UFPB - Campus IV - Lic. em Ciência da Computação

Análise e Projeto de Sistemas

Professor: Rodrigo Vilar

Test-driven development

Escreva os testes antes do código

Esta aula se baseia no artigo:

Test Driven Development – Part I: TFC. Venkat Subramaniam.
http://www.agiledeveloper.com/download.aspx

Objetivos:

- Ilustrar os benefícios de escrever os testes antes da implementação
- Usar JUnit para escrever casos de teste

Gaste 10 minutos modelando uma solução para este problema…

Queremos implementar um “Jogo da Velha” nesse exercício. Existem dois usuários no sistema. Um deles marcará as células com ‘x’ e o outro com ‘o’. Existem 3 colunas e 3 linhas. O primeiro usuário de uma partida precisa indicar se utilizará ‘x’ ou ‘o’. Daí, o primeiro usuário deve marcar uma célula. O usuário só poderá marcar uma célula vazia. O jogo continua até que o jogador tenha formado uma sequência de 3 marcas na horizontal, vertical ou diagonal, ou que não haja mais células vazias. Se o jogo foi ganho, o vencedor é anunciado.

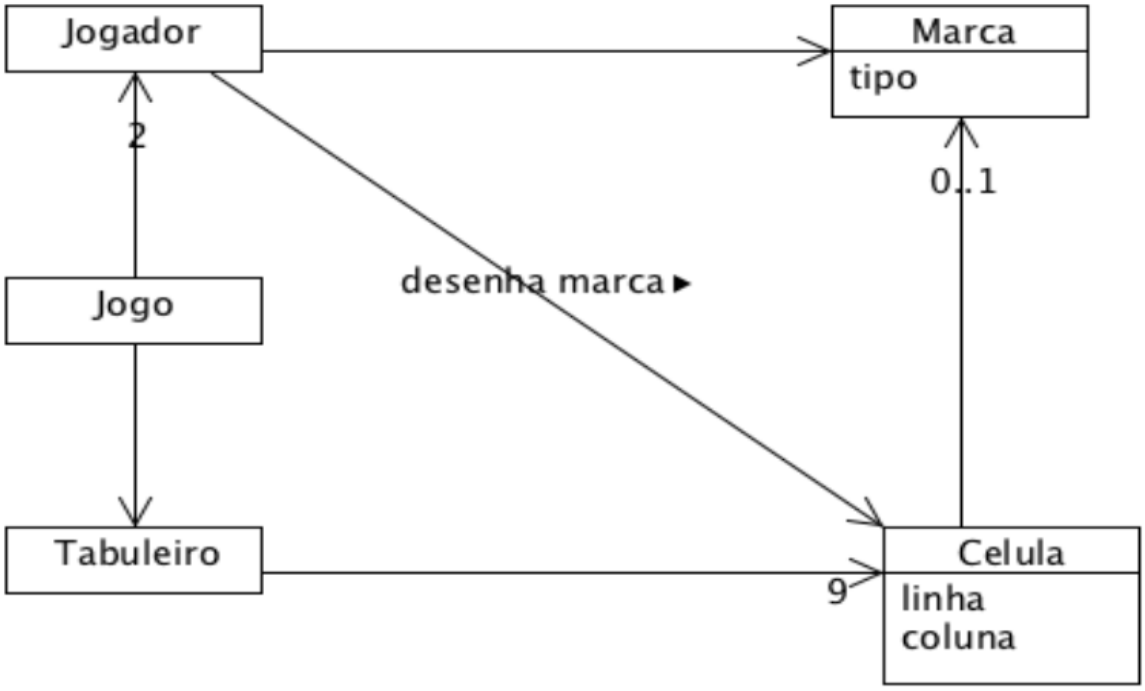
Vamos organizar os requisitos

1. Queremos implementar um “Jogo da Velha” nesse exercício.
2. Existem dois usuários no sistema.
3. Um marcará as células com ‘x’ e o outro com ‘o’.
4. Existem 3 colunas e 3 linhas.
5. O primeiro usuário de uma parXda precisa indicar se utilizará ‘x’ ou ‘o’.
6. Daí, o primeiro usuário deve marcar uma célula.
7. O usuário só poderá marcar uma célula vazia.
8. O jogo continua até que:
    a. O jogador tenha formado uma sequência de 3 marcas na horizontal, vertical ou diagonal,
    b. Ou que não haja mais células vazias.
9. Se o jogo foi ganho, o vencedor é anunciado.

Marcando as palavras-chave

1. Queremos implementar um “Jogo da Velha” nesse exercício.
2. Existem dois usuários no sistema.
3. Um marcará as células com ‘x’ e o outro com ‘o’.
4. Existem 3 colunas e 3 linhas.
5. O primeiro usuário de uma parXda precisa indicar se utilizará ‘x’ ou ‘o’.
6. Daí, o primeiro usuário deve marcar uma célula.
7. O usuário só poderá marcar uma célula vazia.
8. O jogo continua até que:
    a. O jogador tenha formado uma sequência de 3 marcas na horizontal, vertical ou diagonal,
    b. Ou que não haja mais células vazias.
9. Se o jogo foi ganho, o vencedor é anunciado.

Um exemplo de modelagem



Como você testaria esse código?

- Não testo! Chuck Norris não precisa testar!

Após a implementação:

- Testes manuais com uma UI
- Testes automáticos

Teste como um ato de verificação

- Você pode esquecer de verificar algo
- Manutenção dispendiosa

Escrevendo o teste antes

Você pensa:

- Como minha classe será invocada?
- Em vez de cair direto na implementação

Isso é um esforço de Projeto

- Gera um projeto mais simples e pragmático

QUANTO MAIS RÁPIDO SE PEGA NO TECLADO, MAIS SE DEMORA

PARA SOLTÁ-LO.

- Os testes guiarão o seu desenvolvimento
    - Por isso é *Test-DRIVEN development*

Como escrever testes antes?

O que você quer testar?

Pelo menos duas coisas:

- Cenário positivo – Happy day – tudo acontecendo na forma ideal
- Cenário negativo – Exceções e erros – o seu sistema precisa se preparar para essas coisas

Um caso de teste é uma sequência de:

- Ações - Passos, estímulos ao sistema
- Assertivas - verificações do estado do sistema

O sucesso do teste positivo

- O código faz o que você espera

O sucesso do teste negativo

- O código falha como você espera

Faça uma lista de testes

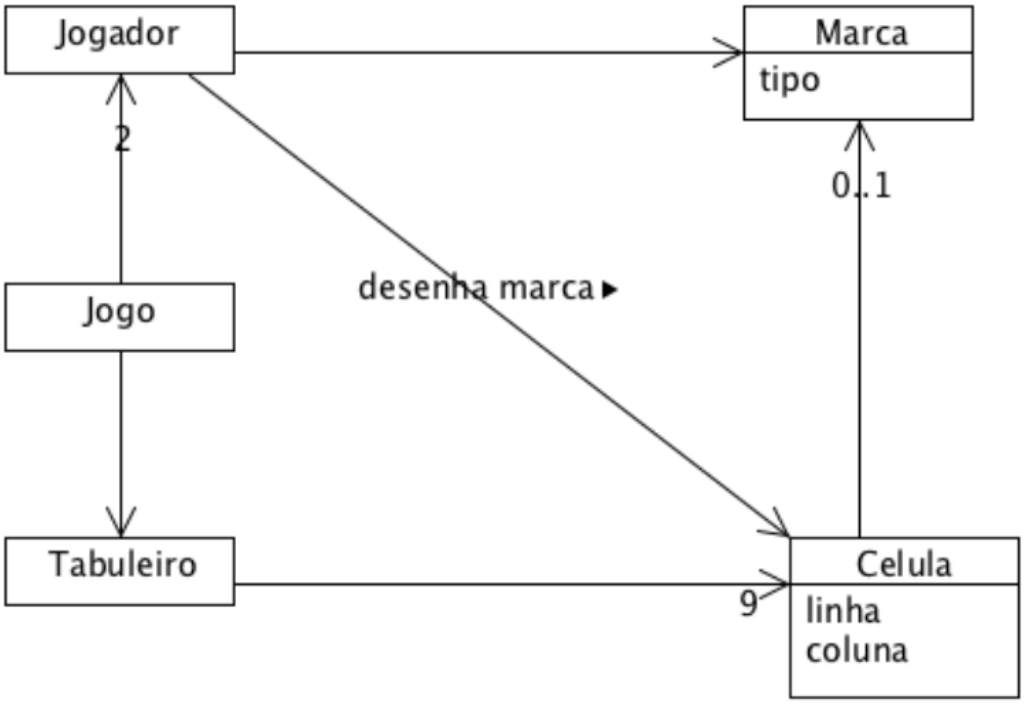
- Planeje alguns testes iniciais
- Escolha um para implementar
- Novas idéias surgirão e a lista aumentará
- Escreva o novo teste na lista
    - Você não precisa fazê-lo na hora que o descobre

MANTENHA A LISTA DE TESTES SEMPRE POR PERTO.

Por onde começamos?

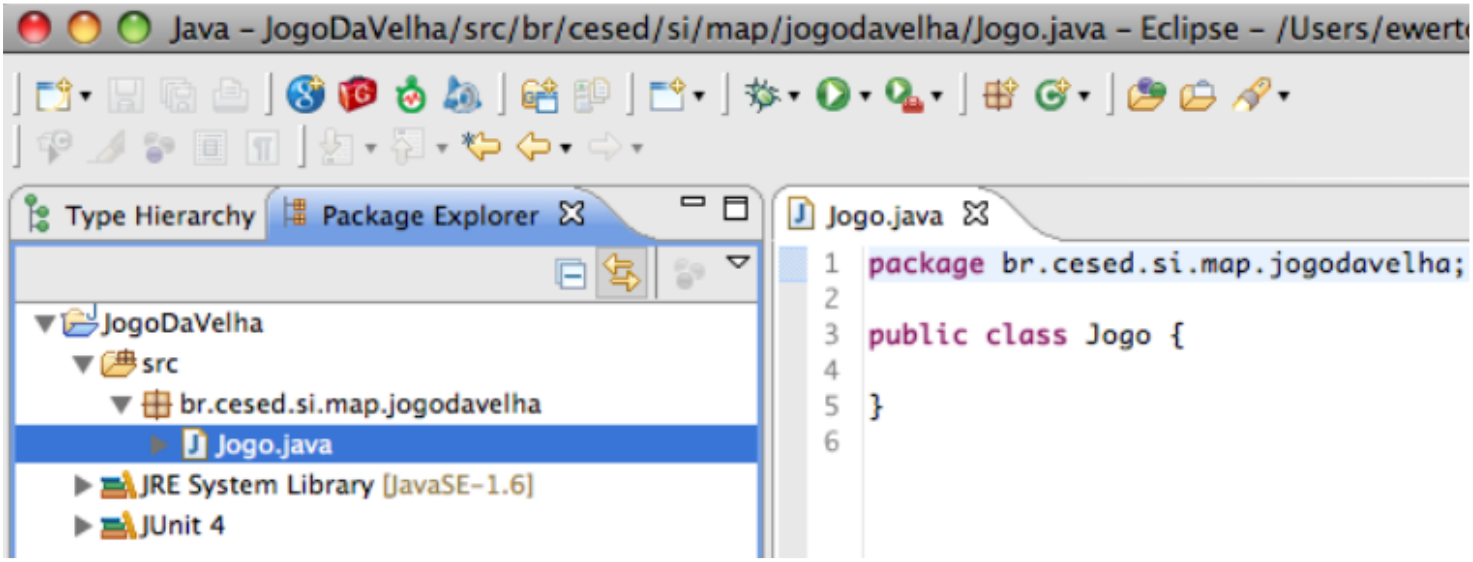
Qual a primeira classe que deve ser escrita?

Lembrando a nossa modelagem:



A classe Jogo agrega indiretamente todas as outras classes e é uma boa candidata para ser a Interface entre o sistema e o teste.

Criação de Projeto no Eclipse



- Incluir o JUnit 4 no classpath
- Criar a classe Jogo vazia
- Começar a planejar a lista de testes

LISTA DE TESTES v1
1. Criar um Jogo

Vamos iniciar a implementação desse teste e se mais idéias de teste surgem.

O que testar com apenas a ação de "Criar um Jogo"?

