



Exercício TDD

Sessão Kata de TDD – Construindo um Tic-Tac-Toe (Jogo da Velha)



Introdução ao TDD

Test Driven Development

O que é o TDD?

- Desenvolvimento orientado a testes
- Mudança de paradigma e de forma de pensar: ao invés de pensar na lógica de programação primeiro pensamos em como testar primeiro.
- Sequência do TDD:
 1. Escreva não mais do que um teste de unidade suficiente para falhar (falhas de compilação são falhas). RED
 2. Escreva não mais do que o código de produção necessário para passar no teste de unidade com falha. GREEN
 3. Refatore o código, visando legibilidade e manutenibilidade. Atenção: testes também são código. REFACTOR

Vantagens do TDD

- Ter SEMPRE uma cobertura de testes de 100%
- Melhora da arquitetura por forçar um design mais desacoplado (ao pensar no teste primeiro, naturalmente nosso código terá de seguir boas práticas para ser testável)
- Escrever o mínimo de código possível para atender aos requisitos. Sem "zonas mortas" de código.
- Identificar problemas/ambiguidades na definição dos requisitos ANTES de codificar.
- Testes fornecem uma documentação pronta de como a aplicação deve funcionar de forma que qualquer um entende.
- Caso uma mudança futura quebre um teste, não subimos. Isso dá confiança para refatorarmos o código e sempre mantê-lo o mais limpo possível. Assim o código não "apodrece".

TDD na Prática

- Muitos times não atuam com o TDD porque apesar de ter regras simples é um paradigma muito diferente e por isso exige muita prática
- Por isso fazemos exercícios que já sabemos como resolver, chamados Katas (terminologia do Karatê).
- Todo profissional deve treinar antes de atuar de fato no dia a dia. Com desenvolvimento de software não é diferente, e os Katas são uma ótima maneira de "afiar" nossas habilidades de desenvolvimento.
- Hoje vamos treinar desenvolvendo um jogo da velha



Exercício - Tic Tac Toe (Jogo da Velha)

Refinamento dos requisitos

Definição dos Requisitos

1. Alocar peças no tabuleiro:
 - No TicTacToe, uma peça (piece) contém os símbolos X ou O
 - Uma peça pode ser colocada em qualquer espaço de uma tabuleiro (board) 3x3
2. Suporte a 2 jogadores:
 - Deve existir uma maneira de se saber qual jogador da próxima jogada
3. Condições de vitória:
 - Um jogador vence quando for o primeiro a conectar uma linha horizontal do tabuleiro ou vertical ou alguma das diagonais com todas as suas peças jogadas.
4. Condições de Empate:
 - Quando todos os locais já forem preenchidos, ocorreu um empate



Codificação

Sessão Kata de TDD do TicTacToe (Jogo da Velha)

- Faremos um treinamento com 28 movimentos no ciclo Red-Green-Refactor
- Temos que implementar 4 requisitos gerando 12 casos de teste
- O propósito é se habituar a prática do TDD e uma maior aproximação com as regras de negócio expressas em BDD
- Em “BDD in Action”, Smart recomenda a prática do TDD utilizando linguagem mais próxima do BDD. Implementar testes unitários dessa maneira é chamada por ele de “BDD de Baixo Nível”, unindo a expressividade do BDD à velocidade de entrega e execução dos testes unitários.
- Cada movimento é representado pelo par (Transição, código que implementa). Exemplo: (GREEN→RED, code-11). O respectivo está disponível como branches no repositório github.com/cristiancmello/kata-tdd-tictactoe



Criando Projeto Java + Maven

Java 11 com suporte ao Maven e JUnit 5

Configuração do Projeto

New Project

Empty Project

Generators

- Maven Archetype
- Java Enterprise
- Spring Initializr
- JavaFX
- Quarkus
- Micronaut
- Ktor
- Kotlin Multiplatform
- Compose Multiplatform
- HTML
- React
- Express
- Angular CLI
- IDE Plugin
- Android

To create a general Maven project, go to the [New Project](#) page.

Name: kata-tdd-tictactoe

Location: ~\Workspace
Project will be created in: ~\Workspace\kata-tdd-tictactoe

☐ Create Git repository

JDK: 11 version 11.0.15

Catalog: Internal [Manage catalogs...](#)

Archetype: org.apache.maven.archetypes:maven-archetype-quickstart [Add...](#)

