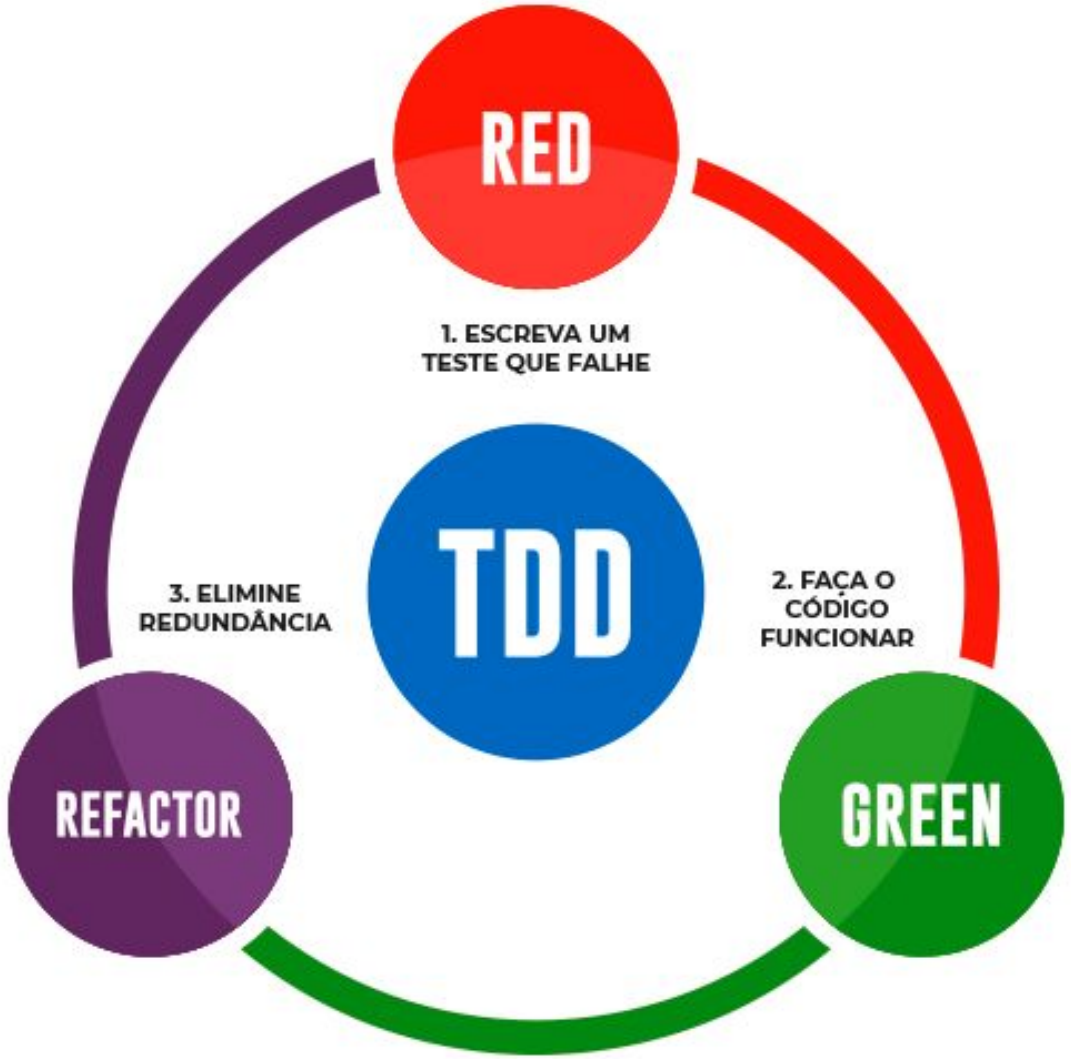
TDD - Test-Driven Development

TDD

- TDD significa Test-Driven Development, que é uma abordagem de desenvolvimento de software que coloca um forte foco na escrita de testes antes de escrever o código real.


RED
1. ESCREVA UM TESTE QUE FALHE
TDD
2. FAÇA O CÓDIGO FUNCIONAR
GREEN
3. ELIMINE REDUNDÂNCIA
REFACTOR

TEST

- O desenvolvedor escreve um teste que descreve a funcionalidade desejada do software.
- Este teste inicial geralmente falha, já que a funcionalidade ainda não foi implementada.

DEVELOP

- O desenvolvedor escreve o código necessário para fazer o teste passar. O objetivo é fazer com que o teste inicialmente falhado seja bem-sucedido. Nenhum código é escrito além do necessário para fazer o teste passar.

REFECTOR

- O desenvolvedor pode refatorar o código para torná-lo mais limpo, eficiente e legível, sem alterar o comportamento observável do software.
- Isso ajuda a manter a qualidade do código.

- Este ciclo é repetido continuamente à medida que novas funcionalidades são adicionadas ou alterações são feitas no código.

USANDO O TDD

- Implementar uma calculadora que realiza operações de soma, subtração, multiplicação e divisão.

```java
@Test
public void testAdd() {
    Calculator calculator = new Calculator();
    int result = calculator.add(5, 3);
    assertEquals(8, result);
}
```

```java
@Test
public void testSubtract() {
    Calculator calculator = new Calculator();
    int result = calculator.subtract(8, 3);
    assertEquals(5, result);
}
```

```java
@Test
public void testMultiply() {
    Calculator calculator = new Calculator();
    int result = calculator.multiply(4, 2);
    assertEquals(8, result);
}
```

```java
@Test
public void testDivide() {
    Calculator calculator = new Calculator();
    double result = calculator.divide(10, 2);
    assertEquals(5.0, result, 0.001);
}
```

```java
@Test
public void testDivideByZero() {
    Calculator calculator = new Calculator();
    assertThrows(IllegalArgumentException.class, () -> {
    calculator.divide(5, 0);
    });
}
```

```java
@Test
public void testDivideByZero() {
    Calculator calculator = new Calculator();
    try {
    calculator.divide(5, 0);
    fail("Expected IllegalArgumentException to be thrown");
    } catch (IllegalArgumentException e) {
    // Espera-se que uma exceção do tipo IllegalArgumentException seja lançada, portanto, este bloco catch será executado.
    }
}
```

NOTA

- O método fail("mensagem") é usado para indicar explicitamente que um teste deve falhar.
- Ele é normalmente usado em conjunto com um bloco try-catch para tratar de situações em que você espera que um teste falhe.
- A mensagem de erro especificada é exibida quando o teste falha, fornecendo informações adicionais sobre o motivo da falha.

USANDO TDD

- Criar uma classe simples para calcular média de um vetor.

```java
@Test
public void testCalculateAverage() {
    AverageCalculator calculator = new AverageCalculator();
    int[] numbers = { 5, 10, 15, 20, 25 };
    double result = calculator.calculateAverage(numbers);
    assertEquals(15.0, result, 0.01);
}
```

USANDO TDD

- Vamos criar uma classe StringManipulator que manipula strings, começando com a função de concatenação e adicionar a função de inverter uma string.

```java
@Test
public void testCalculateAverage() {
    AverageCalculator calculator = new AverageCalculator();
    int[] numbers = { 5, 10, 15, 20, 25 };
    double result = calculator.calculateAverage(numbers);
    assertEquals(15.0, result, 0.01);
}
```

```
@Test
public void testReverseString() {
StringManipulator manipulator = new StringManipulator();
String result = manipulator.reverseString("abcdef");
assertEquals("fedcba", result);
}
```