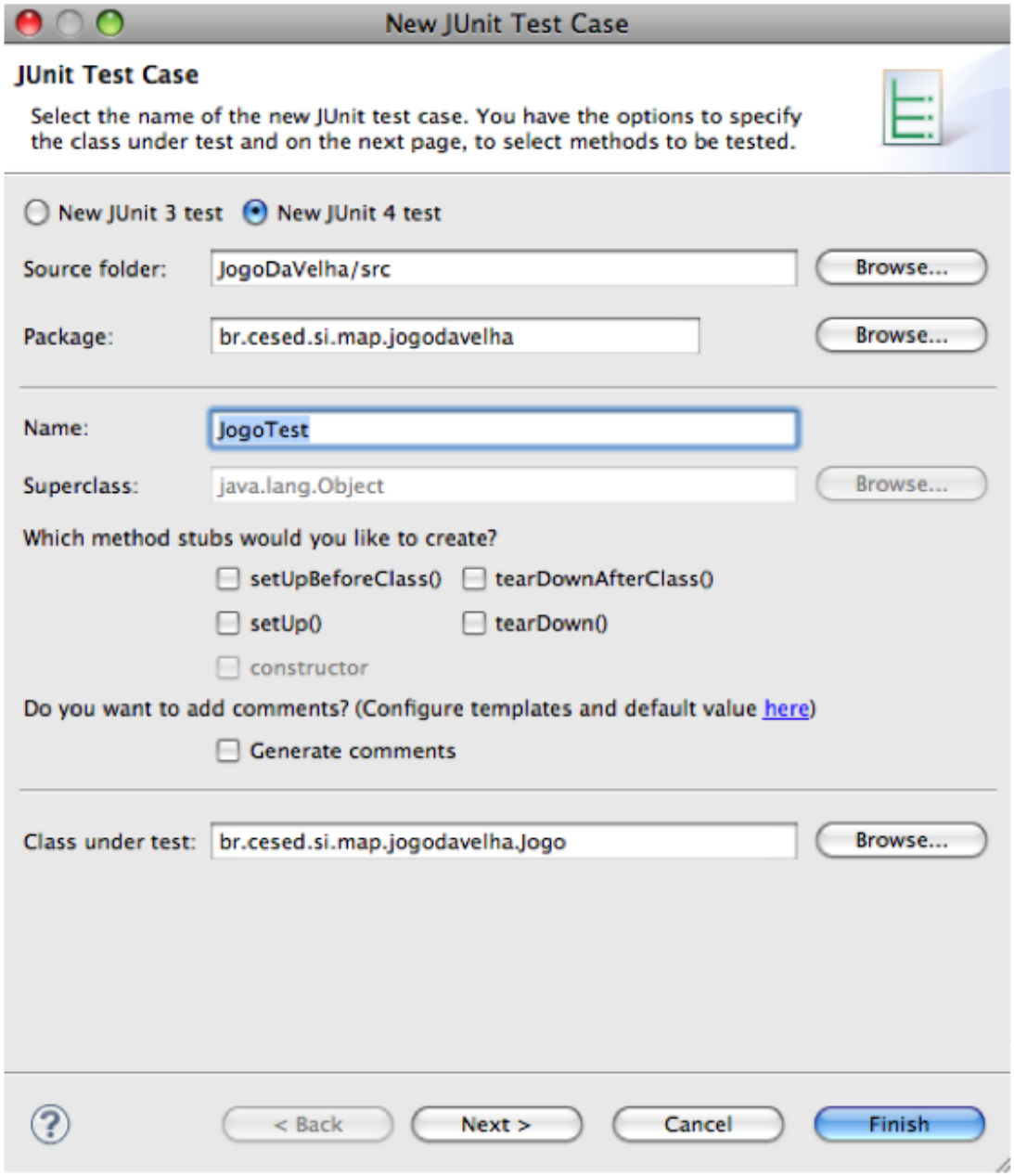
- Não tem muito o que fazer…
- Verificar se o jogo acabou! O que deve ser falso.

Portanto, o primeiro caso de teste será:

- Passo: criar um jogo
- Assertiva: verificar que o jogo não acabou

Assistente para Criação de Classes de teste no Eclipse

Menu de contexto sobre o pacote destino > *New*> *JUnit Test Case*



Código do primeiro teste

```
package br.cesed.si.map.jogodavelha;

import org.junit.Assert;
import org.junit.Test;

public class JogoTest {

  @Test
  public void criarJogo() {
    //Passo: criar um jogo
    Jogo jogo = new Jogo();
    //Assertiva: verificar que o jogo não acabou
    Assert.assertFalse("O jogo iniciou acabado", jogo.acabou());
  }
}
```

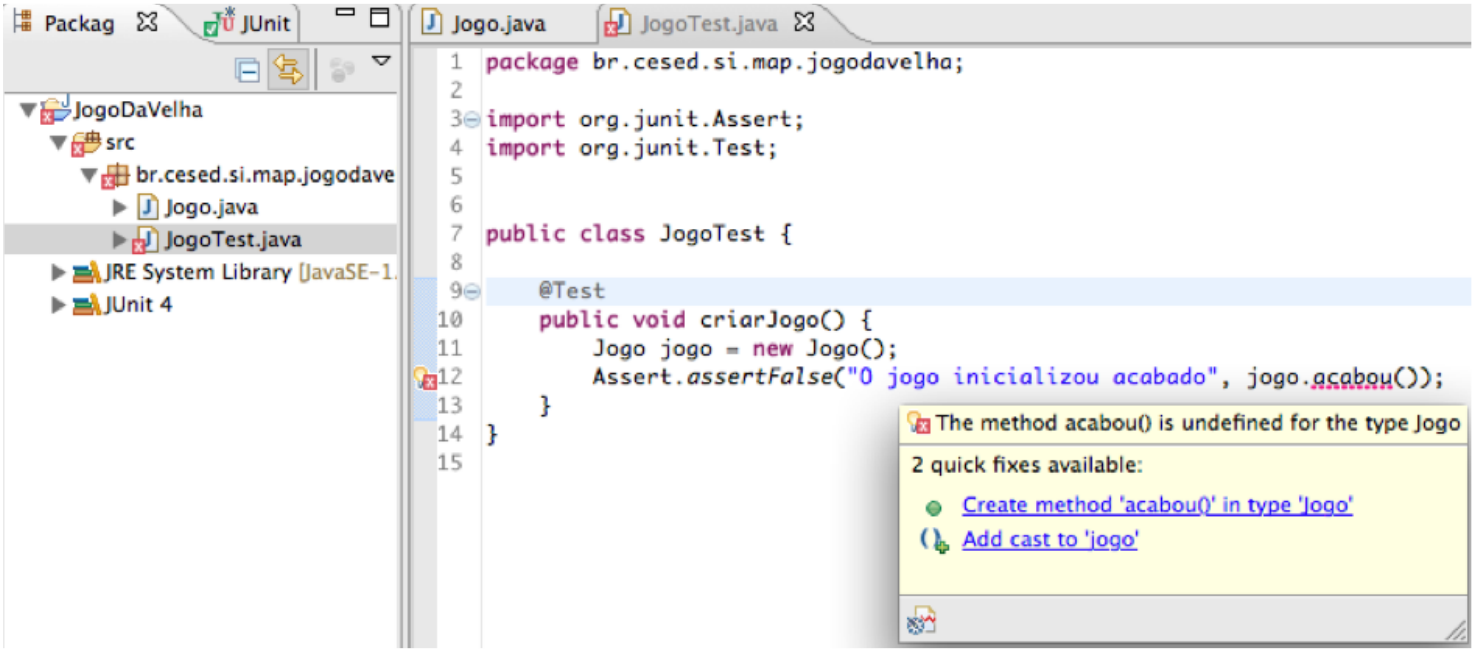
Mas o método acabou() não existe na classe Jogo!

O teste é quem vai dizer que métodos devem existir na classe Jogo.

EM TDD, OS TESTES DIRECIONAM O PROJETO DO SISTEMA.

O Eclipse pode nos ajudar a criar novos métodos:

1. Posicione o cursor sobre o erro de compilação;
2. Tecle <Control> + 1
3. Escolha a opção "Create method 'acabou()' in type 'Jogo'"



Implementação para atender ao primeiro teste

```
public class Jogo {
  public boolean acabou() {
    return false;
  }
}
```

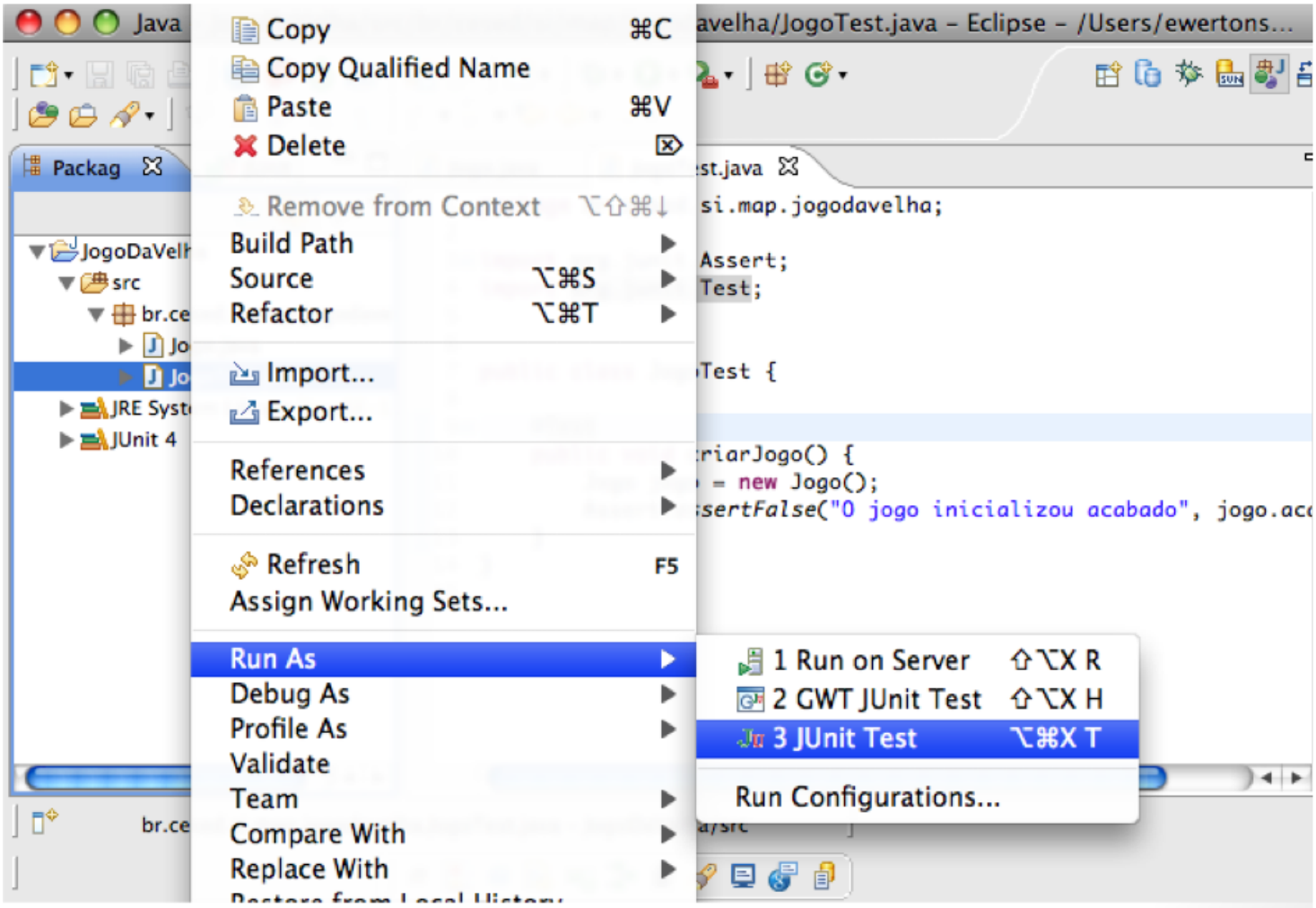
return false?

IMPLEMENTE APENAS O QUE É NECESSÁRIO PARA O TESTE PASSAR

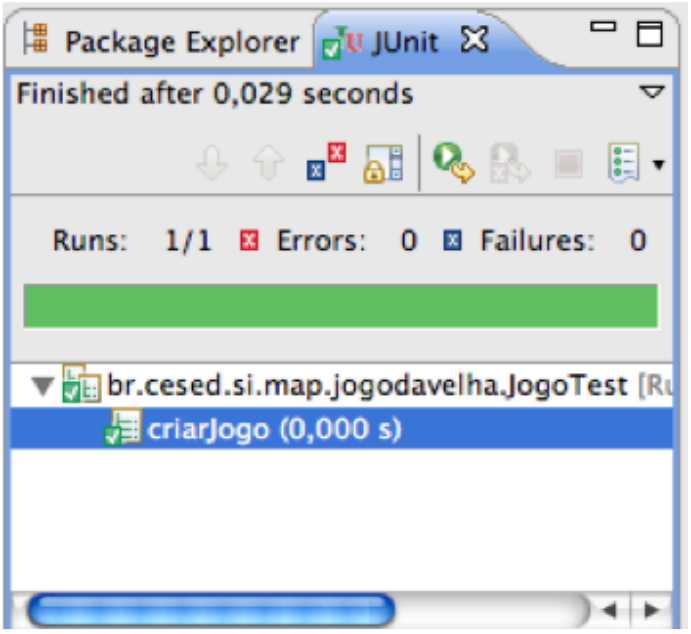
Outro teste forçará a evolução desse código

Executando o primeiro teste no Eclipse

Menu de contexto da classe de teste > Run As > 3 JUnit Test



O resultado é mostrado na *View* JUnit:



Completamos o primeiro ciclo:

LISTA DE TESTES v1
~~1. Criar um Jogo~~

Escrevendo mais testes

Qual seria a próxima funcionalidade a ser testada?

