Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática – CEEI
Unidade Acadêmica de Sistemas e Computação – UASC
Disciplina: Laboratório de Programação 2

Laboratório 07

Como usar esse guia:

- Leia atentamente cada etapa
- Referências bibliográficas incluem:
  - material de referência desenvolvido por professores de p2/lp2 em semestres anteriores (ONLINE)
  - o livro Use a cabeça, Java (LIVRO-UseCabecaJava)
  - o livro Java para Iniciantes (Livro-JavaIniciantes)
- Quadros com dicas tem leitura opcional, use-os conforme achar necessário

Sumário

Conteúdo sendo exercitado 2
Objetivos de aprendizagem 2
Perguntas que você deveria saber responder após este lab 2
Para se aprofundar mais... 2
Introdução 2
O sistema que evoluiremos: Dominó Brutal 1.0 3
Parte 1: Explique o design para a gente 3
Parte 2: Evolução do sistema 3
US1: Desempate 4
US2: Pontuação por vitória 4
US3: Duas estratégias suas 4
Extras 4
US5: Campeonato Brutal 4
US6: Só pode passar se não tiver como jogar 4
Sobre a entrega 5

Acompanhe o seu aprendizado

Conteúdo sendo exercitado

- Leitura e entendimento de código e design feito por terceiros
- Evolução de software
- Testes de unidade
- Padrões GRASP: ênfase no uso de expert da informação, creator, baixo acoplamento, alta coesão e controladores
- Uso de interfaces

Objetivos de aprendizagem

Temos dois objetivos principais nesse lab:

1. ter experiência de um lab parecido com um projeto em termos de tamanho;
2. experimentar com entendimento e evolução de um design feito por outra pessoa e envolvendo polimorfismo e design OO.

Para se aprofundar mais...

- Referências bibliográficas incluem:
  - material de referência desenvolvido por professores de p2/lp2 em semestres anteriores (ONLINE)
  - o livro Use a cabeça, Java (LIVRO-UseCabecaJava)
  - o livro Java para Iniciantes (Livro-JavaIniciantes)
- Projetando com interfaces
- Um pouco mais sobre GRASP

Introdução

Neste laboratório queremos novamente experimentar com como entender e estender um software OO. Partiremos de um sistema pronto e iremos introduzir novas funcionalidades nesse sistema e garantir que o sistema esteja correto após as modificações. Vamos exercitar a leitura e extensão de código escrito por outra pessoa.

O uso de testes automáticos é essencial para evoluirmos um código grande. É a única forma prática de conferirmos que um grande número de funcionalidades ainda está OK depois de alguma modificação.

Diferente de nosso Laboratório anterior, nesse laboratório **usaremos apenas testes de unidade, e você será responsável por evoluir os testes do projeto**. Um ponto muito importante desse laboratório é que **usaremos os testes de unidade para entender o que você implementou.** Leremos os testes para entender o código. O que não estiver testado e compreensível nos testes não será considerado.

O sistema que evoluiremos: Dominó Brutal 1.0

O sistema no qual você trabalhará é um jogo de Dominó. Ele não é interativo: damos a ele as estratégias dos jogadores, ele simula o jogo e nos diz quem ganhou. É um sistema inspirado no Robocode, mas atualizado para um esporte mais relevante e complexo do guerras de robôs, o dominó. Assim como no Robocode, qualquer pessoa pode programar novas estratégias e plugá-las no sistema para testá-la contra outras estratégias dela ou de outras pessoas.

Você baixará o código inicial do sistema **neste repositório no github** (clica em code, download zip). Sua primeira tarefa é entender o código atual e conseguir descrevê-lo. Explicamos mais na próxima seção. Depois desse entendimento, temos as extensões que queremos que você faça.

Parte 1: Explique o design para a gente

Faça isso respondendo **esse quiz no Canvas**.