Version: RELEASE

Additional Properties

No properties

> Advanced Settings

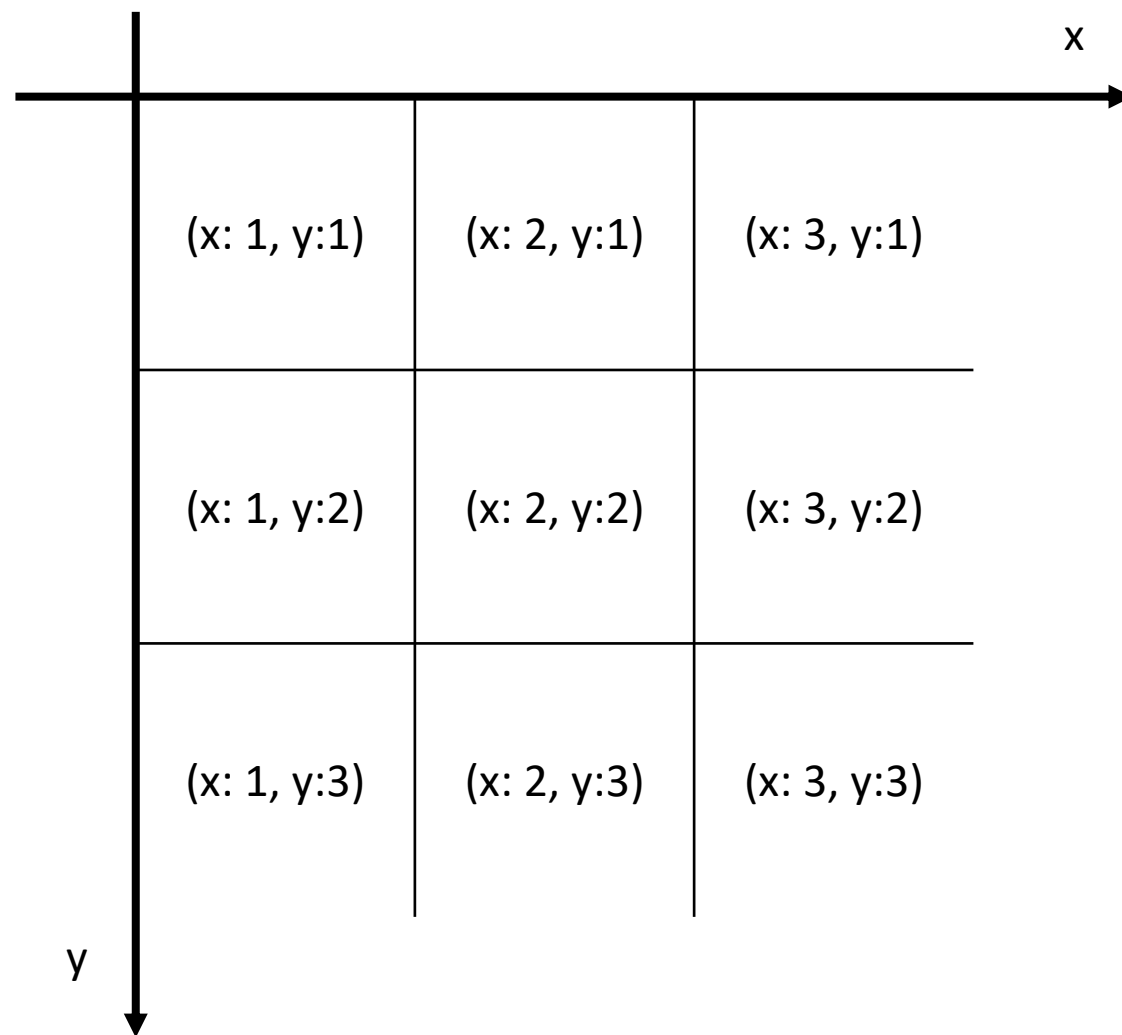
Create Cancel

- IntelliJ: Partindo de um projeto pré-configurado com Maven 3 e JUnit 4.
- O archetype maven quickstart cria projeto configurado com JUnit 4 que está sendo deprecado em favor do JUnit 5 que será adotado neste treinamento.
- Para simplificar o processo de adicionar dependências mais atualizadas do Maven, o repositório github.com/cristiancmello/kata-tdd-tictactoe tem um exemplo já configurado.

Requisito 1 – Alocando peças

- No TicTacToe, uma peça (piece) contém os símbolos X ou O
- Uma peça pode ser colocada em qualquer espaço de uma tabuleiro (board) 3x3
- Podemos quebrar este requisito em 3 testes
 - Teste 1 : **Quando** uma peça é colocada em qualquer lugar fora do eixo x, **então** lance um RuntimeException
 - Teste 2 : **Quando** uma peça é colocada em qualquer lugar fora do eixo y, **então** lance um RuntimeException
 - Teste 3 : **Quando** uma peça é colocada num espaço já ocupado por outra peça, **então** lance um RuntimeException

Tabuleiro 3x3 (proposta para visão do jogador)



(x: 1, y:1)	(x: 2, y:1)	(x: 3, y:1)
(x: 1, y:2)	(x: 2, y:2)	(x: 3, y:2)
(x: 1, y:3)	(x: 2, y:3)	(x: 3, y:3)

Codificação – Requisito 1 Teste 1 (RED, code-1)

- Os três testes verificam se uma Exception é lançada
- Como testar Exceptions?
 - Uma maneira popular é usando o `assertThrows` do JUnit5
- Criar Classe de teste `TicTacToeTest`
 - Criar método de teste `quandoXForaTabuleiroEntaoRuntimeException`
- Execute os testes e eles precisam falhar (RED)
- Obs.: Alguns autores preferem o uso de sintaxe dos cenários do BDD
 - Dado (Given), Quando (When), Entao (Then)

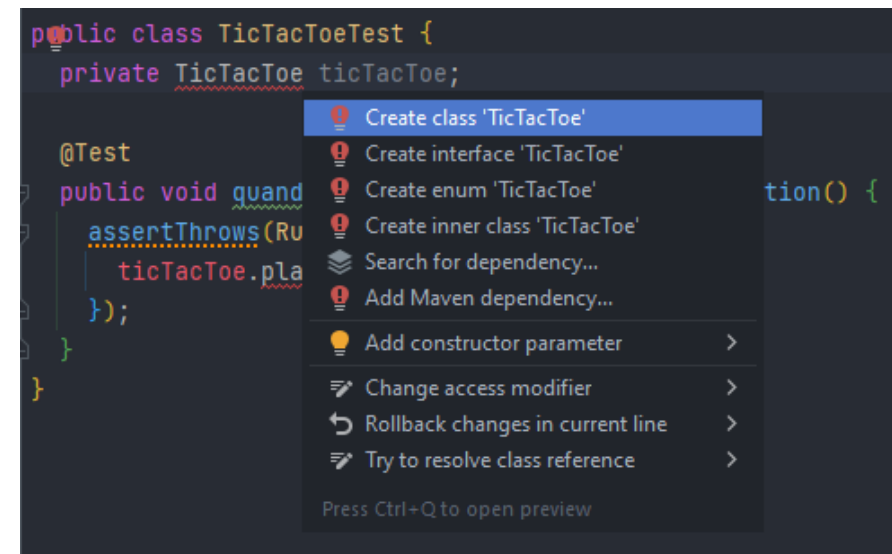
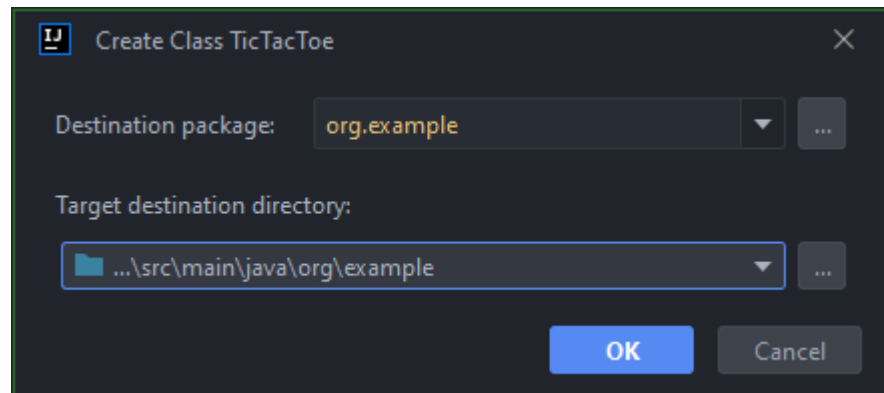
```
6 public class TicTacToeTest {
7     @Test
8     public void quandoXForaTabuleiroEntaoRuntimeException() {
9         assertThrows(RuntimeException.class, () -> {
10
11         });
12     }
13 }
```

```
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running org.example.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.017 s - in org.example.AppTest
[INFO] Running org.example.TicTacToeTest
[ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.001 s <<< FAILURE! - in org.example.TicTacToeTest
[ERROR] quandoXForaTabuleiroEntaoRuntimeException Time elapsed: 0 s <<< FAILURE!
org.opentest4j.AssertionFailedError: Expected java.lang.RuntimeException to be thrown, but nothing was thrown.
    at org.example.TicTacToeTest.quandoXForaTabuleiroEntaoRuntimeException(TicTacToeTest.java:9)

[INFO]
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   TicTacToeTest.quandoXForaTabuleiroEntaoRuntimeException:9 Expected java.lang.RuntimeException to be thrown, but nothing was thrown.
[INFO]
[ERROR] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0
```

Codificação – Requisito 1 Teste 1 (RED, code-2)

