UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA

CURSO: Engenharia de Software
DISCIPLINA: Teste de Software I

Projeto de Teste de Software - Jogo Master Mind

NATAL/RN
2012

Anderson de Paiva Rodrigues
Felipe Cordeiro Alves da Silva

Projeto de Teste de Software - Jogo Master Mind

Projeto de Teste Software para análise do jogo Master Mind, requisitado pela disciplina de Teste de Software I, da Universidade Federal do Rio Grande do Norte, sob a orientação da Profa. Dra. Roberta Coelho.

Classe Tentativa

/
**** Classe que representa uma tentativa realizada pelo Adivinho para tentar adivinhar uma Senha**
**** criada pelo FornecedorDeSenha.**
****/**

Função testada: void adicionarPino(int posicao,String cor) throws PosicaoInvalidaException,CorInvalidaException;
Estratégia de combinação: Each Choice

Modelo do domínio de entrada
Conjunto dos números reais para o parametro posicao, conjunto das Strings.

Características do parametro <u>posicao</u> :
Intervalo de posições válidas e inválidas para um vetor de 4 posições.

Blocos
B1: posicao válida
B2: posicao inválida para menos
B3: posicao inválida para mais
A ferramenta Eclemma usada para verificar a cobertura dos testes, apontava que o teste não estava cobrindo todas as possibilidades na instrução “if(posicao < 0 || posicao > 3)”, quando foi usado só dois blocos (um para válida, e um para inválida). A ferramenta indicava que para melhorar a cobertura, devíamos dividir o parâmetro em 3 blocos: menor que 0, entre 0 e 3, e maior que 3.

Características do parametro <u>cor</u> :
Cores válidas e inválidas para a aplicação

B1: cor válida
B2: cor inválida

Testes

T1: (posicao válida e cor válida)
input: adicionarPino(1, laranja);
output: true;

T2: (posição válida e cor inválida)
input: adicionarPino(0, marrom);
output: exception capturada;

T3: (posição inválida e cor válida)
input: adicionarPino(0, marrom);
output: false;
* Nesse método de teste, ambos limites são testados, tanto o superior quanto o inferior e ambas exceções são capturadas

Função testada: boolean CorEhValida(String c1);
Estratégia de combinação: AllChoices

Modelo do domínio de entrada
Cojunto das strings válidas e inválidas.

Características do parametro c1 :
c1 é uma cor válida ou inválida

Blocos
B1: c1 é cor válida
B2: c1 é cor inválida

Testes
T1: testCorEhValida(); *
* Todas as cores foram testadas apenas para garantir a cobertura completa do código, visto que há um if no qual todas as cores são comparadas. No caso, poderíamos testar somente as comparações limites do if, mas dessa forma o código teria cobertura parcial. Em teoria, testamos o fluxo (todas as linhas).
T2: testCorEhInvalida();

Função testada: String getPino(int posicao);
Estratégia de combinação: Each Choice

Modelo do domínio de entrada
Cojunto dos números inteiros.

Características do parametro posicao :
Conjunto dos números inteiros.

Blocos
B1: posicao válida
B2: posicao inválida para menos
B3: posicao inválida para mais

Testes
T1: testGetPinoPosicaoInvalida();
* Assim como no método adicionarPino, foram testados os limites superior e inferior deste método, para garantir cobertura completa pelo Eclemma.
T2: testGetPinoPosicaoValida() throws PosicaoInvalidaException;

Classe Adivinho

```
/**
 ** classe que representa o proprio usuario do nosso jogo. O adivinho adivinha a
 ** senha do jogo em cada uma de suas jogadas. Ele tem 10 jogadas para tentar
 ** descobrir a senha(que eh o objetivo do jogo).
 **
 ** Essa classe Implementa a Interface Jogador, que tambem eh implementada por
 ** FornecedorDaSenha: quem elabora a senha para ser adivinhada.
 **/
```

Função testada: void realizarTentativa(Tentativa tentativaJogador);
Estratégia de combinação: Each Choice

Modelo do domínio de entrada
Conjunto de combinações possíveis para o tipo Tentativa.

Características do parametro <u>tentativa</u> :
Jogadas completas (4 pinos), jogadas incompletas (menos de 4 pinos) e tentativas inválidas (quando a jogada não é setada).

