Atividade 1 de Teste de Software - T01 2024.1

Aluno: JOAO MARCOS P. CAVALCANTE

Link para Repositório Github

Escolha do Problema

- Foi feita uma pesquisa no StackOverflow com a seguinte string: "[unit-testing] or [junit] or [pytest]". As perguntas então foram ordenadas decrescentemente baseada na pontuação, a pergunta escolhida deveria ter uma resposta correta e no mínimo 400 votos. Procurei preferecialmente uma pergunta envolvendo JUnit.
- Importante resaltar que foi utilizado o JUnit 5.8.1 para realizar esse projeto, sendo que em qualquer versão do JUnit5 o projeto também funcionará.

Descrição do Problema

- O problema é "How to test that no exception was thrown?", ou "Como testar que não foi lançada nenhuma exceção?". O usuário fornece um bloco de código demonstrando como ele "captura" exceções que talvez sejam lançadas durante o teste e pergunta se não há uma forma mais eficiente de fazer essa verificação, até sugerindo um recurso do JUnit chamado **@Rule**.
    - ```
      @Test
      public void foo() {
         try {
            // execute code that you expect not to throw Exceptions.
         } catch(Exception e) {
            fail("Should not have thrown any exception");
         }
      }
      ```
- Todas as demonstrações de código (da pergunta, da resposta correta e das demais respostas) estão no arquivo InicialTest, que está localizado em *./codigo/Pergunta/scr/test/java/InicialTest.java*
- Inicialmente foi implementada a forma como o usuário verifica se as exceções foram lançadas durante o teste:
      ```

```java
@Test
@DisplayName("Não Deve Depositar Com Conta Inativa")
public void naoDeveDepositarComContaInativa() {
    try {
        conta.desativar();
        conta.depositar(depositoInicial);
    } catch(Exception e) {
        fail("Should not have thrown any exception");
    }

}
```

```
iva ×

Tests failed: 1 of 1 test – 35 ms

/home/jm/.jdks/corretto-17.0.12/bin/java ...

org.opentest4j.AssertionFailedError: Should not have thrown any exception
<2 internal lines>
    at InicialTest.naoDeveDepositarComContaInativa(InicialTest.java:25) <1 internal line>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

- Como o depósito foi feito mesmo a conta estando desativada uma exceção foi lançada e "capturada", resultando em falha no teste.

Código que está sendo testado

- Para realizar os testes esse programa que simula funções básicas de uma conta bancária foi usado:

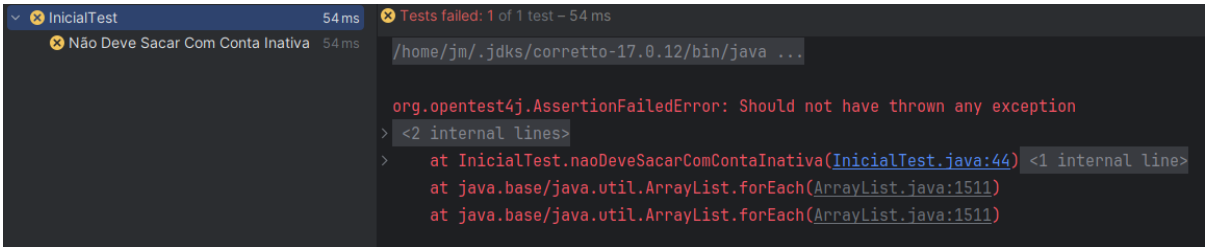
```java
public class Inicial {  2 usages
    private double saldo;  5 usages
    private boolean ativo;  4 usages

    public double getSaldo() {  1 usage
        return saldo;
    }

    public void ativar() {  1 usage
        this.ativo=true;
    }

    public void desativar() {  2 usages
        this.ativo=false;
    }

    public void depositar(double valor) {  4 usages
        if(this.ativo) {
            this.saldo = saldo + valor;
        } else {
            throw new RuntimeException("Conta inativa, deposito negado!");
        }
    }