- Chame o método play com parâmetros x e y da peça que esteja fora do tabuleiro
 - Exemplo: 5, 2
- Registrar etapa Given para instanciar campo ticTacToe
 - @BeforeEach para ser executado antes de cada método de teste
- Teste de novo [RED]
- Dica no IntelliJ: Alt+Enter



Codificação – Requisito 1 Teste 1(RED, code-2)

```
package org.example;

public class TicTacToe {
}

public class TicTacToeTest {
    private TicTacToe ticTacToe;

    @BeforeEach
    public final void beforeEach() {
        // Given: criar tabuleiro equivale a instanciar classe TicTacToe onde ficará a representação do mesmo.
        ticTacToe = new TicTacToe();
    }

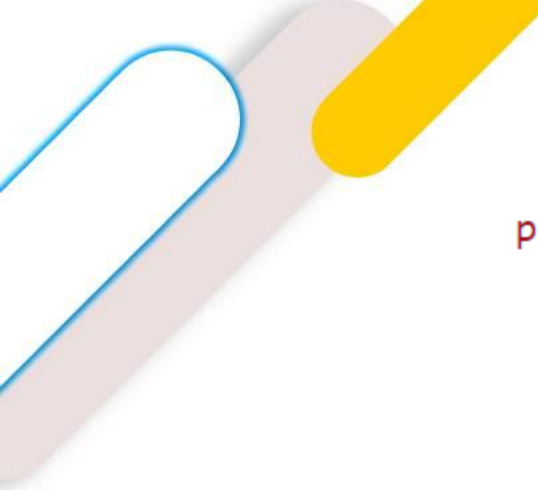
    @Test
    public void quandoXForaTabuleiroEntaoRuntimeException() {
        assertThrows(RuntimeException.class, () -> {
            ticTacToe.play(5, 2);
        });
    }
}
```



Codificação:

Requisito 1 Teste 1 (RED→GREEN, code-3)

- Método play precisa ser implementado e fazer o teste passar minimamente.
 - Foco em escrever um código mínimo, sem muitas preocupações técnicas
 - Mas é necessário bom senso para um código aceitável
 - É preciso treinar maior velocidade de escrita e as tecnologias a serem utilizadas para maior familiaridade
 - Obs.: no futuro o código será refatorado
- Teste novamente. Agora os testes falhos precisam passar.



```
public class TicTacToe {  
    public void play(int x, int y) {  
        if (x < 1 || x > 3) {  
            throw new RuntimeException("X esta fora do tabuleiro");  
        }  
    }  
}
```

```
[INFO] Results:  
[INFO]  
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0  
[INFO]  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.968 s  
[INFO] Finished at: 2022-06-18T18:44:49-03:00  
[INFO] -----
```

Codificação:

Requisito 1 Teste 2 (GREEN→RED, code-4)

- **Quando** uma peça é colocada em qualquer lugar fora do eixo y, **então** lance um RuntimeException
- Teste precisa quebrar

```
@Test
public void quandoYForaTabuleiroEntaoRuntimeException() {
    assertThrows(RuntimeException.class, () -> {
        ticTacToe.play(2, 5);
    });
}
```

```
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running org.example.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.016 s - in org.example.AppTest
[INFO] Running org.example.TicTacToeTest
[ERROR] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.001 s <<< FAILURE! - in org.example.TicTacToeTest
[ERROR] quandoYForaTabuleiroEntaoRuntimeException Time elapsed: 0 s <<< FAILURE!
org.opentest4j.AssertionFailedError: Expected java.lang.RuntimeException to be thrown, but nothing was thrown.
    at org.example.TicTacToeTest.quandoYForaTabuleiroEntaoRuntimeException(TicTacToeTest.java:26)

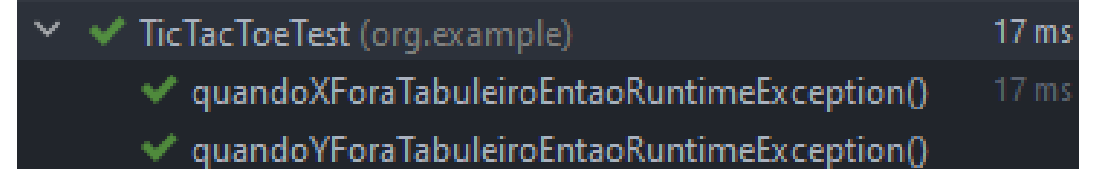
[INFO]
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   TicTacToeTest.quandoYForaTabuleiroEntaoRuntimeException:26 Expected java.lang.RuntimeException to be thrown, but nothing was thrown.
[INFO]
[ERROR] Tests run: 3, Failures: 1, Errors: 0, Skipped: 0
```

Codificação

Requisito 1 Teste 2 (RED→GREEN, code-5)

- Implemente o código que faça o teste 2 passar minimamente

```
public void play(int x, int y) {  
    if (x < 1 || x > 3) {  
        throw new RuntimeException("X esta fora do tabuleiro");  
    } else if (y < 1 || y > 3) {  
        throw new RuntimeException("Y esta fora do tabuleiro");  
    }  
}
```



A screenshot of a test runner interface with a dark background. It shows a list of test results. The first entry is 'TicTacToeTest (org.example)' with a green checkmark and '17 ms'. Below it are two more entries, both with green checkmarks and '17 ms': 'quandoXForaTabuleiroEntaoRuntimeException()' and 'quandoYForaTabuleiroEntaoRuntimeException()'.

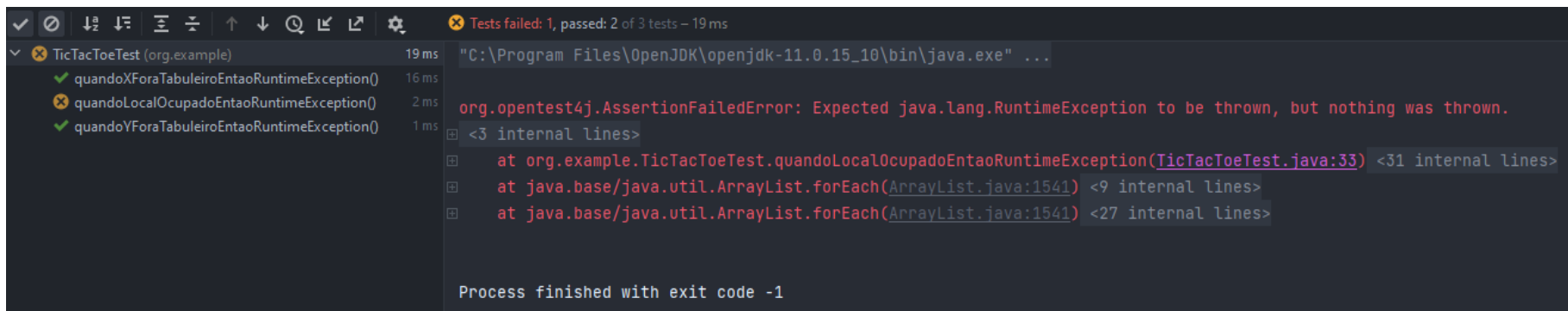
✓	TicTacToeTest (org.example)	17 ms
✓	quandoXForaTabuleiroEntaoRuntimeException()	17 ms
✓	quandoYForaTabuleiroEntaoRuntimeException()	

