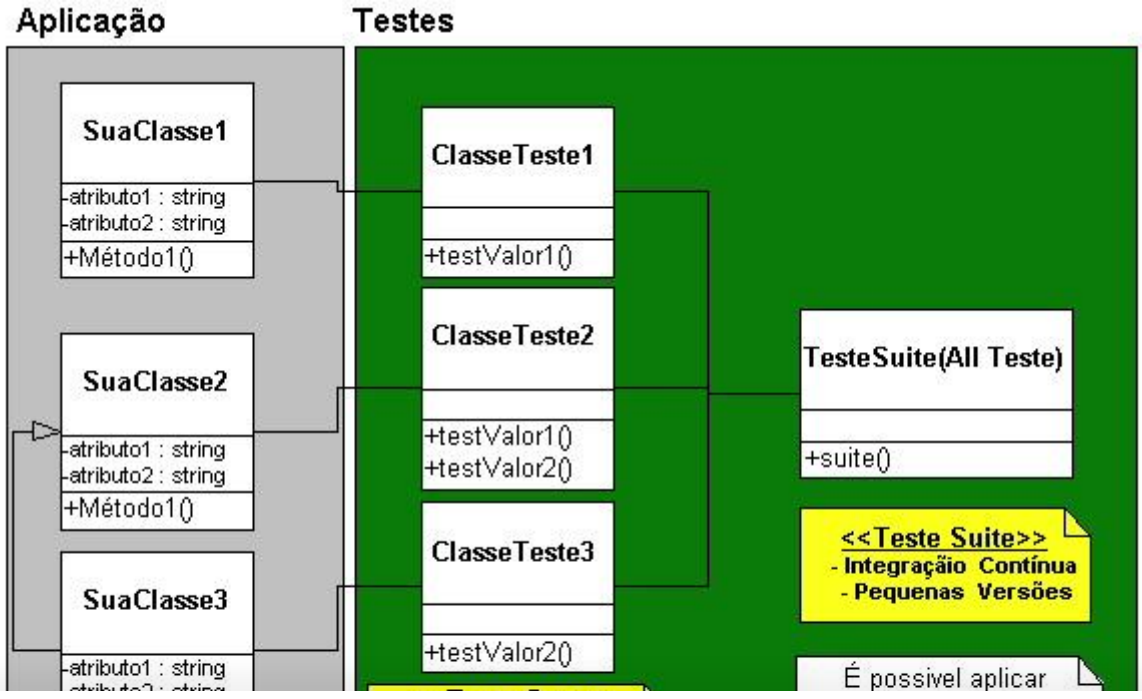
O teste unitário, de certa forma se baseia nessa ideia, pois é uma modalidade de testes que se concentra na verificação da menor unidade do projeto de software. É realizado o teste de uma unidade lógica, com uso de dados suficientes para se testar apenas à lógica da unidade em questão.

Em sistemas construídos com uso de linguagens orientadas a objetos, essa unidade pode ser identificada como um método, uma classe ou mesmo um objeto.

- Previne contra o aparecimento de “BUG’S” oriundos de códigos mal escritos;
- Código testado é mais confiável;
- Permite alterações sem medo (coragem);
- Testa situações de sucesso e de falha;
- Resulta em outras práticas XP como: Código coletivo, refatoração, integração contínua;
- Serve como métrica do projeto (teste ==requisitos);
- Gera e preserva um “conhecimento” sobre as regras de negócios do projeto.

Organização dos testes e práticas XP no Teste Unitário

Abaixo na figura 3, temos um diagrama que mostra a forma como as classes de testes ficam organizadas em um projeto codificado em Java e a correlação com algumas práticas XP.



Quando fazer Teste Unitário?

No início: Primeiro projetar e escrever as classes de testes, depois as classes com regra de negócios.

Diariamente: É SUGERIDO que seja rodado os testes várias vezes ao dia (é fácil corrigir pequenos problemas do que corrigir um "problemão" somente no final do projeto).

Quem faz o Teste Unitário?

Test Case (para cada classe): Desenvolvedor (Projeta, escreve e roda).

Test Suite(Roda vários test cases): Coordenador e Desenvolvedor (Projeta, escreve e roda).

Vale lembrar que o Teste de aceitação (homologação) é feito junto ao cliente.