Blocos
B1: tentativa inválida
B2: tentativa completa
B3: tentativa incompleta

Testes

T1: testRealizarTentativaNullPointerException() throws PosicaoInvalidaException, CorInvalidaException
T2: testRealizarTentativaCompleta() throws PosicaoInvalidaException,CorInvalidaException
T3: testRealizarTentativaIncompleta() throws PosicaoInvalidaException,CorInvalidaException

obs1.: Não havia close no scanner.
obs2.: A classe adivinho possui apenas um atributo, que é o atributo “jogadaTurno” do tipo “Jogada”. Esse atributo não é inicializado no construtor da classe e nem na declaração do atributo. Sendo assim, na hora de testar o método “realizarTentativa(Tentativa)”, se o testCase do JUnit não chamar previamente o método “setJogada(Jogada)”, a exceção NullPointerException é lançada. Isso constitui um erro *cometido pelos autores, que levou a uma falha encontrada* pelos testes, já que se o método realizarTentativa for chamado antes do método setJogada, o programa entrará em travamento devido à NullPointerException. Para corrigir o erro, os autores devem inicializar a variável “jogadaTurno” no construtor ou na declaração do atributo OU tratar a exceção NullPointerException que pode ser lançada, no método “realizarTentativa(Tentativa)”

Classe Senha

Função testada: void adicionarPino(String corPino);

Estratégia de combinação: Each Choice

Modelo do domínio de entrada
Conjunto de todas as strings

Características do parametro <u>corPino</u> :
Contém cores válidas ou inválidas.

Blocos
B1: vetor não está cheio (menos de 4 pinos)
B2: vetor está cheio (mais de 4 pinos)

Testes:
T1: testAdicionaPinoAMais() throws CorInvalidaException
T2: testAdicionaPinoCorInvalida()

Classe Retorno

Função testada: void adicionarPino(String corPino);

Estratégia de combinação: Each Choice

Modelo do domínio de entrada
Conjunto de todas as strings

Características do parametro <u>corPino</u> :
Contém cores válidas ou inválidas.

Blocos
B1: cor válida
B2: cor inválida

Testes:
T1: testAdicionarPinoCorValida()
T2: testAdicionarPinoCorInvalida() throws PosicaoInvalidaException;

Algumas considerações importantes

Sobre a classe Advinho

O método *public void adicionarNovoPinoATentativa(Tentativa tentativa)* é não-testável, pois nele é utilizado um Scanner.in; ou seja, é um método dependente da entrada do usuário. Para alcançar nossos objetivos
de teste, simulamos um Stub, criando o método *public void adicionarNovoPinoATentativaTestadores(Tentativa tentativa, String corPino)*, onde a entrada que seria de resposabilidade do usuário, é passada como parâmetro em “String corPino”. No método *jogar()*, seguimos a mesma estratégia; simulamos mais dois stubs, *jogarTestadores()* e *scannerTestadores(Tentativa tentativa).*

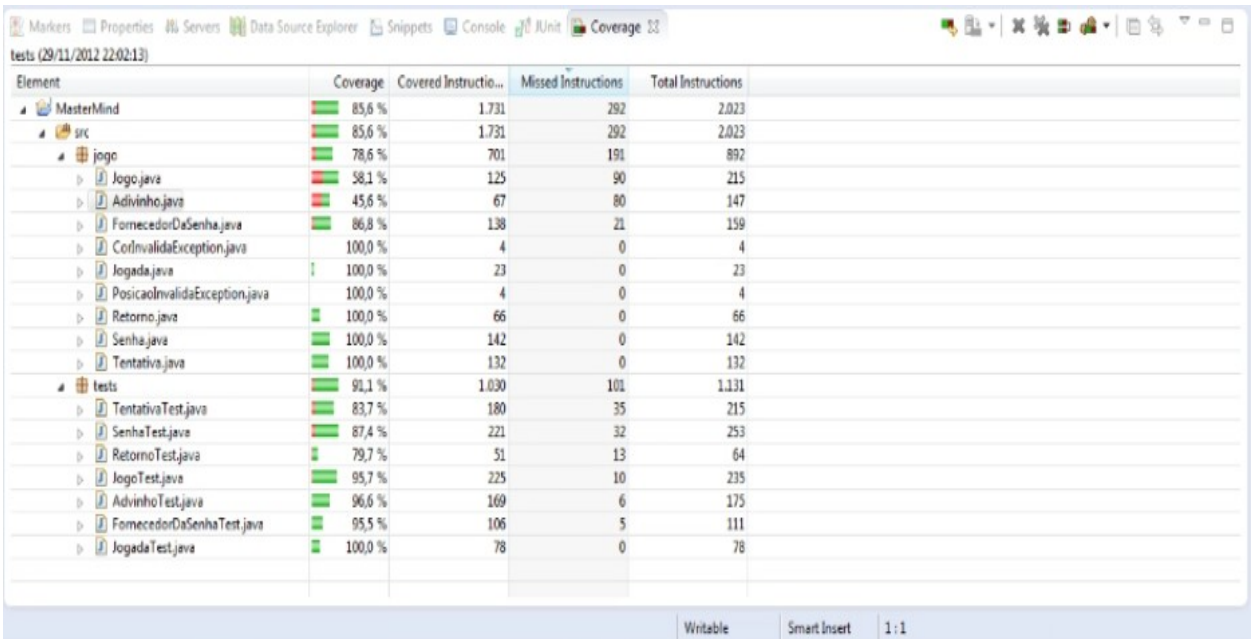
obs.: O código não foi alterado.