Codificação

Requisito 1 Teste 3 (GREEN→RED, code-6)

- **Quando** uma peça é colocada num espaço já ocupado por outra peça, **então** lance um RuntimeException

```
@Test
public void quandoLocalOcupadoEntaoRuntimeException() {
    assertThrows(RuntimeException.class, () -> {
        ticTacToe.play(2, 1);
        ticTacToe.play(2, 1);
    });
}
```



```
✓ Tests failed: 1, passed: 2 of 3 tests - 19 ms
TicTacToeTest (org.example) 19 ms
  ✓ quandoXForaTabuleiroEntaoRuntimeException() 16 ms
  ✗ quandoLocalOcupadoEntaoRuntimeException() 2 ms
  ✓ quandoYForaTabuleiroEntaoRuntimeException() 1 ms
  "C:\Program Files\OpenJDK\openjdk-11.0.15_10\bin\java.exe" ...
  org.opentest4j.AssertionFailedError: Expected java.lang.RuntimeException to be thrown, but nothing was thrown.
  <3 internal lines>
  at org.example.TicTacToeTest.quandoLocalOcupadoEntaoRuntimeException(TicTacToeTest.java:33) <31 internal lines>
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1541) <9 internal lines>
  at java.base/java.util.ArrayList.forEach(ArrayList.java:1541) <27 internal lines>
  Process finished with exit code -1
```

Codificação

Requisito 1 Teste 3 (RED→GREEN, code-7)

- Agora precisamos criar uma estrutura de dados para representar o tabuleiro
- Matriz de caracteres 3x3
- Importante que nessa fase ainda não estamos preocupados em guardar o jogador (player) que fez a jogada (X ou O)


```
private Character[][] board = {
    {'\0', '\0', '\0'},
    {'\0', '\0', '\0'},
    {'\0', '\0', '\0'}
};
```

```
public void play(int x, int y) {
    if (x < 1 || x > 3) {
        throw new RuntimeException("X esta fora do tabuleiro");
    } else if (y < 1 || y > 3) {
        throw new RuntimeException("Y esta fora do tabuleiro");
    }

    if (board[x - 1][y - 1] != '\0') {
        throw new RuntimeException("Espaco ocupado");
    } else {
        board[x - 1][y - 1] = 'X';
    }
}
```

✓ ✓ TicTacToeTest (org.example)	18 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	17 ms
✓ quandoLocalOcupadoEntaoRuntimeException()	1 ms
✓ quandoYForaTabuleiroEntaoRuntimeException()	

Codificação

Requisito 1 Teste 3 (GREEN→REFACT, code-8)

- Atendemos até aqui os 3 testes do requisito 1
- Pode não estar bem claro o que método play faz
- Podemos refatorar o código para separar conceitos
- Código precisa ser mais legível
- Vantagem do TDD: com os testes já feitos, podemos refatorar de modo despreocupado porque os testes devem passar
 - Como sempre todos os testes devem ser executados, dificilmente teremos efeitos colaterais
- Execute todos os testes e verifique se continuam passando

```
public void play(int x, int y) {  
    checkAxis(x);  
    checkAxis(y);  
    setBox(x, y);  
}
```

✓ TicTacToeTest (org.example)	20 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	18 ms
✓ quandoLocalOcupadoEntaoRuntimeException()	1 ms
✓ quandoYForaTabuleiroEntaoRuntimeException()	1 ms

```
private void checkAxis(int axis) {  
    if (axis < 1 || axis > 3) {  
        throw new RuntimeException("X esta fora do tabuleiro");  
    }  
}  
  
private void setBox(int x, int y) {  
    if (board[x - 1][y - 1] != '\0') {  
        throw new RuntimeException("Local esta ocupado");  
    } else {  
        board[x - 1][y - 1] = 'X';  
    }  
}
```

Requisito 2 – Suporte a 2 jogadores

- Deve existir uma maneira de se saber qual jogador da próxima jogada
- Podemos quebrar este requisito em 3 testes
 - Teste 1 : **Dada** a primeira jogada (turn), **quando** pedir pela próxima jogada **então** precisa ser do jogador X
 - Teste 2 : **Dada** a situação da última jogada ter sido do jogador X, **quando** pedir pela próxima jogada **então** deve ser do jogador O
 - Teste 3 : **Dada** a situação da última jogada ter sido do jogador O, **quando** pedir pela próxima jogada **então** deve ser do jogador X

Codificação – Requisito 2 Teste 1 (GREEN→RED, code-9)

- **Dada** a primeira jogada (turn), **quando** pedir pela próxima jogada **então** precisa ser do jogador X

```
@Test
public void dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX() {
    assertEquals('X', ticTacToe.nextPlayer());
}
```

Codificação

Requisito 2 Teste 1 (RED→GREEN, code-10)

- **Dada** a primeira jogada (turn), **quando** pedir pela próxima jogada **então** precisa ser do jogador X
- Observe que a implementação é muito simples mas que no teste 2 seremos forçados a refiná-la

```
public char nextPlayer() {  
    return 'X';  
}
```

✓ TicTacToeTest (org.example)	19 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	17 ms
✓ quandoLocalOcupadoEntaoRuntimeException()	1 ms
✓ quandoYForaTabuleiroEntaoRuntimeException()	
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	1 ms

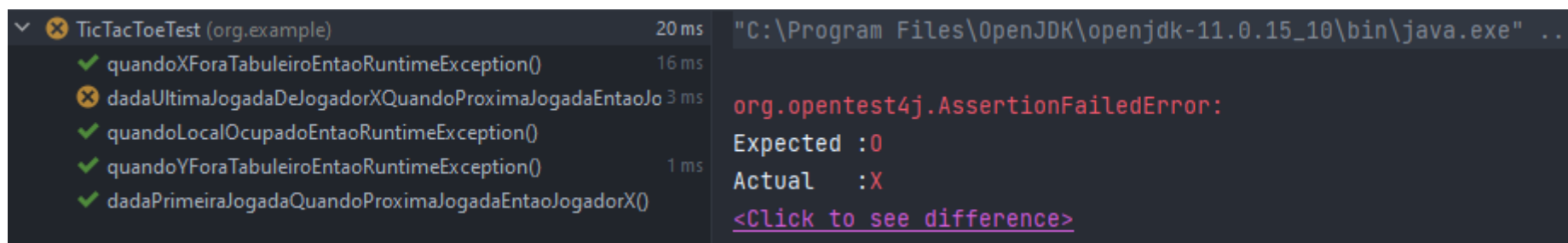
Codificação

Requisito 2 Teste 2 (GREEN→RED, code-11)

- **Dada** a situação da última jogada ter sido do jogador X, **quando** pedir pela próxima jogada **então** deve ser do jogador O

@Test

```
public void dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO() {  
    ticTacToe.play(1, 1);  
    assertEquals('O', ticTacToe.nextPlayer());  
}
```



