**{Introdução**

**Nosso trabalho é sobre testes unitários, mas eu vou começar com uma breve introdução sobre os outros tipos de testes dessa trindade aqui, representados nesta pirâmide: Pra que vcs tenham um entendimento mais amplo sobre o que estamos falando, mas sem me aprofundar muito nos outros tipos de testes.**

>”e2e - **End to End**” ou teste de ponta a ponta (mais caro e complexo);

Testa todo fluxo do sistema, de uma ponta À outras, desde o front end até o último nível de back end.

>”**integration**” ou teste de integração (preço mediano, amplitude maior) ;

No meio temos o teste de integração, que basicamente testa várias unidades em conjunto.

>”**unit test**” ou teste unitário (mais barato).

é sobre ele que vamos falar nessa apresentação, e

Vocês podem notar que A pirâmide mostra que a base “unit test” detém maior porção, por ser mais barato, sendo assim mais usado que os demais. sendo o teste **End to End** o mais caro e complexo de realizar.

**O que são testes unitários?**

O teste unitário é uma verificação feita com uma pequena porção de código, uma unidade isolada de um programa, geralmente um algoritmo, como um método ou uma função. a fim de mostrar que as unidades funcionam individualmente. Diferente do teste end to end né, que verifica todo fluxo do sistema.

**PRINT**

aqui tem como exemplo uma classe **pagamento**, que possui o método **cobrar**. Notem que **cobrar** recebe como parâmetro **cliente**.

Sendo assim, poderíamos escrever um teste para esse método aqui, verificando se o saldo do cliente é menor após a chamada dele. **PRINT** Como resultado teríamos uma classe de teste, como demonstrado nesse exemplo:

**PRINT**

**}**

**{Vantagens de realizar testes unitários**

**-**Precisão, ele aponta o caso específico que está gerando problemas;

**-**Prevenção de BUGS;

-Código testado se torna mais confiável;

-Podemos alterar o código sem medo, já que ele foi testado;

-o código só é escrito uma vez e é usado sempre que for necessário rastrear bugs, independentemente de alterações que ocorrerem.

-Usar testes de unidade é TAMBÉM É vantajoso por ser uma tarefa automatizada. Não há necessidade de pessoas voltadas somente para essa tarefa.

-Representando assim uma redução de custos com esforço manual.

**Quando fazer um teste unitário**

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

**}**

**{Quem faz o teste unitário?**

Test Case ( testa cada classe separadamente): **Desenvolvedor** (Projeta, escreve e roda).

Test Suite(Roda vários test cases): **Coordenador junto do Desenvolvedor**

Teste de aceitação, que nada mais é que a **homologação**, que é feita junto ao **cliente**.

**PRINT**

**Teste unitário: O que testar?**

**Sempre existem dúvidas sobre o que deve ser testado nas classes, existem alguns macetes que podem nos ajudar a descobrir quais e quantos testes deverão ser escritos:**

**-LER DICAS**

**}**

**{Ferramentas para testes unitários**

também conhecidas como “frameworks”

Dentre as mais utilizadas, temos:

-NUnit e xUnit para .Net

-PHPUnit para PHP

-JUnit para Java e nós vamos ver agora um pouco sobre ele, porque afinal é a linguagem que estamos usando

**JUnit para Java**

**O que é JUnit**

**JUnit é um framework que facilita o desenvolvimento e execução de testes unitários em código Java.**

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

#### **Por que usar JUnit?**

* **Facilita a criação, execução automática de testes e a apresentação dos resultados;**
* **É Orientado a Objeto;**
* **É gratuito e pode ser baixado no site do JUnit.**

**Vejam um exemplo de como podemos codificar em Java, uma classe de testes:**

**Aqui foi criada uma classe que extende junit.framework.TestCase para cada classe a ser testada:**

**PRINT**

**Para cada método a ser testado vamos definir um método public void test() no test case SuaClasse:**

**PRINT**

**Analisando o resultado no JUnit**

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

**e pra finalizar, vamos ver algumas imagens do teste rodando no Eclipse**

**PRINT 1 > rodando o teste no eclipse**

**PRINT 2 > o teste apresenta sucesso**

**PRINT 3> o teste apresenta falha**