Outra visão nova, interessante e muito polêmica, é a aproximação da responsabilidade dos testes ao programador, o que em algumas outras abordagens metodológicas é feito somente por equipes separadas, como por exemplo, uma equipe de teste/homologação.

Porém esse contexto é a base de qualquer metodologia ágil, pois dessa forma, o próprio programador, ao criar e executar os testes, adquiri um controle maior e imediato na prevenção e correção de bugs, contribuindo substancialmente para redução do tempo de vida de um projeto.

Teste Unitário: O que testar?

- A principal regra para saber o que testar é: “Tenha criatividade para imaginar as possibilidades de testes”;
- Comece pelas mais simples e deixe os testes “complexos“ para o final;
- Use apenas dados suficientes (não teste 10 condições se três forem suficientes);
- Não teste métodos triviais, tipo get e set;
- No caso de um método set, só faça o teste caso haja validação de dados;
- Achou um bug? Não conserte sem antes escrever um teste que o pegue (se você não o fizer, ele volta).

Para fixar bem essas dicas, na figura 4, temos um exercício de imaginação, onde você deverá achar as possibilidades de testes neste diagrama de classe.

Claro que não existe uma resposta única para esse exercício, pois você, baseando-se em suas experiências anteriores e sua criatividade, pode ter várias visões acerca dessa classe.

Agora sim, depois dessa pequena conceituação acima sobre testes unitários, você verá a partir desse tópico que o **JUnit é um framework que facilita o desenvolvimento e execução de testes unitários em código Java**.

Ele Fornece uma completa API (conjunto de classes) para construir os testes e Aplicações gráficas e em modo console para executar os testes criados.

Por que usar JUnit?

Os principais motivos que favorecem o uso desse framework são:

- JUnit pode verificar se cada unidade de código funciona da forma esperada;
- Facilita a criação, execução automática de testes e a apresentação dos resultados;
- É Orientado a Objeto;
- É free e pode ser baixado em: www.junit.org

Como instalar o JUnit?

Para **usar o JUnit em sua máquina** basta ter em mente essas duas idéias:

- Caso você **não tenha o JUnit instalado**, faça o download do arquivo `junit.jar` em www.junit.org, após inclua-o no classpath para compilar e rodar os programas de teste.
- Porém o JUnit já vem configurado nas versões recentes de IDE's como Eclipse, NetBeans, JBuilder, BlueJ e outros.

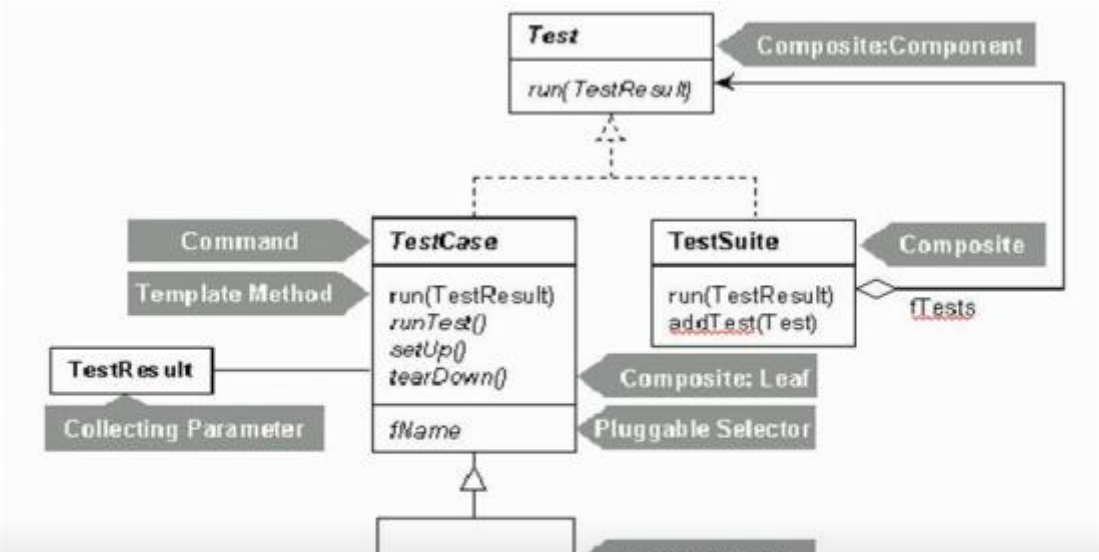
Planejando os testes no JUnit

A lista abaixo exemplifica bem como você pode planejar e executar seus testes:

