# Teste de *Software*

Aula II

# Metas para testes

- Completamente automatizados.
- Auto-verificáveis.
- Repetitíveis.
- Simples.
- Expressivos.
- Com separação de conceitos.
- Robustos.

# Estruturas condicionais

- Evitar testes com estruturas condicionais.
- Testes podem executar de maneiras diferentes.
- Complexidade ciclomática.

# Estruturas condicionais

```
public int conta(int numero) {
  int resultado = 10;
  if (numero > 10) {
    resultado += numero;
  }
  if (numero < 30) {
    resultado *= 2;
  }
  return resultado;
}
```

# Manutenibilidade do código de teste

- O código de teste também precisa ser mantido, compreendido e ajustado.
- Reduzir a duplicação de código beneficia a manutenibilidade dos testes.
- Mudanças na implementação afetam os testes.

# Manutenibilidade do código de teste

```java
@Test
public void testarCaixaEconomica() {
  SistemaBancario sistemaBancario = new SistemaBancario();
  Banco caixaEconomica = sistemaBancario.criarBanco("Caixa Econômica", Moeda.BRL);
  assertEquals("Caixa Econômica", caixaEconomica.obterNome());
  assertEquals(Moeda.BRL, caixaEconomica.obterMoeda());
}

@Test
public void testarTrindade() {
  SistemaBancario sistemaBancario = new SistemaBancario();
  Banco caixaEconomica = sistemaBancario.criarBanco("Caixa Econômica", Moeda.BRL);
  Agencia trindade = caixaEconomica.criarAgencia("Trindade");
  assertEquals("001", trindade .obterIdentificador());
  assertEquals("Trindade", trindade.obterNome());
  assertEquals(caixaEconomica, trindade.obterBanco());
}
```

# Estratégias de *Fixture Setup*

- *Transient Fixture*.
- *Persistent Fixture*.
- *Fresh Fixture Setup*.
- *Shared Fixture Construction*.
- Reuso de código e de execução.

# *Fresh Fixture Setup*

- *Inline Setup*.
- *Implicit Setup*.
- *Delegate Setup*.

# Inline Setup

```
@Test
public void testarCaixaEconomica() {
  SistemaBancario sistemaBancario = new SistemaBancario();
  Banco caixaEconomica = sistemaBancario.criarBanco("Caixa Econômica", Moeda.BRL);
  assertEquals("Caixa Econômica", caixaEconomica.obterNome());
  assertEquals(Moeda.BRL, caixaEconomica.obterMoeda());
}

@Test
public void testarTrindade() {
  SistemaBancario sistemaBancario = new SistemaBancario();
  Banco caixaEconomica = sistemaBancario.criarBanco("Caixa Econômica", Moeda.BRL);
  Agencia trindade = caixaEconomica.criarAgencia("Trindade");
  assertEquals("001", trindade .obterIdentificador());
  assertEquals("Trindade", trindade.obterNome());
  assertEquals(caixaEconomica, trindade.obterBanco());
}
```

# *Inline Setup*

- Boa compreensão da relação de causa e efeito entre *fixtures* e saídas do SUT.
- Liberdade de organização das classes de teste.
- Causa duplicação de código de teste.

# Implicit Setup

```
public class TesteBancoAgencia {

  private Banco caixaEconomica;

  @Before
  public void configurar() {
    SistemaBancario sistemaBancario = new SistemaBancario();
    caixaEconomica = sistemaBancario.criarBanco("Caixa Econômica", Moeda.BRL);
  }

  @Test
  public void testarCaixaEconomica() {
    assertEquals("Caixa Econômica", caixaEconomica.obterNome());
    assertEquals(Moeda.BRL, caixaEconomica.obterMoeda());
  }

  @Test
  public void testarTrindade() {
    Agencia trindade = caixaEconomica.criarAgencia("Trindade");
    assertEquals("001", trindade .obterIdentificador());
    assertEquals("Trindade", trindade .obterNome());
    assertEquals(caixaEconomica, trindade .obterBanco());
  }

}
```

# *Implicit Setup*

- Promove o reuso de código de teste entre testes de uma mesma classe.
- Moderada compreensão da relação de causa e efeito entre *fixtures* e saídas.
- Limita a organização das classes de teste.