Codificação

Requisito 2 Teste 2 (RED→GREEN, code-12)

- **Dada** a situação da última jogada ter sido do jogador X, **quando** pedir pela próxima jogada **então** deve ser do jogador O

```
private char lastPlayer = '\0';

public void play(int x, int y) {
    checkAxis(x);
    checkAxis(y);
    setBox(x, y);
    lastPlayer = nextPlayer();
}

public char nextPlayer() {
    if (lastPlayer == 'X') {
        return 'O';
    }
    return 'X';
}
```

✓ TicTacToeTest (org.example)	19 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	15 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJo	1 ms
✓ quandoLocalOcupadoEntaoRuntimeException()	1 ms
✓ quandoYForaTabuleiroEntaoRuntimeException()	2 ms
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	

Codificação

Requisito 2 Teste 3 (GREEN→RED, code-13)

- **Dada** a situação da última jogada ter sido do jogador O, **quando** pedir pela próxima jogada **então** deve ser do jogador X

```
@Test
public void dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX() {
    ticTacToe.play(1, 1); // vez do Jogador X
    ticTacToe.play(1, 2); // vez do Jogador O

    assertEquals('X', ticTacToe.nextPlayer());
}
```

✓ TicTacToeTest (org.example)	20 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	15 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO()	2 ms
✓ quandoLocalOcupadoEntaoRuntimeException()	
✓ quandoYForaTabuleiroEntaoRuntimeException()	1 ms
✓ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()	1 ms
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	1 ms



Há algo errado...

Codificação

Requisito 2 Teste 3 (RED→GREEN, code-14)

- **Dada** a situação da última jogada ter sido do jogador O, **quando** pedir pela próxima jogada **então** deve ser do jogador X
- O teste foi colocado e deveria ter quebrado.
- Importante: se escrevermos um teste e, sem implementar nada, ele passar, deve ser descartado. Dica: use @Disabled se preciso

```
@Disabled
@Test
public void dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX() {
    ticTacToe.play(1, 1); // vez do Jogador X
    ticTacToe.play(1, 2); // vez do Jogador O

    assertEquals('X', ticTacToe.nextPlayer());
}
```

✓ TicTacToeTest (org.example)	21 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	16 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO()	2 ms
✓ quandoLocalOcupadoEntaoRuntimeException()	1 ms
✓ quandoYForaTabuleiroEntaoRuntimeException()	1 ms
⊗ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()	
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	1 ms

Requisito 3 – Condições de vitória

- Um jogador vence quando for o primeiro a conectar uma linha horizontal do tabuleiro ou vertical ou alguma das diagonais com todas as suas peças jogadas.
- Podemos quebrar este requisito em 5 testes
 - Teste 1 : **Quando** jogar **então** não existe vencedor
 - Teste 2 : **Quando** jogar **E** preencher toda linha horizontal **então** vence
 - Teste 3 : **Quando** jogar **E** preencher toda linha vertical **então** vence
 - Teste 4 : **Quando** jogar **E** preencher toda diagonal cima-baixo **então** vence
 - Teste 5 : **Quando** jogar **E** preencher toda diagonal baixo-cima **então** vence

Codificação – Requisito 3 Teste 1 (GREEN→RED, code-15)

- **Quando** jogar **então** não existe vencedor

```
@Test
public void quandoJogarEntaoSemVencedor() {
    String actual = ticTacToe.play(1, 1);
    assertEquals("Sem vencedor", actual);
}
```

Codificação – Requisito 3 Teste 1 (RED→GREEN, code-16)

- **Quando** jogar **então** não existe vencedor

```
public String play(int x, int y) {  
    checkAxis(x);  
    checkAxis(y);  
    setBox(x, y);  
    lastPlayer = nextPlayer();  
  
    return "Sem vencedor";  
}
```

✓ TicTacToeTest (org.example)	18 ms
✓ quandoJogarEntaoSemVencedor()	15 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	2 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO()	
✓ quandoLocalOcupadoEntaoRuntimeException()	1 ms
✓ quandoYForaTabuleiroEntaoRuntimeException()	
⊗ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()	
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	

Codificação

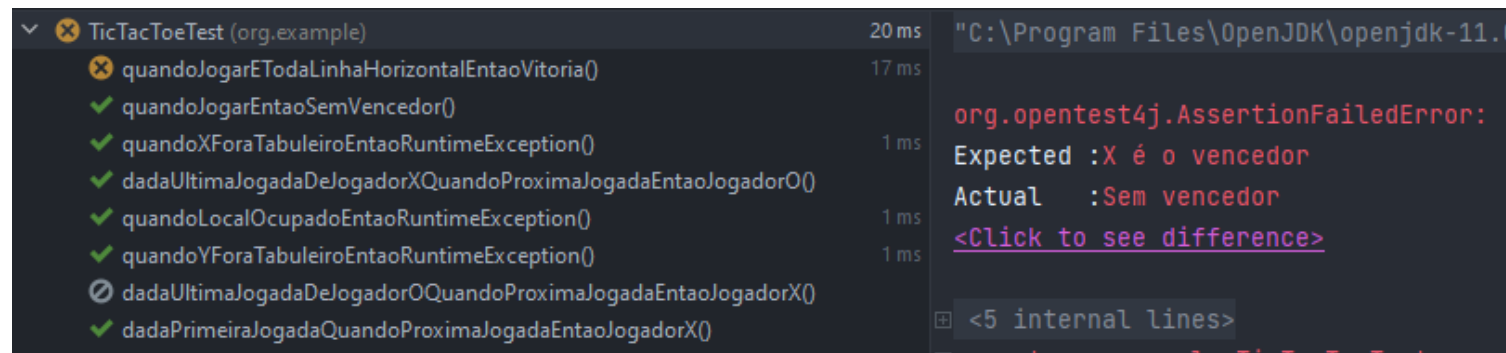
Requisito 3 Teste 2 (GREEN→RED, code-17)

- Quando jogar E preencher toda linha horizontal **então** vence

```
@Test
public void quandoJogarETodaLinhaHorizontalEntaoVitoria() {
    ticTacToe.play(1, 1); // X
    ticTacToe.play(1, 2); // O
    ticTacToe.play(2, 1); // X
    ticTacToe.play(2, 2); // O

    String actual = ticTacToe.play(3, 1); // X
    assertEquals("X é o vencedor", actual);
}
```

(1, 1) X	(2, 1) X	(3, 1) X
(1, 2) O	(2, 2) O	(3, 2)
(1, 3) 	(2, 3) 	(3, 3)



Codificação

Requisito 3 Teste 2 (RED→GREEN, code-18)