- Defina uma lista de tarefas a implementar (o que testar);
- Escreva uma classe (test case) e implemente um método de teste para uma tarefa da lista;
- Rode o JUnit e certifique-se que o teste falha;
- Implemente o código mais simples que rode o teste;
- Refatore o código para remover a duplicação de dados;
- Caso necessário, escreva mais um teste ou refine o existente;
- Faça esses passos para toda a lista de tarefas;

Arquitetura das classes no JUnit

Para uma melhor compreensão de como o JUnit funciona é importante que entenda como suas classes estão organizadas dentro da API do framework (ver **figura 5**).



Como implementar o JUnit

Veja um exemplo de como você pode codificar em Java, classes de testes:

Crie uma classe que estenda `junit.framework.TestCase` para cada classe a ser testada:

```
1 import junit.framework.*;
2 class SuaClasseTest extends TestCase {...
3 }
```

Para cada método a ser testado defina um método public void test???() no test case SuaClasse:

```
1 public int Soma(Object o...) {...
2 }
3 SuaClasseTest: public void testSoma()
```

Analisando o resultado no JUnit

Quando os testes forem executados em modo gráfico, Veja na figura 6, que os métodos testados podem apresentar os seguintes resultados: verde para sucesso, roxo para falha e vermelho para exceção.

Figura 6. Resultados possíveis da execução dos testes.

Criando a classe de teste no Eclipse

Para **exemplificar o uso do JUnit**, usaremos o IDE Eclipse 3.0, que atualmente é um dos melhores ambientes para desenvolvimento de aplicações Java e por ser um projeto totalmente free que já incorpora em sua instalação, várias funcionalidades que aplicam práticas XP como refatoração, testes unitários e padrões. consulte a seção links desse artigo, o endereço oficial do projeto Eclipse para você baixá-lo.

Portanto, nesse caso, você não precisará **fazer o download do JUnit** e instalá-lo separadamente, pois o mesmo já está embutido no Eclipse.

Veja na **figura 7** a estrutura do test case CaculoTest.java, que é uma classe de testes implementada em Java através do Eclipse.