# Delegate Setup

```
public class TesteBanco {

  @Test
  public void testarCaixaEconomica() {
    Banco caixaEconomica = Auxiliar.criarCaixaEconomica();
    assertEquals("Caixa Econômica", caixaEconomica.obterNome());
    assertEquals(Moeda.BRL, caixaEconomica.obterMoeda());
  }

}

public class TesteAgencia {

  @Test
  public void testarTrindade() {
    Banco caixaEconomica = Auxiliar.criarCaixaEconomica();
    Agencia trindade = Auxiliar.criarTrindade(caixaEconomica);
    assertEquals("001", trindade .obterIdentificador());
    assertEquals("Trindade", trindade .obterNome());
    assertEquals(caixaEconomica, trindade .obterBanco());
  }

}
```

# Delegate Setup

```
public class Auxiliar {

  public static Banco criarCaixaEconomica() {
    SistemaBancario sistemaBancario = new SistemaBancario();
    return sistemaBancario.criarBanco("Caixa Econômica", Moeda.BRL);
  }

  public static Agencia criarTrindade(Banco banco) {
    return banco.criarAgencia("Trindade");
  }

}
```

# *Delegate Setup*

- Promove parcialmente o reuso de código de teste.
- Esconde detalhes não necessários para o teste.
- Liberdade de organização das classes de teste.
- Custo adicional para gerenciar classes e métodos auxiliares.
- Permite acesso à apenas um test fixture.
- Dificulta a compreensão da relação de causa e efeito entre *fixtures* e saídas.

# Exercício 4

- Baixar o projeto: github.com/lucasPereira/sistemaBancario/tree/experimento.
- Copiar as classes do pacote **`sistemaBancario`** para o seu projeto.
- Copiar os pacotes `sistemaBancario.experimento.etapa2.`**`inline`**, `sistemaBancario.experimento.etapa2.`**`delegate`** e `sistemaBancario.experimento.etapa2.`**`implicit`**.
- Completar os casos de teste incompletos.
- Observação: as classes dentro do pacote `sistemaBancario.experimento.`**`etapa1`** contêm exemplos das estratégias de *fixture setup*.

# Exercício 5

Dado que:

- Exista o sistema bancário.

Quando:

- For criado o banco Banco Do Brasil.

Então:

- O nome do banco será "Banco do Brasil".
- A moeda do banco será BRL.

# Exercício 5

Dado que:

- Exista o sistema bancário.
- Exista o banco Banco do Brasil.

Quando:

- For criada a agencia Centro.

Então:

- O identificador da agência será "001".
- O nome da agência será "Centro".
- O banco da agência será o Banco do Brasil.

# Exercício 5

Dado que:

- Exista o sistema bancário.
- Exista o banco Banco do Brasil.
- Exista a agencia Centro.

Quando:

- For criada a conta Maria.

Então:

- O identificador da conta será "0001-5".
- O titular da conta será "Maria".
- O saldo da conta será zero.
- A agência da conta será Centro.

# Exercício 5

Dado que:

- Exista o sistema bancário.
- Exista o banco Banco do Brasil.
- Exista a agencia Centro.
- Exista a conta Maria.

Quando:

- For realizada a operação de depósito de dez reais na conta Maria.

Então:

- A operação terá sido realizada com sucesso.
- O saldo da conta Maria será de dez reais.

# Exercício 5

Dado que:

- Exista o sistema bancário.
- Exista o banco Banco do Brasil.
- Exista a agencia Centro.
- Exista a conta Maria.
- A conta Maria tenha um saldo de dez reais.

Quando:

- For realizada a operação de saque de seis reais da conta Maria.

Então:

- A operação terá sido realizada com sucesso.
- O saldo da conta Maria será de quatro reais.

# Exercício 5

Dado que:

- Exista o sistema bancário.
- Exista o banco Banco do Brasil.
- Exista a agencia Centro.
- Exista a conta Maria.
- A conta Maria tenha um saldo de quatro reais.

Quando:

- For realizada a operação de saque de seis reais da conta Maria.

Então:

- A operação não terá sido realizada devido à saldo insuficiente.
- O saldo da conta Maria será de quatro reais.

# Exercício 6

- Adicione um terceiro parâmetro ao método **`criarBanco`** da classe **`SistemaBancario`**.
- Conserte os testes que haviam sido criados na etapa anterior.

```
public Banco criarBanco(String nome, Moeda moeda,Dinheiro taxa)
```