- **Quando** jogar **E** preencher toda linha horizontal **então** vence

```
public String play(int x, int y) {
    checkAxis(x);
    checkAxis(y);
    lastPlayer = nextPlayer();
    setBox(x, y, lastPlayer);

    for (int index = 0; index < 3; index++) {
        if (board[0][index] == lastPlayer
            && board[1][index] == lastPlayer
            && board[2][index] == lastPlayer) {
            return lastPlayer + " é o vencedor";
        }
    }

    return "Sem vencedor";
}
```

```
private void setBox(int x, int y, char lastPlayer) {
    if (board[x - 1][y - 1] != '\0') {
        throw new RuntimeException("Local esta ocupado");
    } else {
        board[x - 1][y - 1] = lastPlayer;
    }
}
```

✓ TicTacToeTest (org.example)	23 ms
✓ quandoJogarETodaLinhaHorizontalEntaoVitoria()	19 ms
✓ quandoJogarEntaoSemVencedor()	1 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	2 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO()	
✓ quandoLocalOcupadoEntaoRuntimeException()	
✓ quandoYForaTabuleiroEntaoRuntimeException()	
⊗ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()	
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	1 ms

Codificação

Requisito 3 Teste 2 (GREEN→REFACT, code-19)

- **Quando** jogar **E** preencher toda linha horizontal **então** vence
- Chegou a hora de refatorar
- Por que satisfazer os testes precisa ser prioridade?
 - Propósito de cumprir a cobertura de código o mais rápido possível

```
private static final int SIZE = 3;
```

```
public String play(int x, int y) {  
    checkAxis(x);  
    checkAxis(y);  
    lastPlayer = nextPlayer();  
    setBox(x, y, lastPlayer);  
  
    if (isWin()) {  
        return lastPlayer + " é o vencedor";  
    }  
  
    return "Sem vencedor";  
}
```

```
private boolean isWin() {
    for (int index = 0; index < SIZE; index++) {
        if (board[0][index] + board[1][index] + board[2][index] == (lastPlayer * SIZE)) {
            return true;
        }
    }

    return false;
}
```

$$264 = 88 * 3$$

✓ TicTacToeTest (org.example)	23 ms
✓ quandoJogarETodaLinhaHorizontalEntaoVitoria()	18 ms
✓ quandoJogarEntaoSemVencedor()	
✓ quandoXForaTabuleiroEntaoRuntimeException()	3 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO()	1 ms
✓ quandoLocalOcupadoEntaoRuntimeException()	1 ms
✓ quandoYForaTabuleiroEntaoRuntimeException()	
⊗ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()	
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	

```
char lastPlayer = 'X'
jshell> lastPlayer * 3
$11 ==> 264

jshell> 'X'*1
$9 ==> 88
```

Codificação

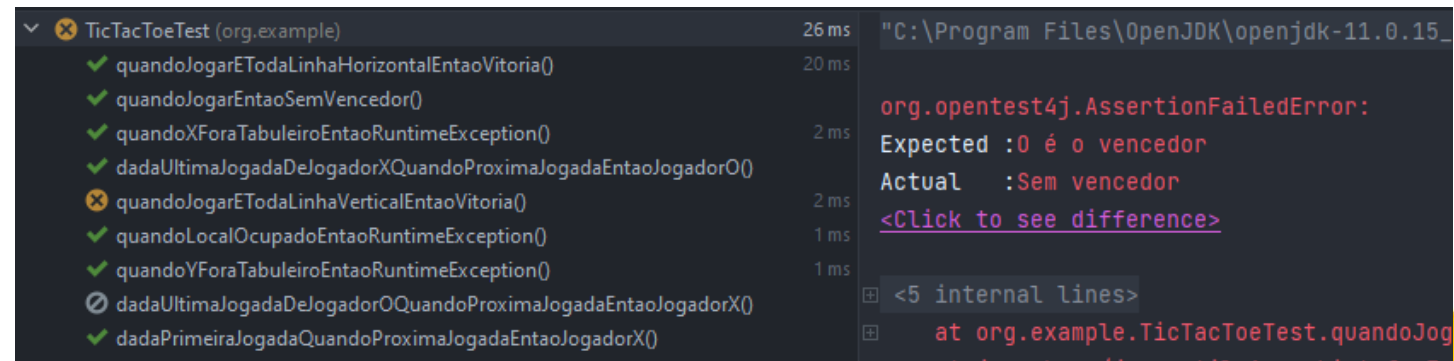
Requisito 3 Teste 3 (GREEN→RED, code-20)

- Quando jogar E preencher toda linha vertical **então** vence

```
@Test
public void quandoJogarETodaLinhaVerticalEntaoVitoria() {
    ticTacToe.play(2, 1);
    ticTacToe.play(1, 1);
    ticTacToe.play(3, 1);
    ticTacToe.play(1, 2);
    ticTacToe.play(2, 2);

    String actual = ticTacToe.play(1, 3);
    assertEquals("O é o vencedor", actual);
}
```

(1, 1) O	(2, 1) X	(3, 1) X
(1, 2) O	(2, 2) X	(3, 2)
(1, 3) O	(2, 3)	(3, 3)



Codificação

Requisito 3 Teste 3 (RED→GREEN, code-21)

- **Quando** jogar **E** preencher toda linha vertical **então** vence
- Quais mudanças ocorreram no código de isWin?

```
private boolean isWin() {  
    int playerTotal = lastPlayer * 3;  
  
    for (int index = 0; index < SIZE; index++) {  
        if (board[0][index] + board[1][index] + board[2][index] == playerTotal) {  
            return true;  
        } else if (board[index][0] + board[index][1] + board[index][2] == playerTotal) {  
            return true;  
        }  
    }  
  
    return false;  
}
```

✓ TicTacToeTest (org.example)	22 ms
✓ quandoJogarETodaLinhaHorizontalEntaoVitoria()	19 ms
✓ quandoJogarEntaoSemVencedor()	1 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	2 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO()	
✓ quandoJogarETodaLinhaVerticalEntaoVitoria()	
✓ quandoLocalOcupadoEntaoRuntimeException()	
✓ quandoYForaTabuleiroEntaoRuntimeException()	
⊗ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()	
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	

Codificação

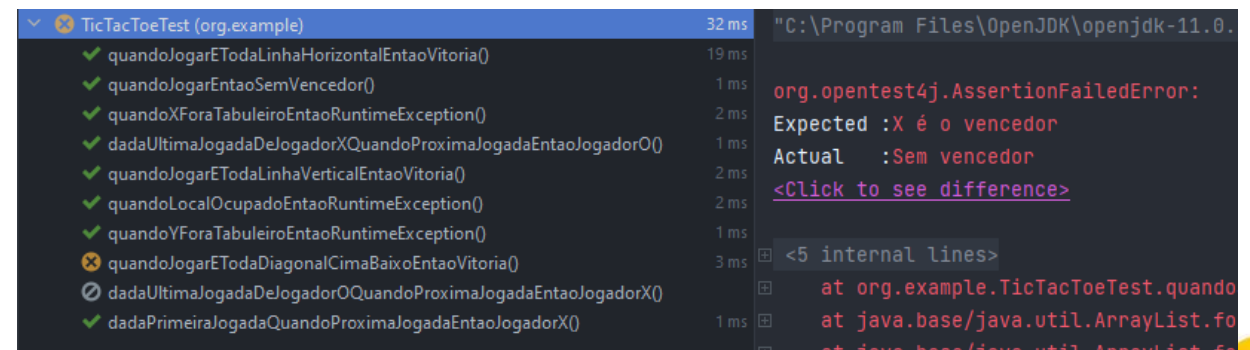
Requisito 3 Teste 4 (GREEN→RED, code-22)