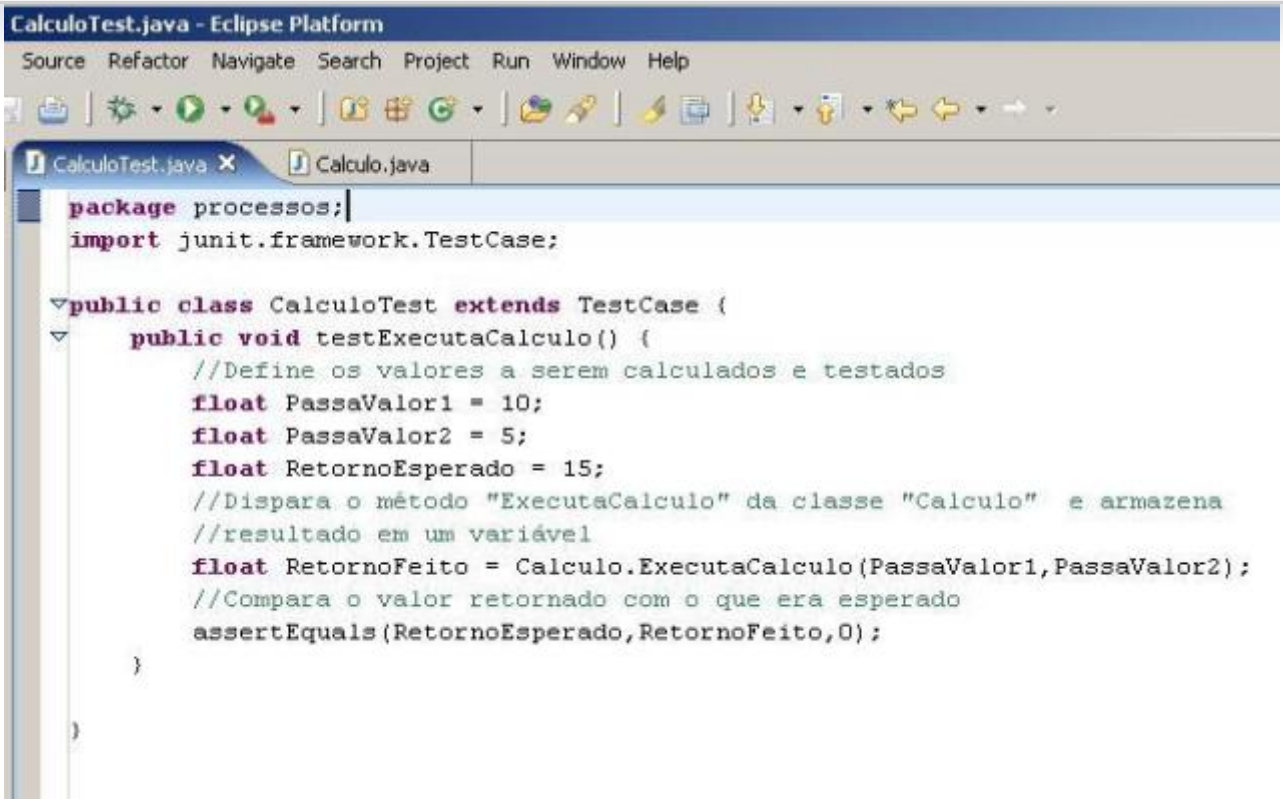


Figura 7. Estrutura de uma classe de teste (test case).

Testando a classe no Eclipse

Na figura anterior, você viu a sua classe de teste, e agora na figura 8, você verá a classe a ser testada para aquele test case.

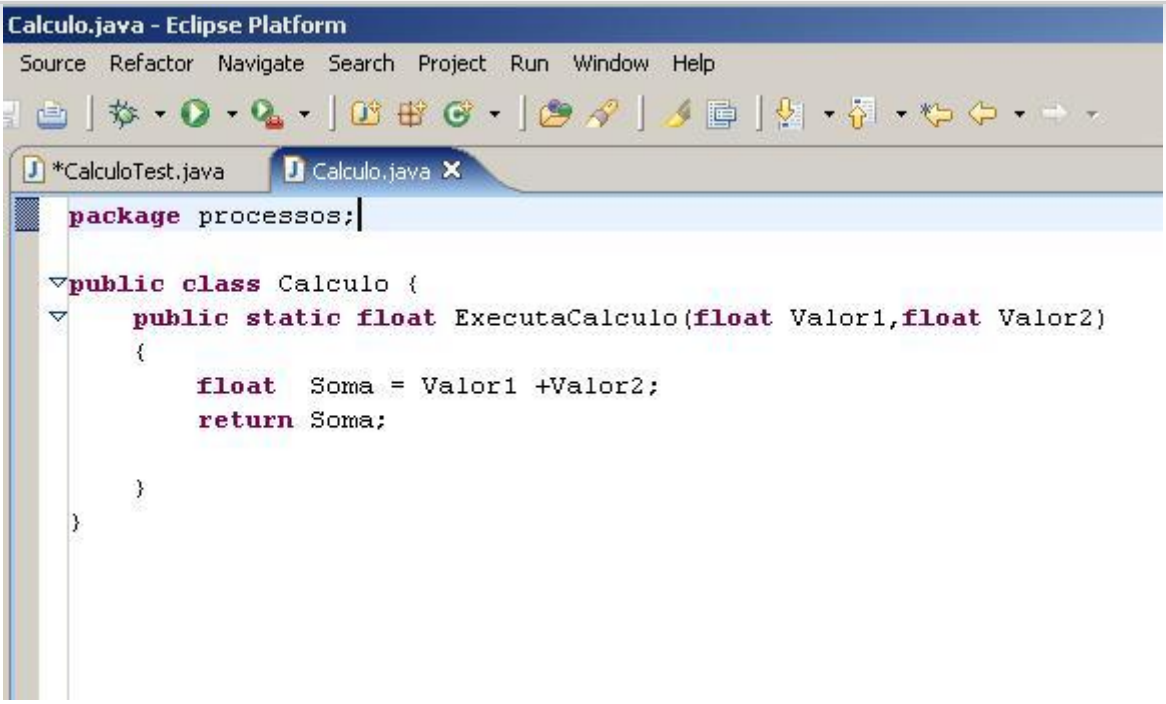


Figura 8. Classe Calculo.java que será testada.

