Testes

- Garante que o software faz corretamente aquilo que ele se propõe a fazer.
- Validação: se o software faz o que deve ser feito;
- Verificação: se o software faz corretamente o que deve ser feito;
- A qualidade de um software depende da qualidade do seu teste
- Previne contra o aparecimento de bugs oriundos de códigos mal escritos;
- Código testado é mais confiável;
- Permite alterações sem medo - refatorações;
- Testa situações de sucesso e falha;
- Gera e preserva um “conhecimento” sobre as regras de negócios do projeto.

Existem dois tipos de técnicas de testes:

- **Testes Caixa Preta:** entrada e saída;
- **Testes Caixa Branca:** Lógica interna.

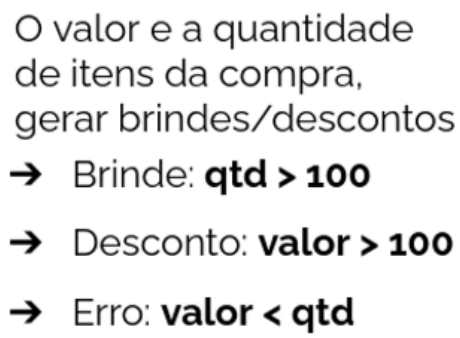
Níveis de Testes

- **Testes de Unidade:** Testaremos a menor unidade do sistema que estamos usando, no caso de POO testaremos classe por classe.
- **Testes de Integração:** Testaremos mais de uma classe por vez, a interação entre elas.
- **Testes de Sistema:** Consideramos a funcionalidade do sistema em geral. Teste de alto nível.
- **Testes de Aceitação:** O sistema faz o que o cliente quer?

Técnicas de Testes

Comece sempre pelos mais simples;

- **Cobertura de comandos -** testar todos os comandos dos testes;
- **Cobertura de caminhos -** testar todos os caminhos possíveis (os caminhos afetam as variáveis);
- **Particionamento em classes de equivalência;**
  - Um método calcula o desconto a partir do número de itens: 25% para 3 ou mais, 50% para 9 ou mais. Temos três classes, considere uma entrada para cada classe.
- **Análise do valor limite -** Um teste para os valores limite de cada classe.
- **Tabela de decisão:**



4 testes possíveis...

Condição\Teste	1	2	3	4
valor > 100	Sim	Não	Sim	Não
qtde > 100	Sim	Sim	Não	Não
Ações				
Brinde	X			
Desconto	X		X	
Erro		X		

- **Testes de Robustez:**
  - Caracteres em números;
  - Propriedade null;
  - Propriedade vazia;
  - Entrada diferente da esperada.

Implementando Testes

Em Java, usamos o framework JUnit 5.

- Uma biblioteca é um conjunto de bytecodes compilado e distribuído em um arquivo.
- Como definir um teste:

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test; //importação da anotação T

class CachorroTest {

    @Test //Estou dizendo ao JUnit que esse método representa
    public void testaLatidoDoCachorro(){
        Cachorro meu Cachorro = new Cachorro("Luna", 3, 20.6)
        assertEquals("Au au", meuCachorro.latir());
    }

    @Test
    public void testaGetPeso(){
        Cachorro meuCachorro = new Cachorro("Luna", 3, 20.6);
        assertEquals(20.6, meuCachorro.getPeso(), 0.1); //Qua
}
```

O método de assert permite comparar o valor esperado com o valor retornado pelo método esperado,

- assertEquals - assertNotEquals
- assertTrue - assertFalse
- assertEquals com doubles

@BeforeEach

- Uma anotação que permite inicializar um teste que será repetido em um método à parte.

```
@BeforeEach
public void initMeuCachorro(){
    meuCachorro = new Cachorro("Luna", 3, 20.6);
}
```