- Quando jogar E preencher toda diagonal cima-baixo **então** vence

```
@Test
public void quandoJogarETodaDiagonalCimaBaixoEntaoVitoria() {
    ticTacToe.play(1, 1);
    ticTacToe.play(1, 2);
    ticTacToe.play(2, 2);
    ticTacToe.play(1, 3);

    String actual = ticTacToe.play(3, 3);
    assertEquals("X é o vencedor", actual);
}
```

(1, 1) X	(2, 1)	(3, 1)
(1, 2) O	(2, 2) X	(3, 2)
(1, 3) X	(2, 3)	(3, 3) X



Codificação

Requisito 3 Teste 4 (RED→GREEN, code-23)

- **Quando** jogar **E** preencher toda diagonal cima-baixo **então** vence

```
private boolean isWin() {
    int playerTotal = lastPlayer * 3;

    for (int index = 0; index < SIZE; index++) {
        if (board[0][index] + board[1][index] + board[2][index] == playerTotal) {
            return true;
        } else if (board[index][0] + board[index][1] + board[index][2] == playerTotal) {
            return true;
        }
    }

    if (board[0][0] + board[1][1] + board[2][2] == playerTotal) {
        return true;
    }

    return false;
}
```

✓ TicTacToeTest (org.example)	22 ms
✓ quandoJogarETodaLinhaHorizontalEntaoVitoria()	17 ms
✓ quandoJogarEntaoSemVencedor()	1 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	1 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO()	
✓ quandoJogarETodaLinhaVerticalEntaoVitoria()	
✓ quandoLocalOcupadoEntaoRuntimeException()	1 ms
✓ quandoYForaTabuleiroEntaoRuntimeException()	2 ms
✓ quandoJogarETodaDiagonalCimaBaixoEntaoVitoria()	
⊗ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()	
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	

Codificação

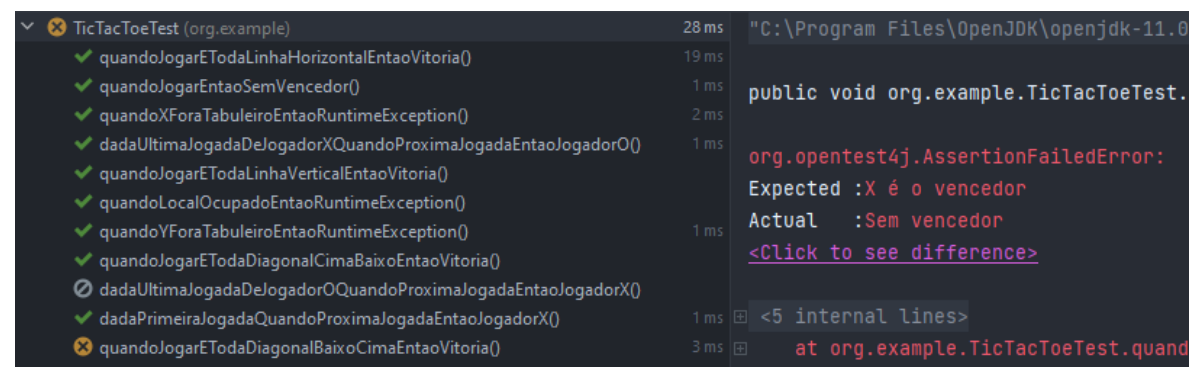
Requisito 3 Teste 5 (GREEN→RED, code-24)

- Quando jogar E preencher toda diagonal baixo-cima **então** vence

```
@Test
public void quandoJogarETodaDiagonalBaixoCimaEntaoVitoria() {
    ticTacToe.play(1, 3);
    ticTacToe.play(1, 1);
    ticTacToe.play(2, 2);
    ticTacToe.play(1, 2);

    String actual = ticTacToe.play(3, 1);
    assertEquals("X é o vencedor", actual);
}
```

(1, 1) O	(2, 1)	(3, 1) X
(1, 2) O	(2, 2) X	(3, 2)
(1, 3) X	(2, 3)	(3, 3)



Codificação

Requisito 3 Teste 5 (GREEN→REFACT, code-25)

- **Quando** jogar **E** preencher toda diagonal baixo-cima **então** vence

✓ TicTacToeTest (org.example)	26 ms
✓ quandoJogarETodaLinhaHorizontalEntaoVitoria()	19 ms
✓ quandoJogarEntaoSemVencedor()	1 ms
✓ quandoXForaTabuleiroEntaoRuntimeException()	2 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO()	
✓ quandoJogarETodaLinhaVerticalEntaoVitoria()	
✓ quandoLocalOcupadoEntaoRuntimeException()	
✓ quandoYForaTabuleiroEntaoRuntimeException()	1 ms
✓ quandoJogarETodaDiagonalCimaBaixoEntaoVitoria()	1 ms
⊗ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()	
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	1 ms
✓ quandoJogarETodaDiagonalBaixoCimaEntaoVitoria()	1 ms


```
private boolean isWin() {
```

```
    int playerTotal = lastPlayer * 3;
```

```
    char diagonal1 = '\0';
```

```
    char diagonal2 = '\0';
```

```
    for (int index = 0; index < SIZE; index++) {
```

```
        diagonal1 += board[index][index];
```

```
        diagonal2 += board[index][SIZE - index - 1];
```

```
        if (board[0][index] + board[1][index] + board[2][index] == playerTotal) {
```

```
            return true;
```

```
        } else if (board[index][0] + board[index][1] + board[index][2] == playerTotal) {
```

```
            return true;
```

```
        }
```

```
    }
```

```
    if (diagonal1 == playerTotal || diagonal2 == playerTotal) {
```

```
        return true;
```

```
    }
```

```
    return false;
```

```
}
```

Para index = 0, (0, 2 = SIZE - 0 - 1)

Para index = 1, (1, 1 = SIZE - 1 - 1)

Para index = 2, (2, 0 = SIZE - 2 - 1)

(1, 1) O	(2, 1)	(3, 1) X
(1, 2) O	(2, 2) X	(3, 2)
(1, 3) X	(2, 3)	(3, 3)

Requisito 4 – Condições de empate

- Quando todos os locais já forem preenchidos, ocorreu um empate
- Este requisito tem apenas 1 teste
 - Teste 1 : **Quando** todos os locais estiverem preenchidos **então** houve um empate

Codificação

Requisito 4 Teste 1 (GREEN→RED, code-26)

- **Quando** todos os locais estiverem preenchidos **então** houve um empate

```
@Test
public void quandoTodosLocaisPreenchidosEntaoEmpate() {
    ticTacToe.play(1, 1);
    ticTacToe.play(1, 2);
    ticTacToe.play(1, 3);
    ticTacToe.play(2, 1);
    ticTacToe.play(2, 3);
    ticTacToe.play(2, 2);
    ticTacToe.play(3, 1);
    ticTacToe.play(3, 3);

    String actual = ticTacToe.play(3, 2);
    assertEquals("O jogo empatou", actual);
}
```