- Definir o primeiro jogador

LISTA DE TESTES v2
~~1. Criar um Jogo~~
2. Definir primeiro jogador
3. Definir primeiro jogador de novo
4. Definir primeiro jogador após início do jogo

Projetando a interface para a funcionalidade "Definir primeiro jogador"

Vamos criar um método para essa funcionalidade e passar a informação necessária por parâmetro.

Primeira opção:

- `jogo.setPrimeiroJogador("Rodrigo")`
  - Mas, o sistema precisa saber o **nome** do jogador?
  - O requisito diz que precisa saber a marca
  - Princípio YAGNI: You Aren't Going to Need It
  - Se não for precisar, não implemente

Segunda opção:

- `jogo.setMarcaPrimeiroJogador("X")`
  - Demandará casos de teste para caracteres estranhos
  - Existem apenas duas marcas possíveis, que podem ser representadas por uma variável booleana

Terceira opção:

- `jogo.setMarcaPrimeiroJogadorX(true)`

Como fazer as assertivas:

- `jogo.isMarcaPrimeiroJogadorX()`

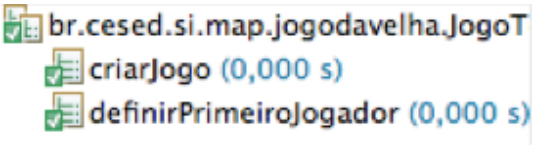
Definindo o teste 2

```
public class JogoTest {

  @Test
  public void definirPrimeiroJogador() {
    Jogo jogo = new Jogo();
    jogo.setMarcaPrimeiroJogadorX(true);
    Assert.assertTrue("Esperava que o primeiro jogador utilizasse X",
      jogo.isMarcaPrimeiroJogadorX());
  }
}
```

E implementando o código necessário para ele passe:

```
public class Jogo {

  public void setMarcaPrimeiroJogadorX(boolean b) {}

  public boolean isMarcaPrimeiroJogadorX() {
    return true;
  }
}
```

Rodando os testes:



Atualizando a lista de testes:

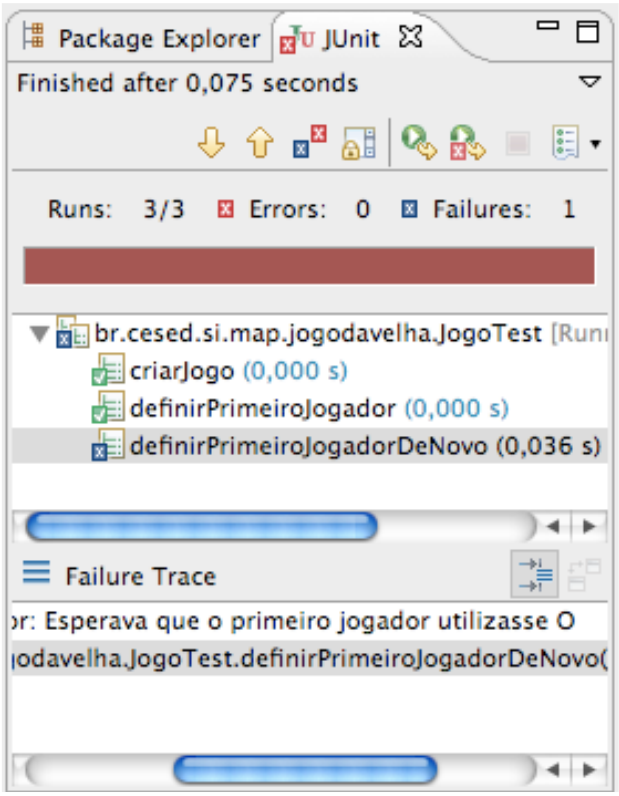
LISTA DE TESTES v2
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
3. Definir primeiro jogador de novo
4. Definir primeiro jogador após início do jogo

Teste: Definir primeiro jogador de novo

```
public class JogoTest {

  @Test
  public void definirPrimeiroJogadorDeNovo() {
    Jogo jogo = new Jogo();
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.setMarcaPrimeiroJogadorX(false);
    Assert.assertFalse("Esperava que o primeiro jogador utilizasse O",
      jogo.isMarcaPrimeiroJogadorX());
```

```
    }
}
```

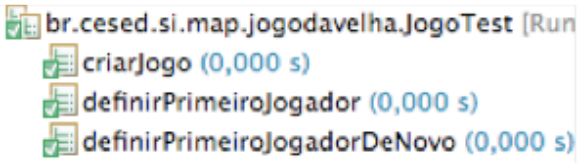
Se os testes forem executados nesse instante, não passarão mais, pois a implementação está defasada.



A solução mais simples é utilizar uma variável booleana para representar a marca do primeiro jogador.

```
public class Jogo {

  private boolean marcaPrimeiroJogadorX;

  public void setMarcaPrimeiroJogadorX(boolean b) {
    this.marcaPrimeiroJogadorX = b;
  }

  public boolean isMarcaPrimeiroJogadorX() {
    return marcaPrimeiroJogadorX;
  }
}
```

Agora os testes passam:



LISTA DE TESTES v2
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~

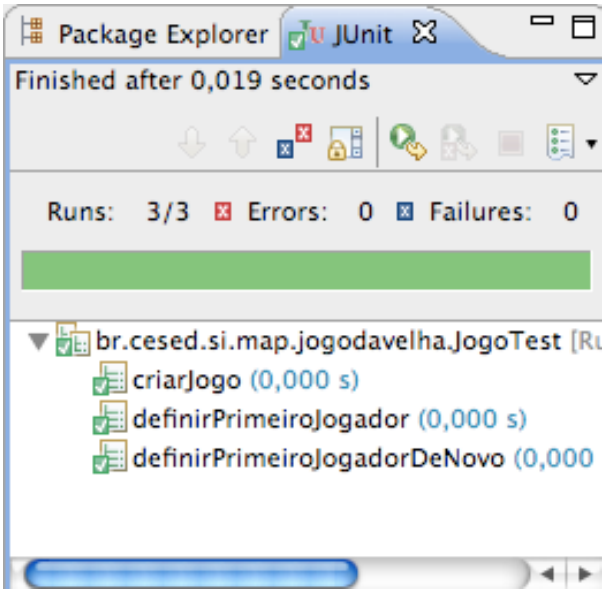
~~3. Definir primeiro jogador de novo~~
4. Definir primeiro jogador após início do jogo

Que tal um pouco de refatoramento agora?

- Princípio DRY: Don't Repeat Yourself
- Todos os testes começam com:
  - `Jogo jogo = new Jogo();`
- Podemos colocar esse código em um método só

```
public class JogoTest {

  private Jogo jogo;

  @Before //Executado antes de todos os testes
  public void iniciar() {
    jogo = new Jogo();
  }

  @Test
  public void criarJogo() {
    Assert.assertFalse("O jogo iniciou acabado", jogo.acabou());
  }

  @Test
  public void definirPrimeiroJogador() {
    jogo.setMarcaPrimeiroJogadorX(true);
    Assert.assertTrue("Esperava que o primeiro jogador utilizasse X",
      jogo.isMarcaPrimeiroJogadorX());
  }

  @Test
  public void definirPrimeiroJogadorDeNovo() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.setMarcaPrimeiroJogadorX(false);
    Assert.assertFalse("Esperava que o primeiro jogador utilizasse O",
      jogo.isMarcaPrimeiroJogadorX());
  }
}
```

Depois de refatorar, precisamos rodar os testes novamente, para garantir que não houve inclusão de bugs.



OS TESTES CONFEREM SEGURANÇA PARA A ATIVIDADE DE REFATORAMENTO.

Ainda não dá para fazer o teste 4

Não sabemos como se inicializa um jogo

Então, vamos projetar novos testes para a inicialização de um jogo.

O jogo começa quando se desenha a primeira marca. Portanto, faremos alguns testes para desenhar marcas:

LISTA DE TESTES v3
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
4. Definir primeiro jogador após início do jogo
5. Desenhar primeira marca
6. Desenhar marca em uma célula ocupada
7. Desenhar marca em uma coluna errada
8. Desenhar marca em uma linha errada

Teste 5: Desenhar primeira marca

Qual a interface para desenhar marcas?

O tabuleiro pode ser compreendido como uma matriz de 3 linhas e 3 colunas, ambas com a numeração começando em 0 e terminando em 2. Criaremos dois métodos, um para definir a marca e outro para verificar a marca atual em uma célula da matriz.

Dado que a marca do primeiro jogador é conhecida, não será preciso passar a informação da marca atual no método desenharMarca(). Além disso, após cada invocação desse método, a marca atual é alternada entre os jogadores.

O método mais simples, para verificação do valor de uma célula, poderia retorna verdadeiro para X e falso para O.

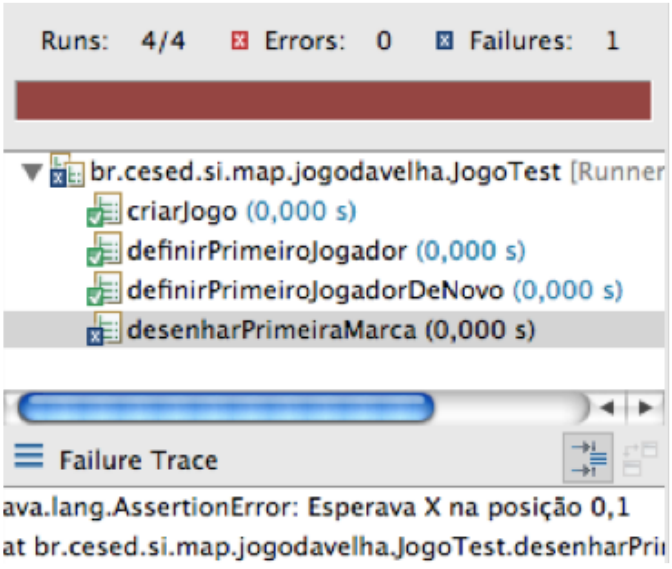
Portanto, o teste pode ser feito assim:

```
public class JogoTest {

  @Test
  public void desenharPrimeiraMarca() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.desenharMarca(0, 1);
    Assert.assertTrue("Esperava X na posica 0,1",
      jogo.isMarcaXNaPosicao(0, 1));
  }
}
```

E a implementação da interface apenas:

```
public class Jogo {

  public void desenharMarca(int linha, int coluna) {}

  public boolean isMarcaXNaPosicao(int linha, int coluna) {
    return false;
  }
}
```

Resultado, o teste não passa!



A implementação precisa prover o que foi acordado na interface:

- Um tabulerio no formato Matriz 3 X 3
- Conceito de marca atual, X ou O

O tabuleiro pode ser um array de Strings bidimensional.

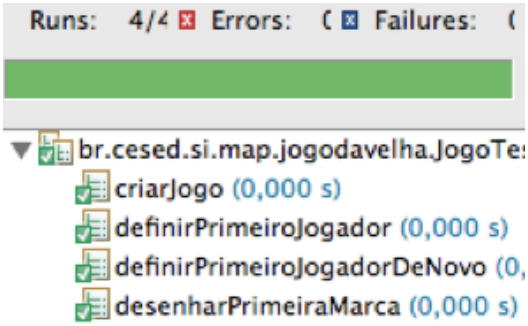
- **Porque um Array bidimensional de booleans não resolve?**

A marca atual pode ser um boolean. Já que teremos o conceito de marca atual, não precisaremos mais de uma variável para representar a marca do primeiro jogador. O método setMarcaPrimeiroJogador() trabalhará com a marca atual antes do jogo começar.

```
public class Jogo {

  private boolean proximaJogadaX;

  private String[][] tabuleiro = new String[][] {
    {null, null, null}, {null, null, null}, {null, null, null}
  };

  public void desenharMarca(int linha, int coluna) {
    String marca = (proximaJogadaX) ? "X" : "O";
    tabuleiro[linha][coluna] = marca;
  }

  public boolean isMarcaXNaPosicao(int linha, int coluna) {
    return tabuleiro[linha][coluna] == "X";
  }

  public void setMarcaPrimeiroJogadorX(boolean b) {
    this.proximaJogadaX = b;
  }

  public boolean isMarcaPrimeiroJogadorX() {
    return proximaJogadaX;
  }
}
```

É importante notar que a interface da classe Jogo não foi alterada. Apenas o detalhes internos de implementação foram modificados. Assim sendo, as classes que invocam Jogo (como JogoTest, por exemplo) não precisam ser alterados.

Agora, os testes voltaram a passar:



LISTA DE TESTES v3
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
4. Definir primeiro jogador após início do jogo

~~5. Desenhar primeira marca~~
6. Desenhar marca em uma célula ocupada
7. Desenhar marca em uma coluna errada
8. Desenhar marca em uma linha errada

Surgem novas idéias para testar ações inválidas!