Parte 2: Evolução do sistema

Evolua o Dominó Brutal versão 1.0 para a versão 2.0 (Dominó Ainda Mais Brutal) com as seguintes funcionalidades:

US1: Desempate

Como jogador, desejo evoluir a avaliação de vencedor do jogo como dito a seguir, para ter um dominó com regras mais realistas. A modificação é que quando o jogo chega em uma situação onde ambos os jogadores passaram a vez (o atual empate), deve haver um vencedor da seguinte maneira:

a. Se um dos jogadores tiver menos peças na mão que o outro, ele será vencedor.
b. Se um dos jogadores tiver uma soma menor nos números das peças de sua mão que o outro, ele será o vencedor.

O jogo só deve ser empate se ambos tiverem o mesmo número de peças e a mesma soma no número das peças.

US2: Pontuação por vitória

Como jogador, quero que lá e lô e carroção sejam vitórias que valem mais, para ter regras mais realistas. A pontuação, retirada da Wikipedia, é a seguinte: *uma batida normal (em uma única "cabeça") vale 1 ponto, mesma pontuação quando o jogo trancar e acontecer a contagem, batida de "carroça"* (uma peça com 2 números iguais) *vale 2 pontos, o famoso "lá e lô" que significa bater com uma pedra simples* que encaixa *nas duas pontas, vale 3 pontos, já o "lá e lô" de carroça, também chamada de "quadrada", "cruzada" ou "carroça cruzada" vale 6 pontos.* Após implementar essa pontuação, altere a classe DominoBrutalRepetido para considerar pontuações em vez de vitórias, e para contar e imprimir quantas vezes cada jogador teve cada tipo de vitória (você escolhe o formato).

US3: Duas estratégias suas

Como jogador, quero ter duas estratégias de jogo novas, para que eu possa me divertir mais jogando. Para isso, você deve criar duas implementações de EstrategiaDeJogo. Exemplos de estratégias possíveis são guardar o número que mais se repete na mão para o final do jogo, jogar sempre o número que o oponente mais jogou até agora, ou jogar sempre primeiro os carroções (peças com 2 números iguais). Mas pode inventar a sua do jeito que quiser.

Nós estudaremos o que sua estratégia faz lendo os teste de unidade dela, então capriche nisso. Além disso, nós queremos saber quão boas são as estratégias que você desenvolveu, então você deve submeter no canvas qual a pontuação que suas estratégias obtêm *ao rodar a classe DominoBrutalRepetido competindo: 1. elas entre si e 2. elas contra a estratégia JogaPrimeiraPossivel.* Submeta esse texto **nessa tarefa no canvas**.

Extras

US4: Campeonato Brutal

Como jogador, desejo executar um campeonato onde todas as estratégias que eu escolher jogam 10 mil vezes contra todas as outras, e ao final vejo um ranking das estratégias que mais pontuaram.

US5: Só pode passar se não tiver como jogar

Em um jogo de dominó entre humanos, um jogador só pode passar a vez se ele não tem como jogar. Implemente essa condição no nosso sistema, fazendo com caso uma estratégia decida passar quando ela possui alguma peça que encaixa na mesa, isso leve a uma EstrategiaInvalidaException e pare o jogo.

Sobre a entrega

Você está recebendo um código que tem testes e onde todos passam. Você deve entregar um código que tem mais testes, e que permite que entendamos as modificações que você fez no projeto lendo os seus testes. Ou seja, tudo que você mudar deve ser testado. Para o que você mudar que precise de alterações nos testes que passamos, altere-os e atualize-os. Considerando isso, faça um **zip** da pasta do seu projeto. Coloque o nome do projeto para:

- LAB7_SEU_NOME e o nome do zip para LAB7_SEU_NOME.zip. Exemplo de projeto: LAB7_LIVIA_SAMPAIO.zip. Este zip deve ser submetido pelo Canvas.

Seu programa será avaliado pela corretude, pela facilidade de compreendermos o que você fez a partir dos testes, e pelo DESIGN do sistema.