Rodando o teste em modo gráfico no Eclipse

Conforme mostra na figura 9 para rodar o teste em modo gráfico pelo Eclipse, abra sua classe de teste, clique no menu Run\Run As e escolha a opção JUnit Test.

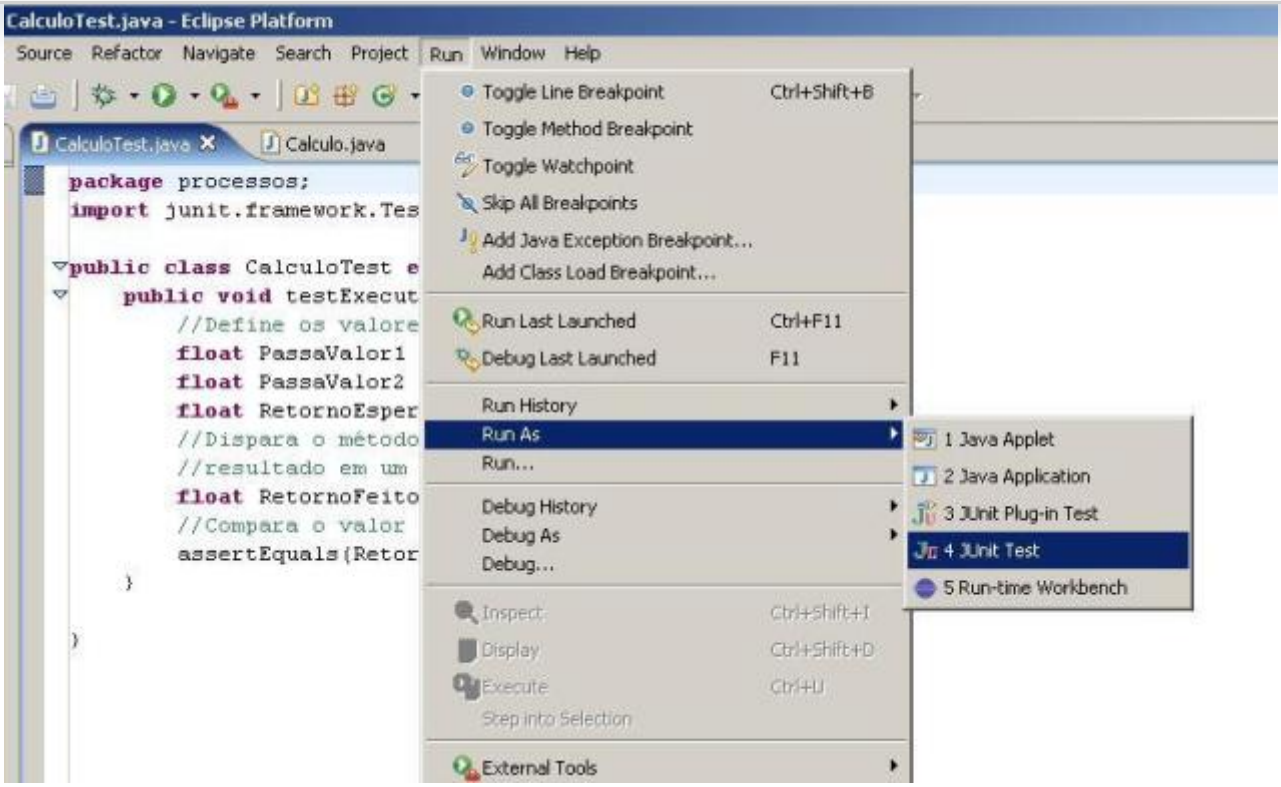


Figura 9. Iniciando o teste em modo gráfico.

Resultado em caso de sucesso

Você pode notar na figura 10, que o teste case CalculoTest.java está usando a declaração assertiva do método `assertEquals()`, que é assinado com dois parâmetros principais: o valor esperado, e valor retornado.