Sobre a classe FornecedorDaSenha

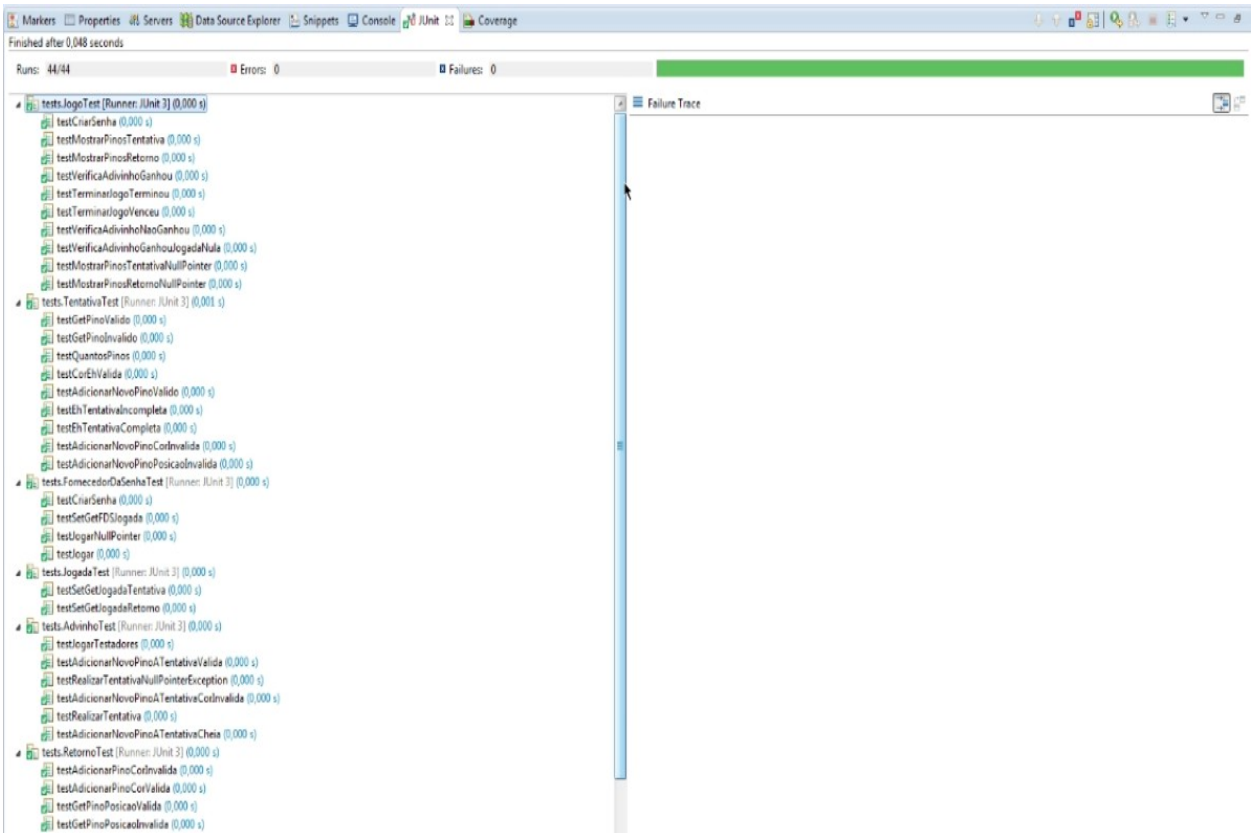
O método *public void testCriarSenha()* não tem assert, pois o método "criarSenha" da classe gera uma senha randômica, com 10680 expectativas.

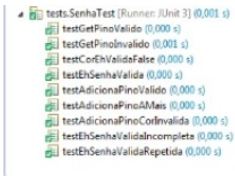
Esse método não usa "throws" para nenhuma exceção. A única exceção (Exception) do método é tratada internamente com try/catch, logo, não há exceptions para o teste capturar. O mesmo acontece no método testJogar().

Cobertura dos testes, análise feita pela Eclemma



Execução dos testes pelo JUnit





Acesso aos Testes (35 testes criados)
Repositório: <***https***://testesoft.googlecode.com>

Sugestões de melhorias:

- Fechar sempre os scanners, visto que isso é uma Boa Prática de Programação; e quando não seguida, pode gerar problemas dependendo da complexidade do software. Ainda nesse quesito, recomenda-se utilizar métodos com nomes mais legíveis, o que facilita na construção dos testes e leitura/entedimento do código.

- Evitar misturar scanners com a lógica do software.

- Lançar exceção em vez de tratá-la dentro próprio método, o que garante a viabilidade dos asserts.