    public void sacar(double valor) {  3 usages
        if(this.ativo && (this.saldo >= valor)) {
            saldo -= valor;
        } else {
            throw new RuntimeException("Conta inativa ou saldo insuficiente, saque negado!");
        }
    }
}
```

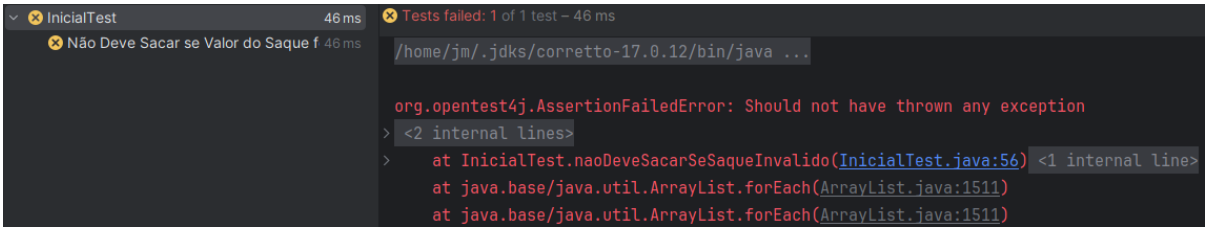
Resposta Correta

- Antes de tratar da pergunta em si o autor da resposta escreve um material detalhando conceitos testes unitários.
- O autor da resposta correta identifica que o autor da pergunta poderia dividir o teste em dois: um para identificar uma entrada válida e outro para identificar uma entrada errada. Sugerindo que o autor da pergunta divida o teste em dois para conseguir identificar possíveis problemas no código de forma mais precisa.

```java
// Métodos utilizados na Resposta Correta
@Test
@DisplayName("Não Deve Sacar Com Conta Inativa")
public void naoDeveSacarComContaInativa() {
    try {

        conta.depositar(depositoInicial);
        conta.desativar();
        conta.sacar(saqueValido);

    } catch(Exception e) {
        fail("Should not have thrown any exception");
    }
}

@Test
@DisplayName("Não Deve Sacar se Valor do Saque for Inválido")
public void naoDeveSacarSeSaqueInvalido() {
    try {
        conta.depositar(depositoInicial);
        conta.sacar(saqueInvalido);