(1, 1) X	(2, 1) O	(3, 1) X
(1, 2) O	(2, 2) O	(3, 2) X
(1, 3) X	(2, 3) X	(3, 3) O

```
TicTacToeTest (org.example) 28 ms
  ✓ quandoJogarETodaLinhaHorizontalEntaoVitoria() 20 ms
  ✓ quandoJogarEntaoSemVencedor() 1 ms
  ✓ quandoXForaTabuleiroEntaoRuntimeException() 2 ms
  ✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO() 1 ms
  ✓ quandoJogarETodaLinhaVerticalEntaoVitoria() 1 ms
  ✗ quandoTodosLocaisPreenchidosEntaoEmpate() 2 ms
  ✓ quandoLocalOcupadoEntaoRuntimeException() 1 ms
  ✓ quandoYForaTabuleiroEntaoRuntimeException()
  ✓ quandoJogarETodaDiagonalCimaBaixoEntaoVitoria()
  ⚙ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()
  ✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()
  ✓ quandoJogarETodaDiagonalBaixoCimaEntaoVitoria()

Expected :0 jogo empatou
Actual    :Sem vencedor
<Click to see difference>

<5 internal lines>
  at org.example.TicTacTo
  at java.base/java.util.
  at java.base/java.util.

public void org.example.Tic
```

Codificação

Requisito 4 Teste 1 (RED→GREEN, code-27)

- **Quando** todos os locais estiverem preenchidos **então** houve um empate

✓ TicTacToeTest (org.example)	23 ms
✓ quandoJogarETodaLinhaHorizontalEntaoVitoria()	18 ms
✓ quandoJogarEntaoSemVencedor()	
✓ quandoXForaTabuleiroEntaoRuntimeException()	2 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO()	
✓ quandoJogarETodaLinhaVerticalEntaoVitoria()	
✓ quandoTodosLocaisPreenchidosEntaoEmpate()	
✓ quandoLocalOcupadoEntaoRuntimeException()	1 ms
✓ quandoYForaTabuleiroEntaoRuntimeException()	1 ms
✓ quandoJogarETodaDiagonalCimaBaixoEntaoVitoria()	1 ms
⊗ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()	
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	
✓ quandoJogarETodaDiagonalBaixoCimaEntaoVitoria()	

```
public String play(int x, int y) {
    checkAxis(x);
    checkAxis(y);
    lastPlayer = nextPlayer();
    setBox(x, y, lastPlayer);
```

```
    if (isWin()) {
        return lastPlayer + " é o vencedor";
    } else if (isDraw()) {
        return "O jogo empatou";
    }
    return "Sem vencedor";
}
```

```
private boolean isDraw() {
    for (int x = 0; x < SIZE; x++) {
        for (int y = 0; y < SIZE; y++) {
            if (board[x][y] == '\0') {
                return false;
            }
        }
    }
    return true;
}
```

- **isWin**: só indica vitória caso tenha ocorrido jogadas e alguma condição de vitória for alcançada
- Caso nenhuma condição de vitória for alcançada, existe possibilidade do empate
- **isDraw**: só indica empate caso todos os locais tiverem sido jogados.
 - O código implementa a seguinte forma: faça uma varredura por todo o tabuleiro. Se em algum local para jogar estiver desocupado (símbolo '\0'), indique que não há condição de empate porque existem ainda jogadas a serem feitas com chance de vitória
- **“Sem vencedor”**: caso nenhuma condição de vitória foi alcançada e sequer deu empate, então por enquanto ninguém venceu ainda e mais jogadas podem ser feitas

Surgiu uma lógica de verificação de condição naturalmente. É uma das grandes vantagens do TDD.

Codificação

Requisito 4 Teste 1 (GREEN→REFACT, code-28)

- **Quando** todos os locais estiverem preenchidos **então** houve um empate

```
public String play(int x, int y) {  
    checkAxis(x);  
    checkAxis(y);  
    lastPlayer = nextPlayer();  
    setBox(x, y, lastPlayer);  
  
    if (isWin(x, y)) {  
        return lastPlayer + " é o vencedor";  
    } else if (isDraw()) {  
        return "O jogo empatou";  
    }  
  
    return "Sem vencedor";  
}
```

✓ TicTacToeTest (org.example)	28 ms
✓ quandoJogarETodaLinhaHorizontalEntaoVitoria()	20 ms
✓ quandoJogarEntaoSemVencedor()	
✓ quandoXForaTabuleiroEntaoRuntimeException()	8 ms
✓ dadaUltimaJogadaDeJogadorXQuandoProximaJogadaEntaoJogadorO()	
✓ quandoJogarETodaLinhaVerticalEntaoVitoria()	
✓ quandoTodosLocaisPreenchidosEntaoEmpate()	
✓ quandoLocalOcupadoEntaoRuntimeException()	
✓ quandoYForaTabuleiroEntaoRuntimeException()	
✓ quandoJogarETodaDiagonalCimaBaixoEntaoVitoria()	
⊗ dadaUltimaJogadaDeJogadorOQuandoProximaJogadaEntaoJogadorX()	
✓ dadaPrimeiraJogadaQuandoProximaJogadaEntaoJogadorX()	
✓ quandoJogarETodaDiagonalBaixoCimaEntaoVitoria()	

```
private boolean isWin(int x, int y) {
    int playerTotal = lastPlayer * 3;
    char diagonal1, diagonal2, horizontal, vertical;
    diagonal1 = diagonal2 = horizontal = vertical = '\0';

    for (int index = 0; index < SIZE; index++) {
        diagonal1 += board[index][index];
        diagonal2 += board[index][SIZE - index - 1];
        horizontal += board[index][y - 1];
        vertical += board[x - 1][index];
    }

    if (diagonal1 == playerTotal
        || diagonal2 == playerTotal
        || horizontal == playerTotal
        || vertical == playerTotal) {
        return true;
    }

    return false;
}
```



Code Coverage

Visualizar a pontuação de cobertura de testes com JaCoCo

TicTacToe

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● isWin(int, int)	<div><div></div></div>	100%	<div><div></div></div>	100%	0	6	0	10	0	1
● TicTacToe()	<div><div></div></div>	100%		n/a	0	1	0	6	0	1
● play(int, int)	<div><div></div></div>	100%	<div><div></div></div>	100%	0	3	0	9	0	1
● setBox(int, int, char)	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	4	0	1
● isDraw()	<div><div></div></div>	100%	<div><div></div></div>	100%	0	4	0	5	0	1
● checkAxis(int)	<div><div></div></div>	100%	<div><div></div></div>	75%	1	3	0	3	0	1
● nextPlayer()	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	3	0	1
Total	0 of 266	100%	1 of 28	96%	1	21	0	40	0	7

Demonstração de que TDD nos auxilia com alta cobertura desde o início