LISTA DE TESTES v4
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
4. Definir primeiro jogador após início do jogo
~~5. Desenhar primeira marca~~
6. Desenhar marca em uma célula ocupada
7. Desenhar marca em uma coluna errada
8. Desenhar marca em uma linha errada
9. Desenhar marca antes de definir o primeiro jogador
10. Ler a marca de uma célula desocupada

Teste 6: Desenhar marca em uma célula ocupada

O método desenharMarca(int,int) permite que o cliente faça algumas invocações inválidas da classe Jogo. Portanto, essa classe precisa se proteger contra esses erros. Nesses casos, ela deve emitir erros compreensíveis, em vez de em vez de:

- Lançar exceções malucas, como ArrayIndexOutOfBoundsException ou NullPointerException;
- Ou simplesmente não emitir nenhuma notificação de erro.

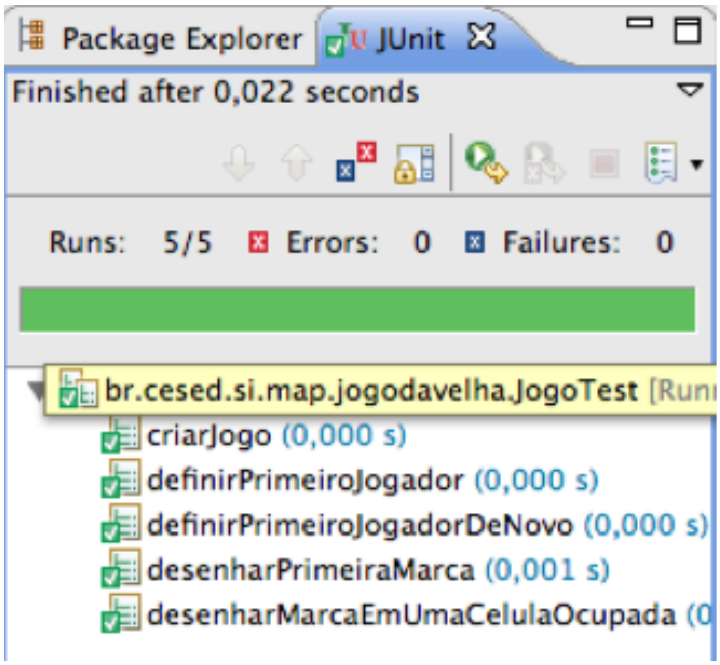
No exemplo a seguir, o teste diz que, quando houver sobreposição de uma célula já marcada, o jogo deve lançar uma ExcecaoJogoDaVelha. Obviamente, precisaremos criar essa classe de Exceção e ela fará parte da API do Jogo.

```
public class JogoTest {

  @Test(expected=ExcecaoJogoDaVelha.class)
  public void desenharMarcaEmUmaCelulaOcupada() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.desenharMarca(0, 1);
    jogo.desenharMarca(0, 1);
  }
}

public class ExcecaoJogoDaVelha extends RuntimeException {

  public ExcecaoJogoDaVelha(String msg) {
```

```
    super(msg);
  }
}
```

Adicionando o código de validação no método desenharMarca(int,int). Apenas o que é necessário para o teste passar.

```
public class Jogo {

  public void desenharMarca(int linha, int coluna) {
    if (tabuleiro[linha][coluna] != null) {
      throw new ExcecaoJogoDaVelha("Célula ocupada");
    }

    String marca = (proximaJogadaX) ? "X" : "O";
    tabuleiro[linha][coluna] = marca;
  }
}
```



LISTA DE TESTES v4
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
4. Definir primeiro jogador após início do jogo
~~5. Desenhar primeira marca~~
~~6. Desenhar marca em uma célula ocupada~~
7. Desenhar marca em uma coluna errada
8. Desenhar marca em uma linha errada

9. Desenhar marca antes de definir o primeiro jogador
10. Ler a marca de uma célula desocupada

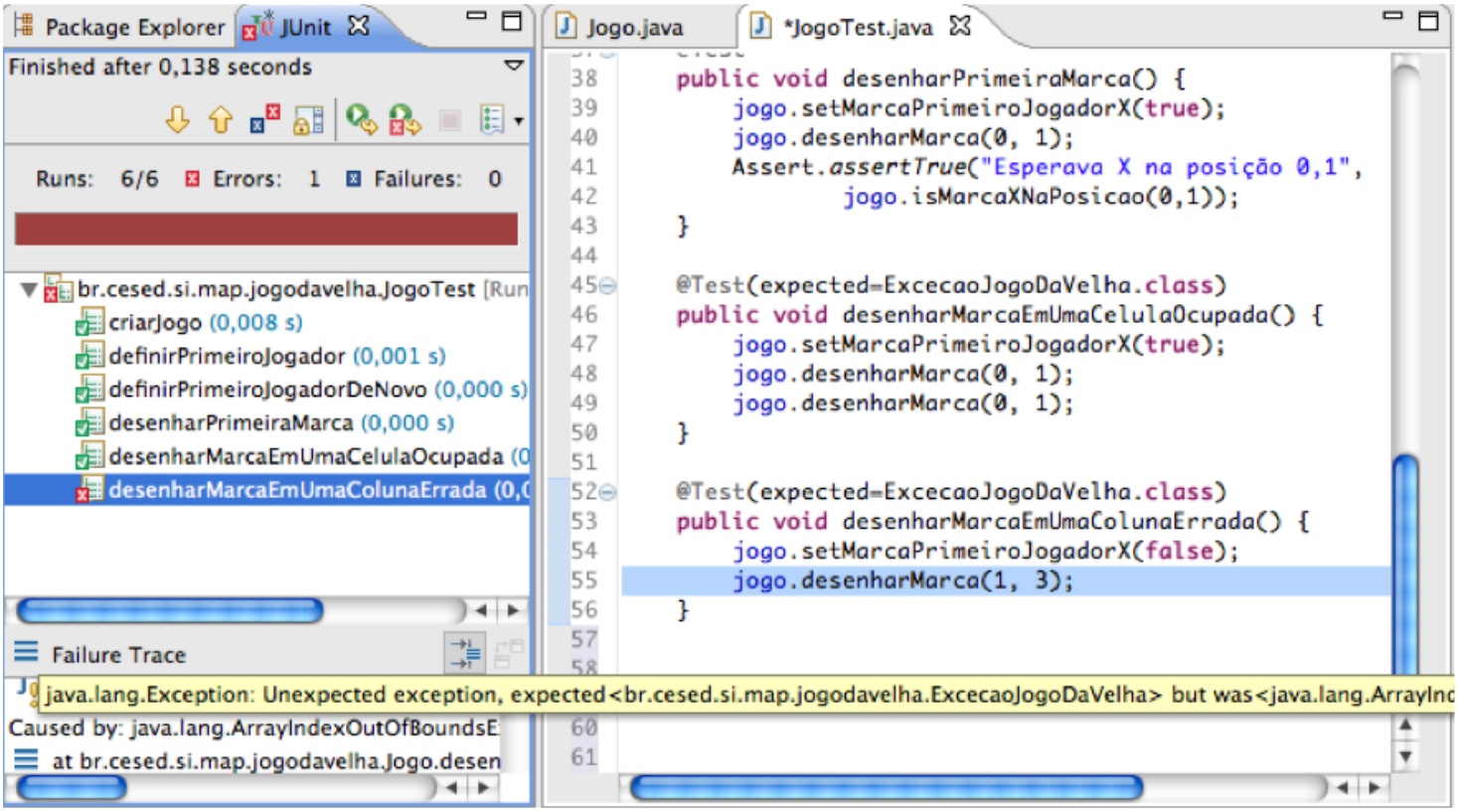
Agora testaremos os limites dos parâmetros do método desenharMarca(int,int)

Teste 7: Desenhar marca em uma coluna errada

Teste 8: Desenhar marca em uma linha errada

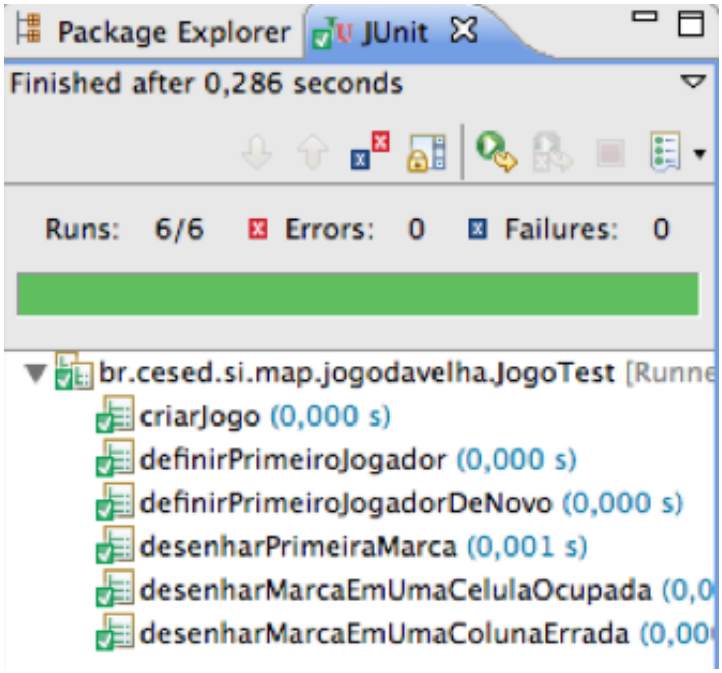
```
public class JogoTest {

  @Test(expected=ExcecaoJogoDaVelha.class)
  public void desenharMarcaEmUmaColunaErrada() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.desenharMarca(1, 3);
  }

  @Test(expected=ExcecaoJogoDaVelha.class)
  public void desenharMarcaEmUmaLinhaErrada() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.desenharMarca(-1, 1);
  }
}
```

Antes de implementar a validação necessária, o teste falha pois a classe Jogo lança uma exceção diferente da esperada:



A seguir, a implementação das validações. A validação de limites deve vir antes da validação de célula ocupada, se não a exceção errada continuará sendo lançada.

```
public class Jogo {

  public void desenharMarca(int linha, int coluna) {
    if (linha < 0 || linha > 2) {
      throw new ExcecaoJogoDaVelha("Linha fora da faixa permitida");
    }

    if (coluna < 0 || coluna > 2) {
      throw new ExcecaoJogoDaVelha("Coluna fora da faixa permitida");
    }

    if (tabuleiro[linha][coluna] != null) {
      throw new ExcecaoJogoDaVelha("Célula ocupada");
    }

    String marca = (proximaJogadaX) ? "X" : "O";
    tabuleiro[linha][coluna] = marca;
  }
}
```

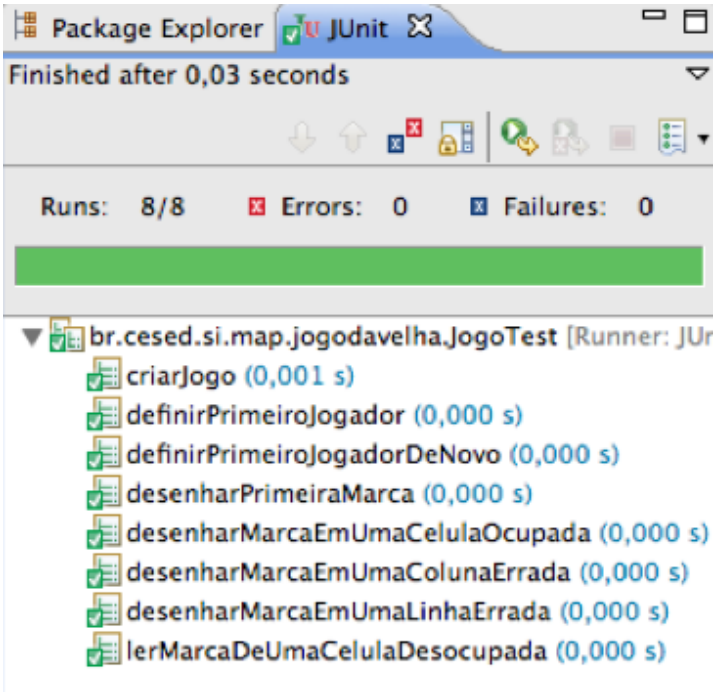


LISTA DE TESTES v4
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
4. Definir primeiro jogador após início do jogo
~~5. Desenhar primeira marca~~
~~6. Desenhar marca em uma célula ocupada~~
~~7. Desenhar marca em uma coluna errada~~
~~8. Desenhar marca em uma linha errada~~
9. Desenhar marca antes de definir o primeiro jogador
10. Ler a marca de uma célula desocupada

