Testes de Unidade e Integração
Aplicação Prática

1 Objetivo

Desenvolver e executar testes de unidade e integração para um sistema simples de gerenciamento de contas bancárias. O objetivo é garantir que o código funcione corretamente sob diferentes condições e identificar possíveis falhas. Considere aplicar todos os tipos de teste que foram descritos na teoria.

2 Descrição do Sistema

O sistema gerencia contas bancárias e possui as seguintes funcionalidades:

- Criar uma conta bancária com um saldo inicial.
- Depositar um valor na conta.
- Sacar um valor da conta, desde que haja saldo suficiente.
- Transferir um valor para outra conta bancária.
- Verificar o saldo da conta.

O código base do sistema é fornecido abaixo:

```python
import threading

class ContaBancaria:
  def __init__(self, titular: str, saldo_inicial: float):
    if not isinstance(titular, str):
      raise TypeError("O titular deve ser uma string")
    if not isinstance(saldo_inicial, (int, float)):
      raise TypeError("O saldo inicial deve ser um numero")
    if saldo_inicial < 0:
      raise ValueError("Saldo inicial nao pode ser negativo")

    self._titular = titular
    self._saldo = saldo_inicial
    self._historico = []
    self._lock = threading.Lock()  # Para garantir seguranca em acesso
       concorrente

  @property
  def titular(self):
    return self._titular

  @property
```

```
22    def saldo(self):
23      return self._saldo
24
25    @property
26    def historico(self):
27      return self._historico.copy()
28
29    def depositar(self, valor: float):
30      if not isinstance(valor, (int, float)):
31        raise TypeError("O valor do deposito deve ser um numero")
32      if valor <= 0:
33        raise ValueError("O valor do deposito deve ser positivo")
34      with self._lock:
35        self._saldo += valor
36        self._historico.append(f"Deposito de R$ {valor:.2f}")
37
38    def sacar(self, valor: float):
39      if not isinstance(valor, (int, float)):
40        raise TypeError("O valor do saque deve ser um numero")
41      if valor <= 0:
42        raise ValueError("O valor do saque deve ser positivo")
43      with self._lock:
44        if valor > self._saldo:
45          raise ValueError("Saldo insuficiente")
46        self._saldo -= valor
47        self._historico.append(f"Saque de R$ {valor:.2f}")
48
49    def transferir(self, destino, valor: float):
50      if not isinstance(destino, ContaBancaria):
51        raise TypeError("O destinatario deve ser uma instancia de
            ContaBancaria")
52      if not isinstance(valor, (int, float)):
53        raise TypeError("O valor da transferencia deve ser um numero")
54      if valor <= 0:
55        raise ValueError("O valor da transferencia deve ser positivo")
56      with self._lock:
57        self.sacar(valor)
58      with destino._lock:
59        destino.depositar(valor)
60        self._historico.append(f"Transferencia de R$ {valor:.2f} para {
            destino.titular}")
61        destino._historico.append(f"Transferencia de R$ {valor:.2f} recebida
            de {self.titular}")
62
63    def verificar_saldo(self):
64      return self._saldo
65
66    def __str__(self):
67      return f"Conta de {self._titular} - Saldo: R$ {self._saldo:.2f}"
```

3 Parte 1: Testes de Unidade (50 minutos)

Crie e execute testes de unidade para validar o comportamento esperado de cada função da classe `ContaBancaria`.

3.1 Tarefas

1. Criar uma classe de testes utilizando `unittest`.

2. Selecione dez tipos de teste descritos na lista para implementar.
    - Entrada e Saída;
    - Tipagem;
    - Consistência;
    - Inicialização e Liberação de Memória;
    - Mocking e Stubbing;
    - Modificação de Dados;
    - Concorrência e Acesso Simultâneo;
    - Limite Inferior;
    - Limite Superior;
    - Valores Fora do Intervalo;
    - Precisão em Valores Limítrofes;
    - Fluxos de Controle;
    - Cobertura de Decisões;
    - Estruturas de Repetição;
    - Cobertura Total;
    - Entradas Inválidas;
    - Exceções e Falhas Internas;
    - Erros em Dependências Externas;
    - Mensagens de Erro;
3. Executar os testes e verificar se todos passam.

4 Parte 2: Testes de Integração (50 minutos)

Agora, implemente testes de integração para verificar a interação entre múltiplas instâncias da classe `ContaBancaria`.

4.1 Tarefas

1. Criar um teste que simule múltiplas operações consecutivas, incluindo:
    - Criar duas contas bancárias.
    - Realizar depósitos e saques alternadamente.
    - Transferir dinheiro entre as contas.
    - Verificar se os saldos estão corretos após as operações.
2. Testar valores limite, como transferências de saldo total de uma conta para outra.
3. Criar um teste de erro verificando o comportamento do sistema ao tentar transferir dinheiro para um objeto inválido.

5 Entrega

Os alunos devem entregar:

- O código dos testes em um arquivo `.py`.
- Um pequeno relatório (máximo de 5 linhas) explicando se os testes passaram e se algum erro foi encontrado.