Observe que para isso instanciamos os atributos(variáveis) PassaValor1 = 10, PassaValor2 = 5 e RetornoEsperado = 15.

Sendo que para o atributo RetornoFeito atribuímos o resultado do método `xecutaCalculo()` da classe Calculo, passando como parâmetros PassaValor1 e PassaValor2.

Veja que na lateral esquerda da tela temos o resultado da **execução do teste através do JUnit**, que nesse exemplo retornou com sucesso(verde), pois os valores passados(10 e 5) certamente quando somados retornam 15.

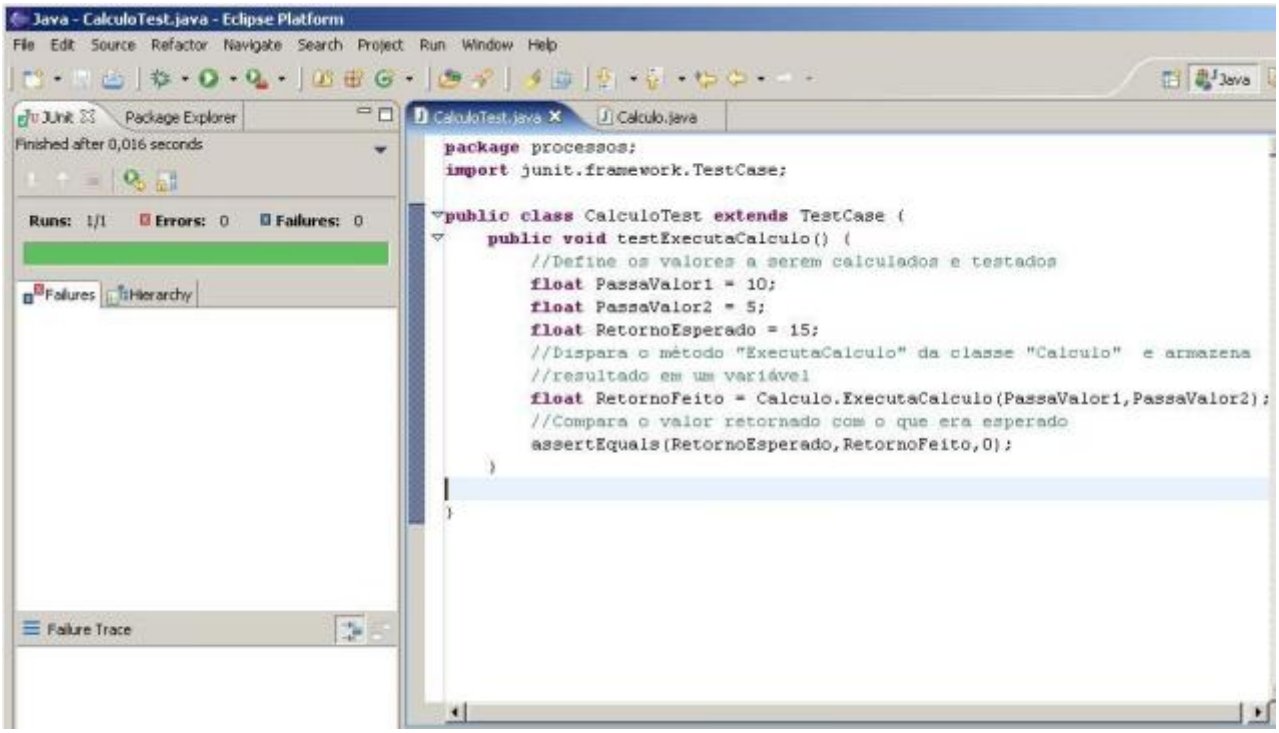


Figura 10. Resultado com SUCESSO após a execução do teste.

Resultado em caso de FALHA no Eclipse

Agora na figura 11, seguindo o mesmo raciocínio, vamos simular uma execução com falha, para isso, mudaremos o atributo PassaValor2 para 6, dessa forma, claramente que 10 somado a 6 não dá 15 (valor esperado), dessa forma, o JUnit retornará uma falha (vermelha), observe também, que na parte inferior da lateral esquerda da tela, ele mostra qual método falhou, e qual foi à falha.