Teste 10: Ler a marca de uma célula desocupada

```
public class JogoTest {

  @Test(expected=ExcecaoJogoDaVelha.class)
  public void lerMarcaDeUmaCelulaDesocupada() {
    jogo.setMarcaPrimeiroJogadorX(false);
    jogo.isMarcaXNaPosicao(0, 1);
  }
}
```

```
public class Jogo {

  public boolean isMarcaXNaPosicao(int linha, int coluna) {
    String celula = tabuleiro[linha][coluna];

    if (celula == null) {
      throw new ExcecaoJogoDaVelha("Célula vazia");
    }

    return celula == "X";
  }
}
```



LISTA DE TESTES v4
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
4. Definir primeiro jogador após início do jogo
~~5. Desenhar primeira marca~~
~~6. Desenhar marca em uma célula ocupada~~
~~7. Desenhar marca em uma coluna errada~~
~~8. Desenhar marca em uma linha errada~~
9. Desenhar marca antes de definir o primeiro jogador
~~10. Ler a marca de uma célula desocupada~~

Ao testar esse novo método [isMarcaXNaPosicao(int,int)], podemos imitar os testes do método desenharMarca(int,int). Logo, a lista de testes crescerá:

LISTA DE TESTES v5
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
4. Definir primeiro jogador após início do jogo
~~5. Desenhar primeira marca~~
~~6. Desenhar marca em uma célula ocupada~~
~~7. Desenhar marca em uma coluna errada~~
~~8. Desenhar marca em uma linha errada~~
9. Desenhar marca antes de definir o primeiro jogador
~~10. Ler a marca de uma célula desocupada~~
11. Ler a marca em uma coluna errada
12. Ler a marca em uma linha errada

Teste 11: Ler a marca em uma coluna errada

Teste 12: Ler a marca em uma linha errada

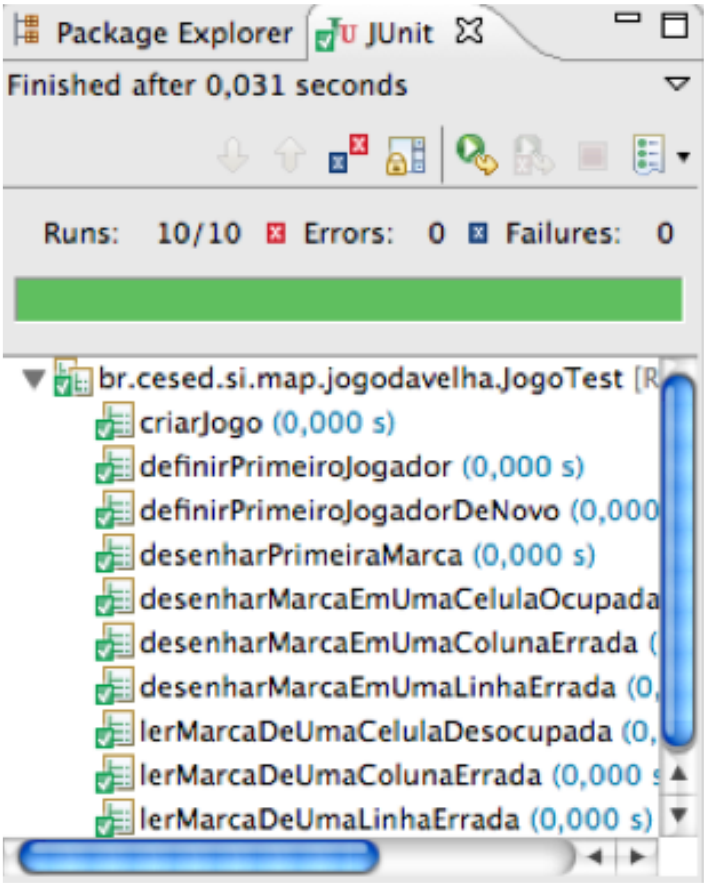
```
public class JogoTest {

  @Test(expected=ExcecaoJogoDaVelha.class)
  public void lerMarcaDeUmaColunaErrada() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.isMarcaXNaPosicao(1, 3);
  }

  @Test(expected=ExcecaoJogoDaVelha.class)
  public void lerMarcaDeUmaLinhaErrada() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.isMarcaXNaPosicao(-1, 1);
  }
}
```

A implementação requerida por estes testes é bem semelhante ao código que foi utilizado no método desenharMarca(int,int).

```
public class Jogo {

  public boolean isMarcaXNaPosicao(int linha, int coluna) {
    if (linha < 0 || linha > 2) {
      throw new ExcecaoJogoDaVelha("Linha fora da faixa permitida");
```

```
    }

    if (coluna < 0 || coluna > 2) {
      throw new ExcecaoJogoDaVelha("Coluna fora da faixa permitida");
    }

    String celula = tabuleiro[linha][coluna];

    if (celula == null) {
      throw new ExcecaoJogoDaVelha("Célula vazia");
    }

    return celula == "X";
  }
}
```

Resultado:

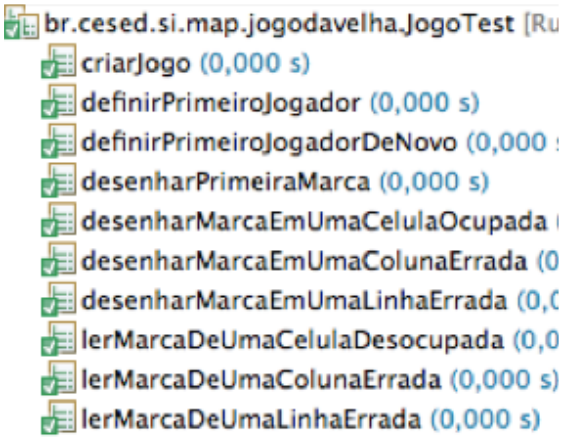


Devido ao código replicado, somos motivados a fazer o seguinte refatoramento:

```
public class Jogo {

  public void desenharMarca(int linha, int coluna) {
    verificarLimites(linha, coluna);

    if (tabuleiro[linha][coluna] != null) {
      throw new ExcecaoJogoDaVelha("Célula ocupada");
    }
```

```
    String marca = (proximaJogadaX) ? "X" : "O";
    tabuleiro[linha][coluna] = marca;
  }

  public boolean isMarcaXNaPosicao(int linha, int coluna) {
    verificarLimites(linha, coluna);
    String celula = tabuleiro[linha][coluna];

    if (celula == null) {
      throw new ExcecaoJogoDaVelha("Célula vazia");
    }

    return celula == "X";
  }

  private void verificarLimites(int linha, int coluna) {
    if (linha < 0 || linha > 2) {
      throw new ExcecaoJogoDaVelha("Linha fora da faixa permitida");
    }

    if (coluna < 0 || coluna > 2) {
      throw new ExcecaoJogoDaVelha("Coluna fora da faixa permitida");
    }
  }
}
```

Verificando se o refatoramento não quebrou os testes:



LISTA DE TESTES v5
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
4. Definir primeiro jogador após início do jogo
~~5. Desenhar primeira marca~~
~~6. Desenhar marca em uma célula ocupada~~
~~7. Desenhar marca em uma coluna errada~~

~~8. Desenhar marca em uma linha errada~~
9. Desenhar marca antes de definir o primeiro jogador
~~10. Ler a marca de uma célula desocupada~~
~~11. Ler a marca em uma coluna errada~~
~~12. Ler a marca em uma linha errada~~

Agora já sabemos como iniciar um jogo, portanto podemos voltar a um dos testes iniciais.

Teste 4: Definir primeiro jogador após início do jogo

Teste 9: Desenhar marca antes de definir o primeiro jogador

Após iniciar o jogo, o primeiro jogador não pode ser mais definido. Semelhantemente, o jogo não pode ser iniciado sem que antes se defina o primeiro jogador. Assim sendo, essas duas ações devem lançar exceção.

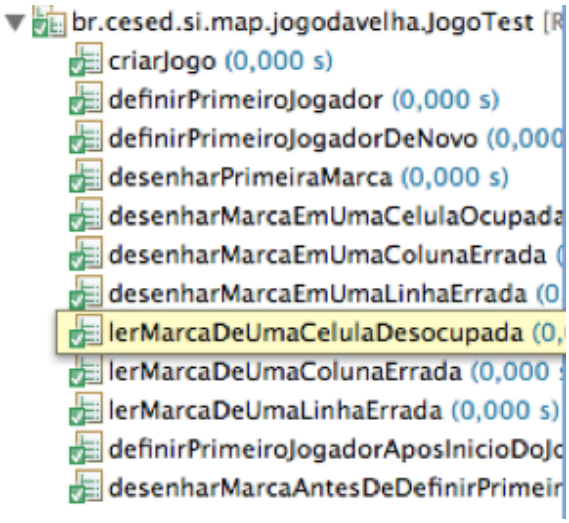
```
public class JogoTest {

  @Test(expected=ExcecaoJogoDaVelha.class)
  public void definirPrimeiroJogadorAposInicioDoJogo() {
    jogo.setMarcaPrimeiroJogadorX(false);
    jogo.desenharMarca(0, 1);
    jogo.setMarcaPrimeiroJogadorX(true);
  }

  @Test(expected=ExcecaoJogoDaVelha.class)
  public void desenharMarcaAntesDeDefinirPrimeiroJogador() {
        jogo.desenharMarca(0, 1);
  }
}
```

A seguir, veja a implementação para esses dois testes:

```
public class Jogo {

  private boolean iniciouJogo = false;
  private boolean primeiroJogadorDefinido = false;

  public void setMarcaPrimeiroJogadorX(boolean b) {
    if (iniciouJogo) {
      throw new ExcecaoJogoDaVelha("O jogo já foi iniciado");
    }
    proximaJogadaX = b;
    primeiroJogadorDefinido = true;
  }

  public void desenharMarca(int linha, int coluna) {
    if (!primeiroJogadorDefinido) {
      throw new ExcecaoJogoDaVelha("O primeiro jogador ainda não foi "  +
```

```
            "definido");
        }

        verificarLimites(linha, coluna);
        if (tabuleiro[linha][coluna] != null) {
          throw new ExcecaoJogoDaVelha("Célula ocupada");
        }

        String marca = (proximaJogadaX) ? "X" : "O";
        tabuleiro[linha][coluna] = marca;
        inicioJogo = true;
    }
}
```

Todos os testes estão passando:



LISTA DE TESTES v5
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
~~4. Definir primeiro jogador após início do jogo~~
~~5. Desenhar primeira marca~~
~~6. Desenhar marca em uma célula ocupada~~
~~7. Desenhar marca em uma coluna errada~~
~~8. Desenhar marca em uma linha errada~~
~~9. Desenhar marca antes de definir o primeiro jogador~~
~~10. Ler a marca de uma célula desocupada~~
~~11. Ler a marca em uma coluna errada~~
~~12. Ler a marca em uma linha errada~~

Concluímos os testes de criação e inicialização de um jogo, o que poderia ser uma primeira *release* do sistema para o cliente. Podemos prosseguir com mais funcionalidades de um jogo: desenhar mais marcas e ganhar jogo. Essas novas funcionalidades estão especificadas através de novos testes:

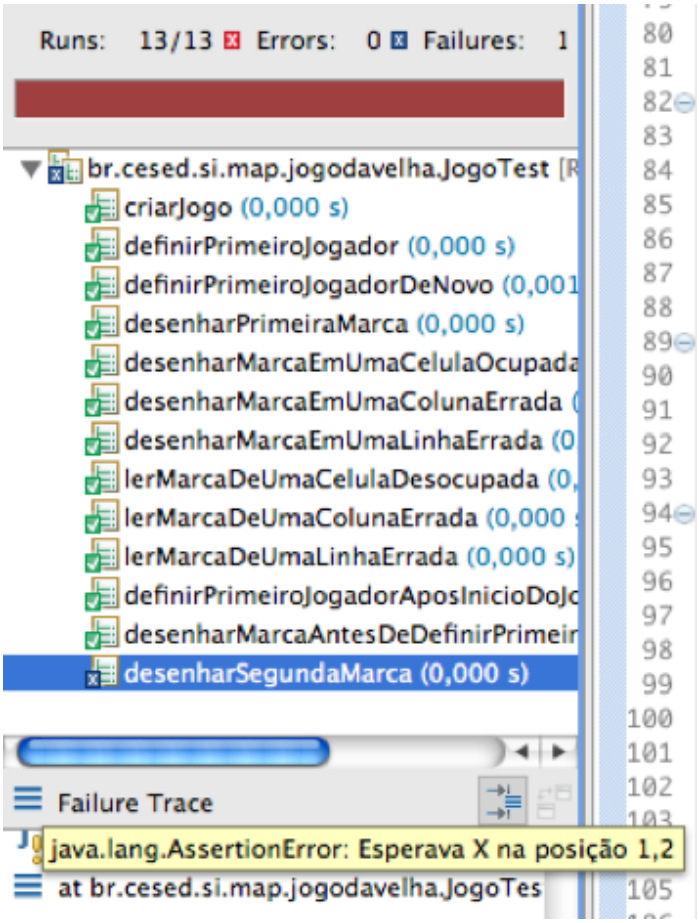
LISTA DE TESTES v6
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
~~4. Definir primeiro jogador após início do jogo~~
~~5. Desenhar primeira marca~~
~~6. Desenhar marca em uma célula ocupada~~
~~7. Desenhar marca em uma coluna errada~~
~~8. Desenhar marca em uma linha errada~~
~~9. Desenhar marca antes de definir o primeiro jogador~~
~~10. Ler a marca de uma célula desocupada~~
~~11. Ler a marca em uma coluna errada~~
~~12. Ler a marca em uma linha errada~~
13. Desenhar segunda marca
14. Jogo ganho através de coluna
15. Jogo ganho através de linha
16. Jogo ganho através de diagonal
17. Desenhar marca após jogo ganho

Teste 13: Desenhar segunda marca

Esse teste é o que define a alternância entre os símbolos dos jogadores.