    } catch(Exception e) {
        fail("Should not have thrown any exception");
    }
}
```

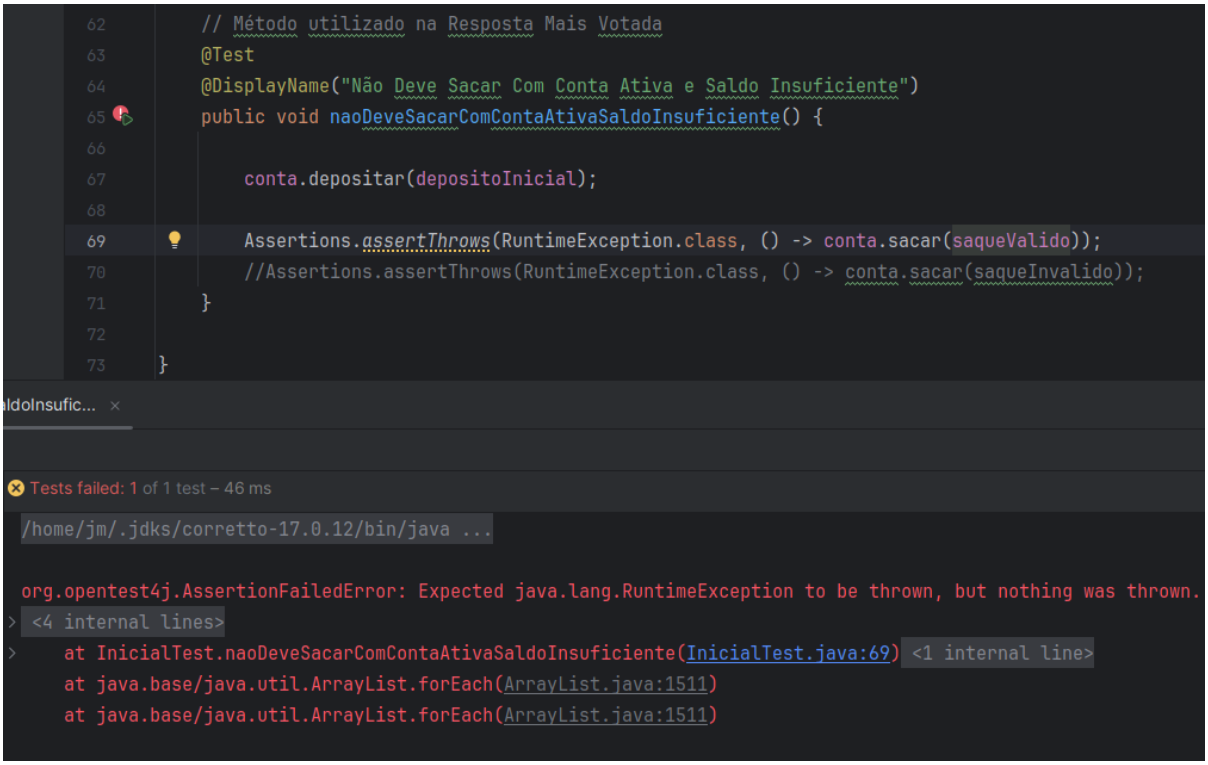
- A seguir as exceções lançadas por cada método:



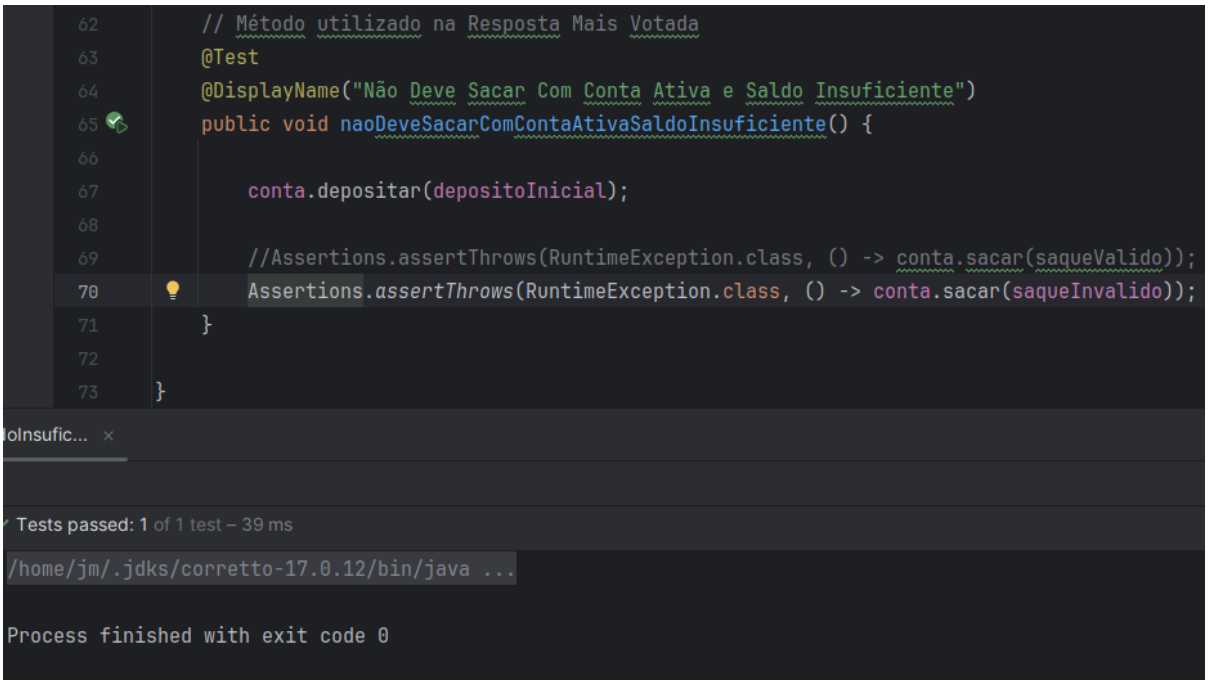


- No entanto, nessa resposta nenhuma solução referente ao tratamento de exceções utilizando recuros do *JUnit* é disponibilizada.

Análise da Resposta Mais Votada

- O autor da resposta mais votada é mais preciso com sua resposta, ele apresenta métodos específicos do *JUnit* para se lidar com exceções:
  - `assertAll()`
  - `assertDoesNotThrow()`
  - `assertThrows()`
- Para este projeto é implementado o `assertThrows()`, que verifica se o executável que foi fornecido lança uma exceção, verificando também se o tipo de exceção corresponde a um tipo específico que é determinado no método.
- Nessa primeira situação como o valor do saque é válido, o programa executa corretamente e não lança uma exceção. Assim o teste falha, já que a condição de sucesso é que uma exceção seja lançada:
  - 

- Na segunda situação o valor do saque é inválido e uma exceção é lançada, resultando em sucesso no teste:


// Método utilizado na Resposta Mais Votada
@Test
@DisplayName("Não Deve Sacar Com Conta Ativa e Saldo Insuficiente")
public void naoDeveSacarComContaAtivaSaldoInsuficiente() {
conta.depositar(depositoInicial);
//Assertions.assertThrows(RuntimeException.class, () -> conta.sacar(saqueValido));
Assertions.assertThrows(RuntimeException.class, () -> conta.sacar(saqueInvalido));
}
}
Tests passed: 1 of 1 test – 39 ms
/home/jm/.jdks/corretto-17.0.12/bin/java ...
Process finished with exit code 0