Plano de Testes

Simulador de Banco em Python

Alunos: Dayvid Araujo Ferreira da Silva - 1352221597
Danilo Ferreira de Souza - 1352220438
Lucas Baccas de Souza Nascimento - 1352221862

1. ## Cadastro de Conta

1.1 Criar nova conta com dados válidos:

- Verificar se é possível criar nova conta fornecendo informações válidas, como nome, número de identificação, e saldo inicial.
- Confirmar se a conta é criada com sucesso no sistema.

1.2 Cadastro com informações válidas:

- Testar o cadastro de uma conta com diferentes combinações de informações válidas (por exemplo, nomes diferentes, números de identificação únicos).
- Verificar se o sistema aceita e processa corretamente essas informações.

1.3 Garantir a unicidade de cada conta nova:

- Tentar cadastrar duas contas com as mesmas informações.
- Confirmar que o sistema impede a criação de contas duplicadas.

2. ## Acesso à Conta

2.1 Login com credenciais corretas:

- Efetuar login usando credenciais corretas.
- Verificar se o acesso é concedido e o usuário é redirecionado corretamente para sua conta.

2.2 Bloqueio de acesso com credenciais incorretas:

- Tentar efetuar login com combinações incorretas de nome de usuário e senha.
- Confirmar que o sistema bloqueia o acesso e exibe uma mensagem de erro apropriada.

3. ## Realizar Saque

3.1 Saque com saldo suficiente:

- Garantir que é possível realizar um saque quando o saldo é suficiente.
- Verificar se o valor do saldo é atualizado corretamente após o saque.

3.2 Saque com saldo insuficiente:

- Tentar efetuar um saque quando o saldo é insuficiente.
- Confirmar que o sistema impede o saque e exibe uma mensagem adequada.

3.3 Saque com valores inválidos:

- Testar o saque utilizando valores inválidos (negativos, caracteres não numéricos).
- Verificar se o sistema trata esses casos corretamente.

4. Realizar Depósito

4.1 Depósito com valor válido:

- Realizar um depósito com um valor válido.
- Confirmar que o valor é adicionado corretamente ao saldo da conta.

4.2 Depósito com valores negativos ou inválidos:

- Tentar realizar um depósito com valores negativos ou caracteres não numéricos.
- Verificar se o sistema rejeita essas entradas e exibe uma mensagem de erro apropriada.

4.3 Atualização do saldo após depósito:

- Verificar se o saldo da conta é atualizado corretamente após realizar um depósito.

Detalhamento Da Abordagem de Testes

1. Cadastro de conta

1.1 Criar uma nova conta com dados validos.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste Funcional
Objetivo do Teste	Verificar se o sistema é capaz de criar uma nova conta com dados válidos, garantindo que as informações fornecidas são aceitas e a conta é corretamente registrada.

1.2 Cadastro com as informações validas.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste de Aceitação
Objetivo do Teste	Testar o sistema com diferentes combinações de informações válidas para garantir que ele aceita e processa corretamente os dados fornecidos.

1.3 Garantir a unicidade de cada conta nova.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste de Validação
Objetivo do Teste	Confirmar que o sistema impede a criação de contas duplicadas, garantindo que cada conta tenha informações únicas.

2. Acesso à conta

2.1 Login com credenciais corretas.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste Funcional
Objetivo do Teste	Verificar se o sistema permite o acesso à conta quando as credenciais corretas são fornecidas.

2.2 Bloqueio de acesso com credenciais incorretas.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste de estresse
Objetivo do Teste	Garantir que o sistema bloqueia o acesso quando credenciais incorretas são fornecidas, simulando uma situação de ataque ou tentativas repetidas.

3. Realizar saque

3.1 Saque com saldo suficiente.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste Funcional
Objetivo do Teste	Confirmar que o sistema permite saques quando o saldo é suficiente e que o saldo é atualizado corretamente.

3.2 Saque com saldo insuficiente.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste de Exceção
Objetivo do Teste	Verificar se o sistema impede saques quando o saldo é insuficiente e exibe uma mensagem de erro apropriada.

3.3 Saque com valores inválidos.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste de Validação
Objetivo do Teste	Testar o sistema com valores inválidos para garantir que ele trata esses casos corretamente.

4. Realizar deposito

4.1 Deposito com valor valido.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste Funcional
Objetivo do Teste	Confirmar que o sistema permite depósitos com valores válidos e que o saldo é atualizado corretamente.

4.2 Deposito com valores negativos ou inválidos.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste de Validação
Objetivo do Teste	Verificar se o sistema rejeita depósitos com valores negativos ou inválidos, exibindo mensagens de erro apropriadas.

4.3 Atualização do saldo após deposito.

Tipo do Teste	Teste de Unidade
Subtipo do Teste	Teste Funcional
Objetivo do Teste	Confirmar que o saldo da conta é atualizado corretamente após a realização de um depósito.

Classe de Testes

1. Testes

```
import unittest
from simulador import ContaBancaria,
ContaExistenteError, CredenciaisIncorretasError

class TestContaBancaria(unittest.TestCase):
    def test_depositar(self):
        conta = ContaBancaria("Mateus", 1000.0)
        conta.depositar(500.0)
        self.assertEqual(conta.saldo, 1500.0)

    def test_sacar_saldo_suficiente(self):
        conta = ContaBancaria("Mateus", 1000.0)
        conta.sacar(500.0)
        self.assertEqual(conta.saldo, 500.0)

    def test_sacar_saldo_insuficiente(self):
        conta = ContaBancaria("Mateus", 1000.0)
        conta.sacar(1500.0)
        self.assertEqual(conta.saldo, 1000.0)

    def test_saldo_negativo_apos_saque(self):
        conta = ContaBancaria("Mateus", 1000.0)
        conta.sacar(1200.0)
        self.assertEqual(conta.saldo, 1000.0)

    def test_criar_nova_conta_com_dados_validos(self):
        nova_conta =
ContaBancaria.criar_nova_conta("Luana", 1500.0)
```

```
        self.assertEqual(nova_conta.titular, "Luana")
        self.assertEqual(nova_conta.saldo, 1500.0)

    def test_unicidade_de_cada_conta_nova(self):
        with self.assertRaises(ContaExistenteError):
            conta1 = ContaBancaria.criar_nova_conta("Alice",
2000.0)
            conta2 = ContaBancaria.criar_nova_conta("Lucas",
3000.0)

    def test_login_com_credenciais_corretas(self):
        conta = ContaBancaria("Mateus", 1000.0)
        self.assertTrue(conta.realizar_login("Mateus",
"senha_correta"))

    def
test_bloqueio_de_acesso_com_credenciais_incorretas(s
elf):
        conta = ContaBancaria("Mateus", 1000.0)
        self.assertFalse(conta.realizar_login("Mateus",
"senha_incorreta"))

if _name_ == '_main_':
    unittest.main()
```