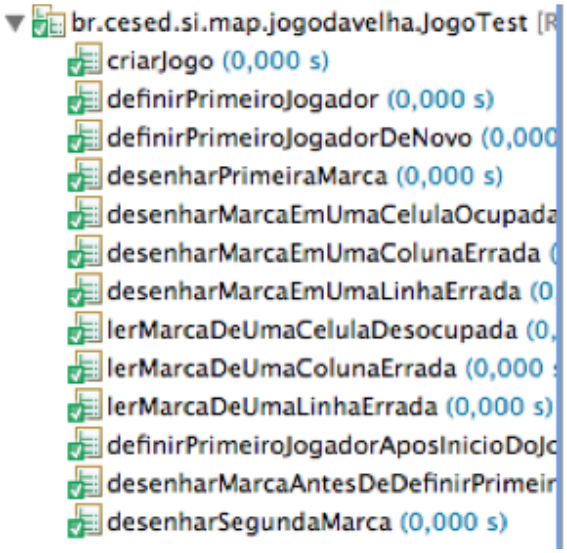
```
public class JogoTest {

  @Test
  public void desenharSegundaMarca() {
    jogo.setMarcaPrimeiroJogadorX(false);
    jogo.desenharMarca(0, 1);
    Assert.assertFalse("Esperava O na posição 0,1",
      jogo.isMarcaXNaPosicao(0,1));
    jogo.desenharMarca(1, 2);
    Assert.assertTrue("Esperava X na posição 1,2",
      jogo.isMarcaXNaPosicao(1,2));
  }
}
```

Como essa funcionalidade não foi implementada, o teste quebra:



Basta adicionar uma linha de código para que o teste passe:

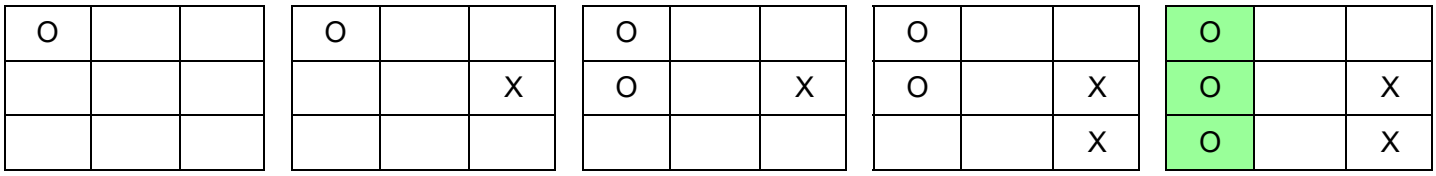
```
public class Jogo {

  public void desenharMarca(int linha, int coluna) {
    if (!primeiroJogadorDefinido) {
      throw new ExcecaoJogoDaVelha("O primeiro jogador ainda não foi "  +
        "definido");
    }

    verificarLimites(linha, coluna);
    if (tabuleiro[linha][coluna] != null) {
      throw new ExcecaoJogoDaVelha("Célula ocupada");
    }

    String marca = (proximaJogadaX) ? "X" : "O";
    tabuleiro[linha][coluna] = marca;
    iniciouJogo = true;
    proximaJogadaX = !proximaJogadaX;
  }
}
```



LISTA DE TESTES v6
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
~~4. Definir primeiro jogador após início do jogo~~
~~5. Desenhar primeira marca~~
~~6. Desenhar marca em uma célula ocupada~~
~~7. Desenhar marca em uma coluna errada~~
~~8. Desenhar marca em uma linha errada~~
~~9. Desenhar marca antes de definir o primeiro jogador~~
~~10. Ler a marca de uma célula desocupada~~
~~11. Ler a marca em uma coluna errada~~
~~12. Ler a marca em uma linha errada~~
~~13. Desenhar segunda marca~~
14. Jogo ganho através de coluna
15. Jogo ganho através de linha
16. Jogo ganho através de diagonal
17. Desenhar marca após jogo ganho

Teste 14: Jogo ganho através de coluna

Para testar um jogo ganho através de coluna, vamos esboçar a sequência de passos de um jogo que termina com essa configuração:



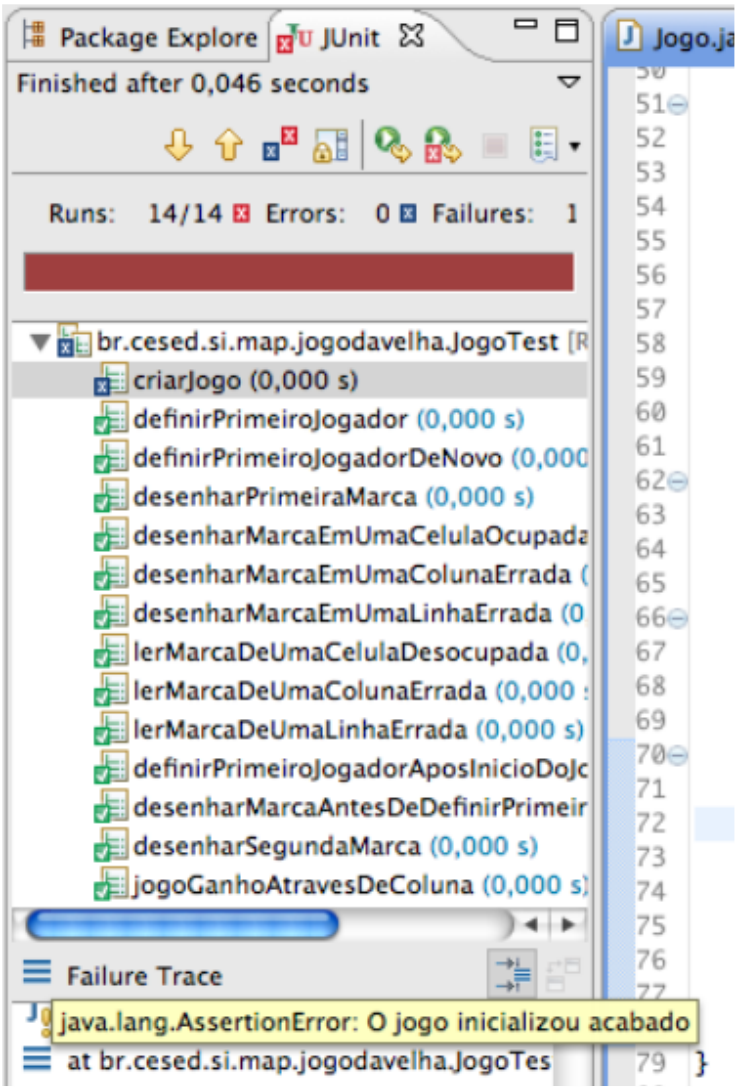
Vamos escrever um teste que reproduza esse cenário. Daí, o jogo deve informar que acabou:

```
public class JogoTest {

  @Test
  public void jogoGanhoAtravesDeColuna() {
    jogo.setMarcaPrimeiroJogadorX(false);
    jogo.desenharMarca(0, 0);
    jogo.desenharMarca(1, 2);
    jogo.desenharMarca(1, 0);
    jogo.desenharMarca(2, 2);
    jogo.desenharMarca(2, 0);
    Assert.assertTrue("Esperava que o jogo tivesse acabado",
      jogo.acabou());
  }
}
```

Na implementação, verificaremos se alguma das colunas está preenchida com símbolos iguais:

```
public class Jogo {

  public void acabou() {
    return verificarColunas();
  }

  private boolean verificarColunas() {
    for (int i = 0; i < 3; i++) {
      if (tabuleiro[0][i] == tabuleiro[1][i] &&
          tabuleiro[1][i] == tabuleiro[2][i]) {
        return true;
      }
    }
    return false;
  }
}
```

Concluída a implementação, agora vamos rodar os testes:

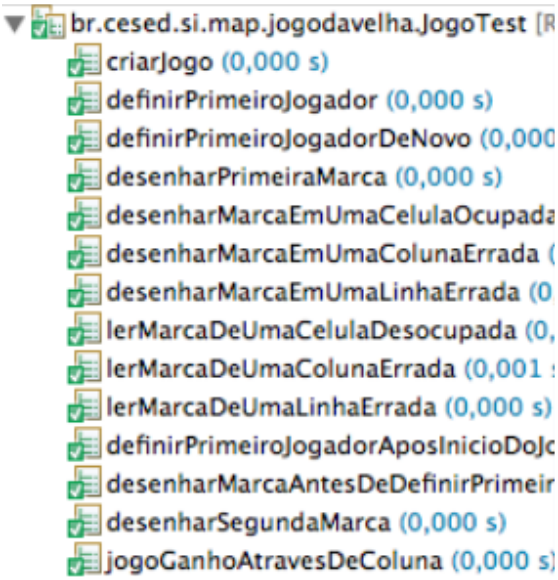


Pegamos um bug! A implementação satisfez o teste jogoGanhoAtravesDeColuna(), porém quebrou o primeiro teste, criarJogo(), pois agora o jogo já se inicia acabado.

Isso ocorreu porque a primeira coluna está toda preenchida com símbolos iguais (null).

Correção do bug:

```
public class Jogo {

  private boolean verificarColunas() {
    for (int i = 0; i < 3; i++) {
      if (tabuleiro[0][i] == tabuleiro[1][i] &&
          tabuleiro[1][i] == tabuleiro[2][i] && tabuleiro[2][i] != null)
{
        return true;
      }
    }
    return false;
  }
}
```

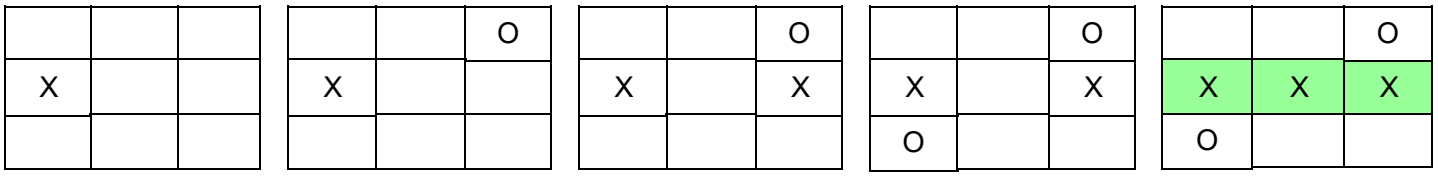


Agora podemos marcar o teste como concluído:

LISTA DE TESTES v6
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
~~4. Definir primeiro jogador após início do jogo~~
~~5. Desenhar primeira marca~~
~~6. Desenhar marca em uma célula ocupada~~
~~7. Desenhar marca em uma coluna errada~~
~~8. Desenhar marca em uma linha errada~~
~~9. Desenhar marca antes de definir o primeiro jogador~~
~~10. Ler a marca de uma célula desocupada~~
~~11. Ler a marca em uma coluna errada~~
~~12. Ler a marca em uma linha errada~~
~~13. Desenhar segunda marca~~
~~14. Jogo ganho através de coluna~~
15. Jogo ganho através de linha
16. Jogo ganho através de diagonal
17. Desenhar marca após jogo ganho

Teste 15: Jogo ganho através de linha

Para testar um jogo ganho através de linha, vamos esboçar a sequência de passos de um jogo que termina com essa configuração:

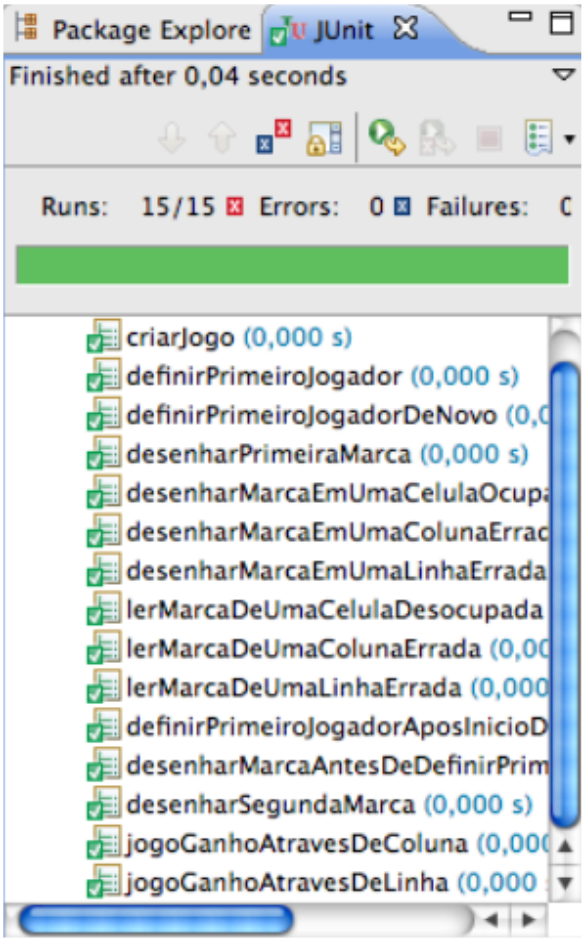


Vamos escrever um teste que reproduza esse cenário. Daí, o jogo deve informar que acabou:

```
public class JogoTest {

  @Test
  public void jogoGanhoAtravesDeLinha() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.desenharMarca(1, 0);
    jogo.desenharMarca(0, 2);
    jogo.desenharMarca(1, 2);
    jogo.desenharMarca(2, 0);
    jogo.desenharMarca(1, 1);
    Assert.assertTrue("Esperava que o jogo tivesse acabado",
      jogo.acabou());
  }
}
```

Na implementação, verificaremos se alguma das linhas está preenchida com símbolos iguais:

```
public class Jogo {

  public void acabou() {
    return verificarColunas() || verificarLinhas();
  }


  private boolean verificarLinhas() {
    for (int i = 0; i < 3; i++) {
      if (tabuleiro[i][0] == tabuleiro[i][1] &&
          tabuleiro[i][1] == tabuleiro[i][2] && tabuleiro[i][2] != null)
{
        return true;
      }
    }
    return false;
  }
}
```



Teste 16: Jogo ganho através de diagonal

Para testar um jogo ganho através de diagonal, vamos esboçar a sequência de passos de um jogo que termina com essa configuração:

		X	O		X	O		X	O		X	O		X
							X			X			X	
											O	X		O

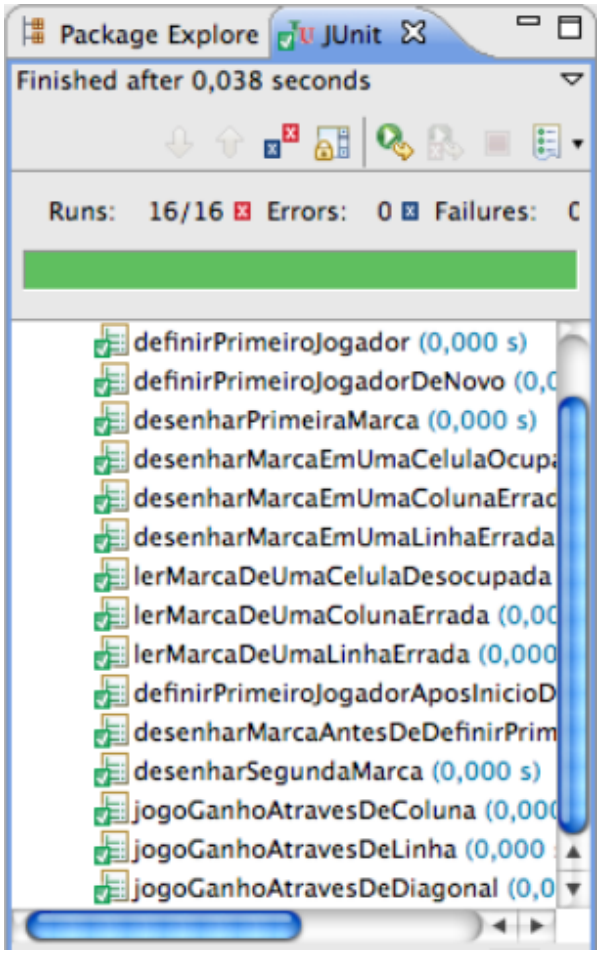
Vamos escrever um teste que reproduza esse cenário. Daí, o jogo deve informar que acabou:

```
public class JogoTest {

  @Test
  public void jogoGanhoAtravesDeDiagonal() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.desenharMarca(0, 2);
    jogo.desenharMarca(0, 0);
    jogo.desenharMarca(1, 1);
    jogo.desenharMarca(2, 2);
    jogo.desenharMarca(2, 0);
```

```
    Assert.assertTrue("Esperava que o jogo tivesse acabado",
      jogo.acabou());
  }
}
```

Na implementação, verificaremos se alguma das duas diagonais está preenchida com símbolos iguais:

```
public class Jogo {

  public void acabou() {
    return verificarColunas() || verificarLinhas() ||
      verificarDiagonais();
  }


  private boolean verificarDiagonais() {
    if (tabuleiro[0][0] == tabuleiro[1][1] &&
        tabuleiro[1][1] == tabuleiro[2][2] && tabuleiro[2][2] != null) {
      return true;
    }

    if (tabuleiro[0][2] == tabuleiro[1][1] &&
        tabuleiro[1][1] == tabuleiro[2][0] && tabuleiro[2][0] != null) {
      return true;
    }

    return false;
  }
}
```



LISTA DE TESTES v6
~~1. Criar um Jogo~~
~~2. Definir primeiro jogador~~
~~3. Definir primeiro jogador de novo~~
~~4. Definir primeiro jogador após início do jogo~~
~~5. Desenhar primeira marca~~
~~6. Desenhar marca em uma célula ocupada~~
~~7. Desenhar marca em uma coluna errada~~
~~8. Desenhar marca em uma linha errada~~
~~9. Desenhar marca antes de definir o primeiro jogador~~
~~10. Ler a marca de uma célula desocupada~~
~~11. Ler a marca em uma coluna errada~~
~~12. Ler a marca em uma linha errada~~
~~13. Desenhar segunda marca~~
~~14. Jogo ganho através de coluna~~
~~15. Jogo ganho através de linha~~

~~16. Jogo ganho através de diagonal~~
17. Desenhar marca após jogo ganho

Falta apenas um teste!

Teste 17: Desenhar marca após jogo ganho

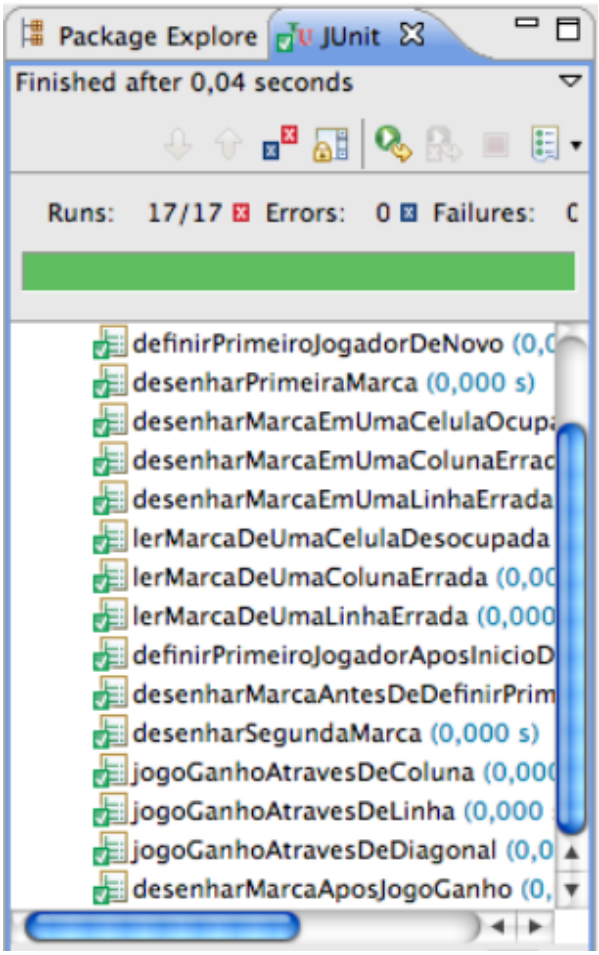
Vamos aproveitar o cenário do último teste, para forçar o uso do jogo após a sua conclusão. Obviamente, uma exceção deve ser lançada.

```
public class JogoTest {

  @Test(expected=ExcecaoJogoDaVelha.class)
  public void desenharMarcaAposJogoGanho() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.desenharMarca(0, 2);
    jogo.desenharMarca(0, 0);
    jogo.desenharMarca(1, 1);
    jogo.desenharMarca(2, 2);
    jogo.desenharMarca(2, 0);
    jogo.desenharMarca(2, 1);
  }
}
```

Para passar no teste, a classe Jogo precisa apenas verificar se o jogo acabou, antes de desenhar uma marca.

```
public class Jogo {

  public void desenharMarca(int linha, int coluna) {
    if (acabou()) {
      throw new ExcecaoJogoDaVelha("O jogo já acabou");
    }

    if (!primeiroJogadorDefinido) {
      throw new ExcecaoJogoDaVelha("O primeiro jogador ainda não foi "  +
        "definido");
    }

    verificarLimites(linha, coluna);
    if (tabuleiro[linha][coluna] != null) {
      throw new ExcecaoJogoDaVelha("Célula ocupada");
    }

    String marca = (proximaJogadaX) ? "X" : "O";
    tabuleiro[linha][coluna] = marca;
    iniciouJogo = true;
    proximaJogadaX = !proximaJogadaX;
  }
}
```

Esta barra verde atesta que a implementação da lógica de Jogo da Velha está concluída e validada:



O teste e a implementação ficaram assim:

```
public class JogoTest {


  private Jogo jogo;


  @Before //Executado antes de todos os testes
  public void iniciar() {
    jogo = new Jogo();
  }

  @Test
  public void criarJogo() {
    Assert.assertFalse("O jogo iniciou acabado", jogo.acabou());
  }

  @Test
  public void definirPrimeiroJogador() {
    jogo.setMarcaPrimeiroJogadorX(true);
    Assert.assertTrue("Esperava que o primeiro jogador utilizasse X",
      jogo.isMarcaPrimeiroJogadorX());
  }

  @Test
  public void definirPrimeiroJogadorDeNovo() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.setMarcaPrimeiroJogadorX(false);
    Assert.assertFalse("Esperava que o primeiro jogador utilizasse O",
```

```
    jogo.isMarcaPrimeiroJogadorX());
}

@Test
public void desenharPrimeiraMarca() {
  jogo.setMarcaPrimeiroJogadorX(true);
  jogo.desenharMarca(0, 1);
  Assert.assertTrue("Esperava X na posica 0,1",
    jogo.isMarcaXNaPosicao());
}

@Test(expected=ExcecaoJogoDaVelha.class)
public void desenharMarcaEmUmaCelulaOcupada() {
  jogo.setMarcaPrimeiroJogadorX(true);
  jogo.desenharMarca(0, 1);
  jogo.desenharMarca(0, 1);
}

@Test(expected=ExcecaoJogoDaVelha.class)
public void desenharMarcaEmUmaColunaErrada() {
  jogo.setMarcaPrimeiroJogadorX(true);
  jogo.desenharMarca(1, 3);
}

@Test(expected=ExcecaoJogoDaVelha.class)
public void desenharMarcaEmUmaLinhaErrada() {
  jogo.setMarcaPrimeiroJogadorX(true);
  jogo.desenharMarca(-1, 1);
}

@Test(expected=ExcecaoJogoDaVelha.class)
public void lerMarcaDeUmaCelulaDesocupada() {
  jogo.setMarcaPrimeiroJogadorX(false);
  jogo.isMarcaXNaPosicao(0, 1);
}

@Test(expected=ExcecaoJogoDaVelha.class)
public void lerMarcaDeUmaColunaErrada() {
  jogo.setMarcaPrimeiroJogadorX(true);
  jogo.isMarcaXNaPosicao(1, 3);
}

@Test(expected=ExcecaoJogoDaVelha.class)
public void lerMarcaDeUmaLinhaErrada() {
  jogo.setMarcaPrimeiroJogadorX(true);
  jogo.isMarcaXNaPosicao(-1, 1);
}


@Test(expected=ExcecaoJogoDaVelha.class)
public void definirPrimeiroJogadorAposInicioDoJogo() {
  jogo.setMarcaPrimeiroJogadorX(false);
  jogo.desenharMarca(0, 1);
  jogo.setMarcaPrimeiroJogadorX(true);
}

@Test(expected=ExcecaoJogoDaVelha.class)
public void desenharMarcaAntesDeDefinirPrimeiroJogador() {
      jogo.desenharMarca(0, 1);
```