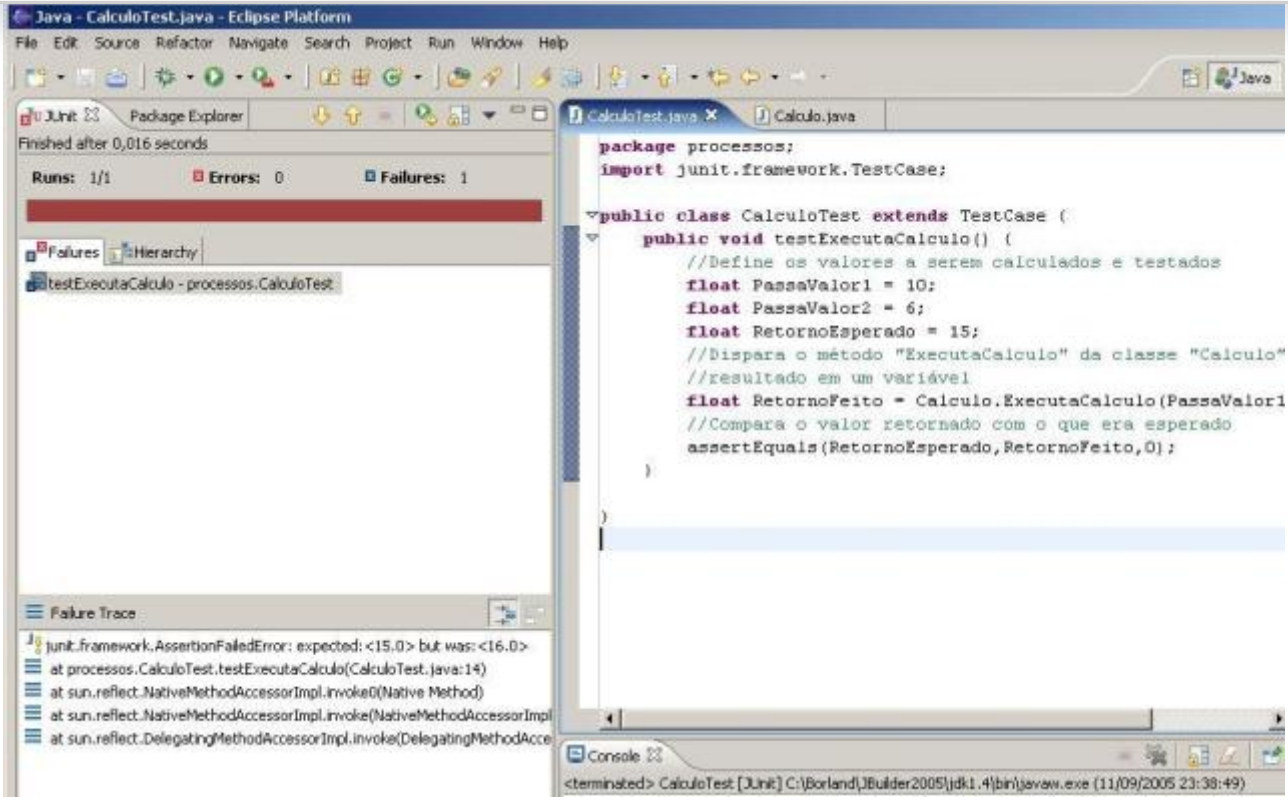


Figura 11. Resultado com FALHA após a execução do teste.

Confira neste artigo sobre Java Test onde iremos nos aprofundar mais no assunto.

Links Úteis

- **Bate-papo sobre mobile**:

  Neste DevCast temos um bate-papo sobre o desenvolvimento de apps mobile. Falamos aqui sobre quais tecnologias estudar, quando optar por cada uma e o que o mercado espera do desenvolvedor atualmente.

- **CRUD em PHP e MVC com Busca e Paginação**:

  Aprenda a implementar uma busca, paginação e conversão monetária em PHP e MVC. Para isso vamos utilizar o projeto que criamos no curso “Como implementar um CRUD em PHP com MVC”.

- **Conectando no SQL Server utilizando PDO em PHP**:

  Neste conteúdo você aprenderá a conectar no Banco de Dados SQL Server utilizando PDO em PHP. PDO é uma classe desenvolvida especificamente para trabalhar com procedimentos relacionados a BDs.