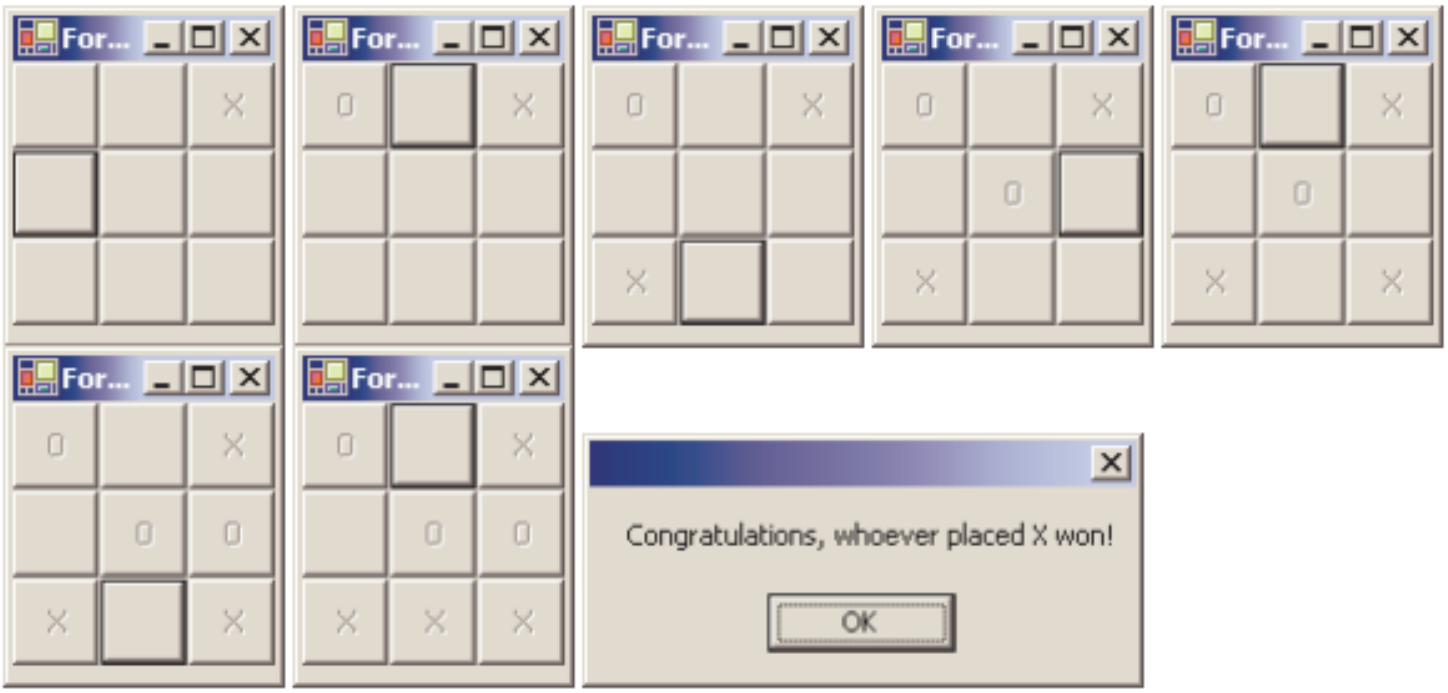
```
  }

  @Test
  public void desenharSegundaMarca() {
    jogo.setMarcaPrimeiroJogadorX(false);
    jogo.desenharMarca(0, 1);
    Assert.assertFalse("Esperava O na posição 0,1",
      jogo.isMarcaXNaPosicao(0,1));
    jogo.desenharMarca(1, 2);
    Assert.assertTrue("Esperava X na posição 1,2",
      jogo.isMarcaXNaPosicao(1,2));
  }

  @Test
  public void jogoGanhoAtravesDeColuna() {
    jogo.setMarcaPrimeiroJogadorX(false);
    jogo.desenharMarca(0, 0);
    jogo.desenharMarca(1, 2);
    jogo.desenharMarca(1, 0);
    jogo.desenharMarca(2, 2);
    jogo.desenharMarca(2, 0);
    Assert.assertTrue("Esperava que o jogo tivesse acabado",
      jogo.acabou());
  }

  @Test
  public void jogoGanhoAtravesDeLinha() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.desenharMarca(1, 0);
    jogo.desenharMarca(0, 2);
    jogo.desenharMarca(1, 2);
    jogo.desenharMarca(2, 0);
    jogo.desenharMarca(1, 1);
    Assert.assertTrue("Esperava que o jogo tivesse acabado",
      jogo.acabou());
  }

  @Test
  public void jogoGanhoAtravesDeDiagonal() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.desenharMarca(0, 2);
    jogo.desenharMarca(0, 0);
    jogo.desenharMarca(1, 1);
    jogo.desenharMarca(2, 2);
    jogo.desenharMarca(2, 0);
    Assert.assertTrue("Esperava que o jogo tivesse acabado",
      jogo.acabou());
  }

  @Test(expected=ExcecaoJogoDaVelha.class)
  public void desenharMarcaAposJogoGanho() {
    jogo.setMarcaPrimeiroJogadorX(true);
    jogo.desenharMarca(0, 2);
    jogo.desenharMarca(0, 0);
    jogo.desenharMarca(1, 1);
    jogo.desenharMarca(2, 2);
```

```
    jogo.desenharMarca(2, 0);
    jogo.desenharMarca(2, 1);
  }
}


public class Jogo {


  private boolean proximaJogadaX;
  private boolean iniciouJogo = false;
  private boolean primeiroJogadorDefinido = false;

  private String[][] tabuleiro = new String[][] {
    {null, null, null}, {null, null, null}, {null, null, null}
  };


  public void acabou() {
    return verificarColunas() || verificarLinhas() ||
      verificarDiagonais();
  }

  public void desenharMarca(int linha, int coluna) {
    if (acabou()) {
      throw new ExcecaoJogoDaVelha("O jogo já acabou");
    }

    if (!primeiroJogadorDefinido) {
      throw new ExcecaoJogoDaVelha("O primeiro jogador ainda não foi "  +
        "definido");
    }

    verificarLimites(linha, coluna);
    if (tabuleiro[linha][coluna] != null) {
      throw new ExcecaoJogoDaVelha("Célula ocupada");
    }

    String marca = (proximaJogadaX) ? "X" : "O";
    tabuleiro[linha][coluna] = marca;
    iniciouJogo = true;
    proximaJogadaX = !proximaJogadaX;
  }

  public boolean isMarcaPrimeiroJogadorX() {
    return proximaJogadaX;
  }

  public boolean isMarcaXNaPosicao(int linha, int coluna) {
    verificarLimites(linha, coluna);
    String celula = tabuleiro[linha][coluna];

    if (celula == null) {
      throw new ExcecaoJogoDaVelha("Célula vazia");
    }

    return celula == "X";
  }

  public void setMarcaPrimeiroJogadorX(boolean b) {
    if (iniciouJogo) {
      throw new ExcecaoJogoDaVelha("O jogo já foi iniciado");
    }
```

```
    proximaJogadaX = b;
    primeiroJogadorDefinido = true;
  }

  private boolean verificarColunas() {
    for (int i = 0; i < 3; i++) {
      if (tabuleiro[0][i] == tabuleiro[1][i] &&
          tabuleiro[1][i] == tabuleiro[2][i] && tabuleiro[2][i] != null)
{
        return true;
      }
    }
    return false;
  }

  private boolean verificarDiagonais() {
    if (tabuleiro[0][0] == tabuleiro[1][1] &&
        tabuleiro[1][1] == tabuleiro[2][2] && tabuleiro[2][2] != null) {
      return true;
    }

    if (tabuleiro[0][2] == tabuleiro[1][1] &&
        tabuleiro[1][1] == tabuleiro[2][0] && tabuleiro[2][0] != null) {
      return true;
    }

    return false;
  }

  private void verificarLimites(int linha, int coluna) {
    if (linha < 0 || linha > 2) {
      throw new ExcecaoJogoDaVelha("Linha fora da faixa permitida");
    }

    if (coluna < 0 || coluna > 2) {
      throw new ExcecaoJogoDaVelha("Coluna fora da faixa permitida");
    }
  }

  private boolean verificarLinhas() {
    for (int i = 0; i < 3; i++) {
      if (tabuleiro[i][0] == tabuleiro[i][1] &&
          tabuleiro[i][1] == tabuleiro[i][2] && tabuleiro[i][2] != null)
{
        return true;
      }
    }
    return false;
  }

}


public class ExcecaoJogoDaVelha extends RuntimeException {

  public ExcecaoJogoDaVelha(String msg) {
    super(msg);
  }
}
```

Implementação quase finalizada

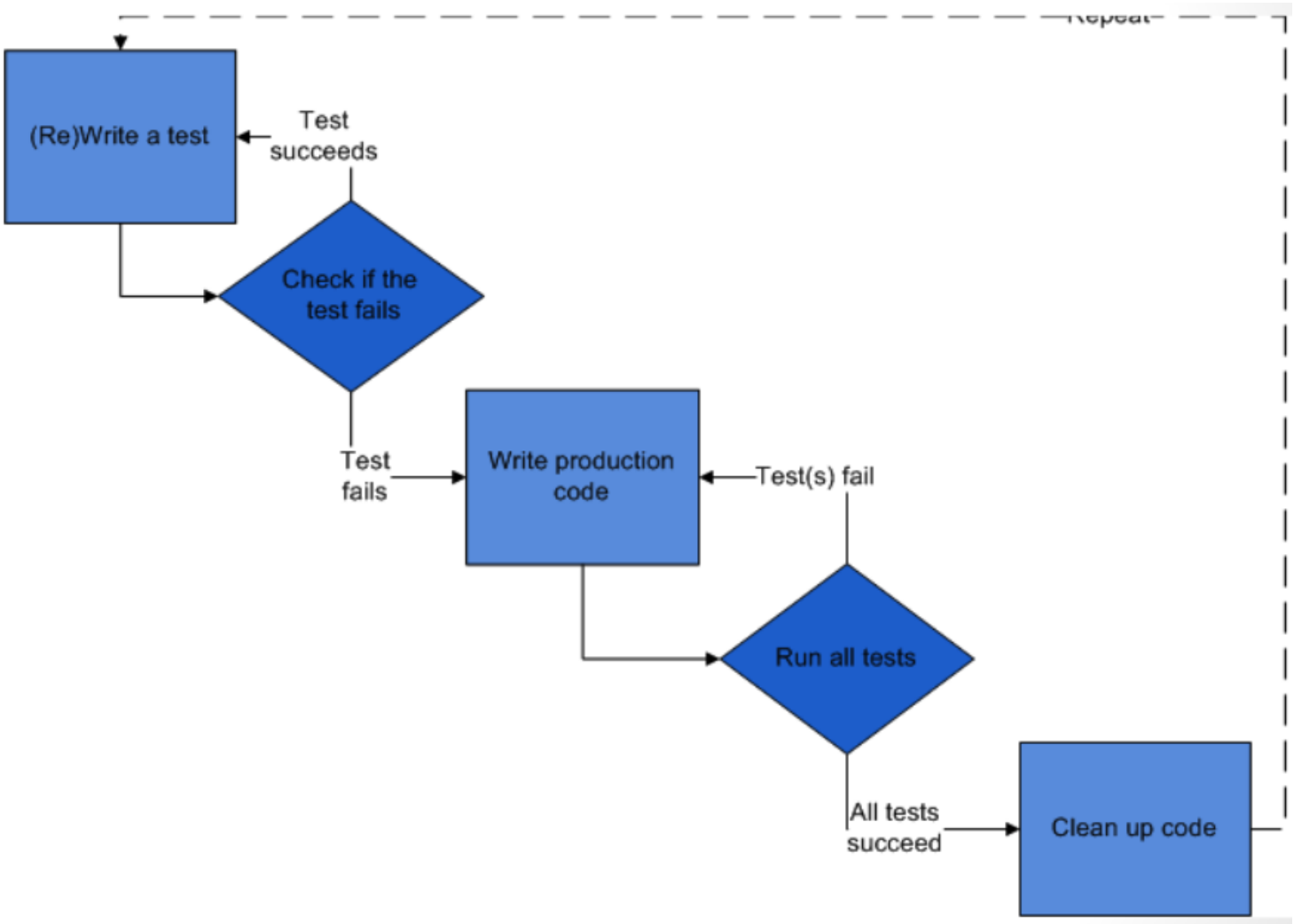
- Falta a interface com o usuário - UI
- O autor deste artigo fez uma GUI em 8 minutos:



Vantagens de TDD

- Cria uma camada de código de lógica
  - Este mesmo código pode ser invocado por uma aplicação web
- Simplifica o projeto
- Ajuda a criar as interfaces
- Torna o código mais testável
- Os testes funcionam como documentação
- Confere robustez ao código
- Antecede o programador às falhas
- Permite o refatoramento do código com segurança

Resumo do processo TDD



Existe apenas um erro nesse diagrama...

Depois de refatorar o código, é preciso rodar todos os testes novamente.

Dicas Finais

- Ciclo Red/Green/Refactor
- Esteja sempre a um passo do Green
- Não faça muitas mudanças de uma vez só
- Coloque os testes no mesmo projeto do código
- Isole os testes uns dos outros
- Lembre-se das assertivas
- Encontrou um bug? Escreva um teste que o capture
- Não refatore sem testes que protejam o código

Publicado por Google Drive – Denunciar abuso – 5Atualizado automaticamente a cada minutos