UNIVERSIDADE FEDERAL DO CEARÁ

Documentação do projeto da disciplina Projeto detalhado de software.

Profa. Paulyne Matthews Jucá - paulyne@ufc.br

Desenvolvendo uma versão minimalista do jogo Perfil.

Desenvolvido por

Paula Luana Oliveira da Silva, 418105

1. Apresentação das funcionalidades do jogo:

O jogo apresenta uma versão minimalista do jogo Perfil. Nessa versão, o jogo possui 6 cartas criadas e cada carta possui 5 dicas, onde entre as dicas sempre há uma ação. Para essa versão foi escolhida uma ação que é “perca sua vez”.
Ao executar o jogo, é apresentado que no jogo o mediador é sempre o computador e que todos os jogadores iniciam na posição 0 do tabuleiro.

Após isso, é pedido a quantidade de jogadores e em seguida, mostra como o tabuleiro, com todos os jogadores na casa 0, inclusive o computador. Em seguida, uma carta é sorteada e o seu tipo é apresentado.

Com uma carta sorteada, as numerações das dicas são apresentadas, já que no jogo real os jogadores podem visualizar quais dicas já foram pedidas. A cada jogador que pede uma dica, o jogo mostra as dicas disponíveis para escolha.

Quando uma dica é uma ação, ou seja, um “perca sua vez”, o mediador(computador) mostra o texto de “perca sua vez”, mas não chega a pedir um palpite. Quando é uma frase, ele mostra a frase e pede o palpite em seguida. Se o palpite estiver certo, ele informa ao jogador que está correto, atualiza o tabuleiro e mostra as novas posições. Já quando um palpite estiver errado, se ainda houver dicas disponíveis, ele dá dica para o próximo jogador. Já se as dicas acabarem e ninguém acertar o palpite, o computador anda, mostra o tabuleiro e sorteia uma nova carta, iniciando uma nova partida.

2. Interfaces criadas no trabalho:

ICarta: Criamos a ICarta para implementar na CartaAbstrata, que por consequência seus filhos podem herdar os mesmo métodos. Criar essa interface possibilita o uso do padrão fábrica na criação de cartas.

IDica: Criamos a IDica a fim de aplicar a injeção de dependência, já que a classe carta precisa de uma lista de dicas.

Observador: Essa interface permite que usemos o padrão observador. Ela possui o protótipo linhaDeChegada, que usamos para verificar mudança de estado e caso necessário notificar.

Observavel: Possui o protótipo usado por Observador. Ou seja, o protótipo de método que notifica a mudança de estado.

3. Classes entidades: classes que guardam.

Dica: Possui como atributo tipo e texto. Para informar se a dica é uma ação ou uma frase. O texto é o que vai ser mostrado para o jogador.

CartaAbstrata: Classe que implementa ICarta e em seu construtor recebe uma injeção de dependência ao receber uma lista IDica. Também possui o atributo um arraylist de dicas usadas. Essa é a classe pai dos tipos de cartas que temos no jogo.

CartaAno: Carta que estende carta abstrata, aplicando polimorfismo. A classe guarda uma lista de IDica e uma string resposta.

CartaPessoa: Carta que estende carta abstrata, aplicando polimorfismo. A classe guarda uma lista de IDica e uma string resposta.

CartaLugar: Carta que estende carta abstrata, aplicando polimorfismo. A classe guarda uma lista de IDica e uma string resposta.

CartaCoisa: Carta que estende carta abstrata, aplicando polimorfismo. A classe guarda uma lista de IDica e uma string resposta.

BancoDeCartas: Classe responsável por guardar as cartas. Possui como parâmetros, uma lista de cartas usuais e um arraylist de cartas usadas.

Jogador: A classe Jogador guarda as informações de id e posição do jogador. Criamos o parâmetro posição para que não precisássemos percorrer o tabuleiro para saber a posição do jogador. Dessa forma, quando um jogador anda no tabuleiro, atualizamos sua posição no tabuleiro e o atributo posição na classe Jogador. Quando queremos calcular a nova posição do jogador, basta somar quantas dicas não foram gastas com a posição do jogador salva na classe Jogador.

Casa: Toda casa no tabuleiro tem um número e têm jogadores. Sendo assim, a classe Casa tem um numero(int) e um arraylist de jogadores, já que em uma casa pode haver mais de um jogador.

Tabuleiro: Possui a quantidade de casas(int) e um arraylist de casas. Temos a quantidade de casas a fim de poder saber quando podemos fazer com que os jogadores parem de andar, já que não podemos definir o tamanho de um arraylist. Assim, quando iniciarmos um tabuleiro podemos verificar também se o tamanho do arraylist de casa corresponde ao atributo número de casas.

4. Classes controllers: classes que controlam.

Esse tópico tem o intuito de explicar o que os métodos das classes controllers fazem.

Mediador: O mediador é o computador. Essa classe possui os métodos que devem ser exercidos pelo computador. Essa classe praticamente só chama métodos de outras. Ela foi criada a fim de deixar claro as funções exercidas pelo mediador no jogo Perfil.

Justificativa do método: Utilizado para ser aplicado o padrão singleton.
public static Mediador getInstance(): retorna uma instância estática do Mediador. A fim de que no nosso jogo, haja apenas um objeto Mediador, que pode ser chamado em outras classes.

Justificativa do método: Esse método nos ajuda reduzindo o código no ControllerJogo e com esse método reduzimos um pouco as chamadas dos controllers na classe ControllerJogo.
public IDica pedeDica(`ICarta carta, int num`): Retorna a dica daquele respectivo número de carta.

Justificativa do método: Impede que peçamos um palpite ao jogador, identificando se a dica é ação ou frase.
public boolean confereDica(Dica dica): O computador confere se vai receber o palpite ou não do jogador. Retorna verdadeiro quando é para pedir palpite.

Justificativa do método: Da forma que colocamos esse método, o seu retorno vai funcionar muito bem para conferirmos se o palpite está correto.
public String pedePalpite(): Pede o palpite já usando o scanner e retorna a string.

Justificativa do método: Usa o retorno anterior para informar se o palpite foi correto ou não. Se for correto, podemos andar nas casas.
public boolean conferePalpite(ICarta c, String palpite): Confere se o palpite dado é verdadeiro. Retorna true, se for verdadeiro, false caso contrário.

ControllerDica:

Justificativa do método: Utilizado para ser aplicado o padrão singleton.
public static ControllerDicas getInstance(): retorna uma instância estática do ControllerDicas. A fim de que no nosso jogo, haja apenas um objeto ControllerDicas, que pode ser chamado em outras classes.

Justificativa do método: Esse método vai servir quando formos criar as dicas para passar para a carta.
public IDica criaDica(int tipo, String texto): Cria o objeto Dica ao receber o tipo da dica e o texto que ela contém. Serve para quando

Justificativa do método: Essa classe vai ser chamada pelo mediador.
public IDica retornaDica(ICarta carta, int i): Retorna dica i-1 da carta.

Justificativa do método: Quando usamos uma dica da carta, ela precisa ser “descartada” para não ser usada novamente. Mas estamos guardando as dicas usadas no arraylist cartasUsadas.
public ICarta removeDica(ICarta carta, IDica dica): Recebe a dica e remove da carta, retornando a carta atualizada.

Justificativa do método: Esse método serve para verificar no final da rodada se ainda tem dica, porque caso não tenha o mediador(computador) já pode ser classificado como ganhador.
public boolean temDica(ICarta carta): Verifica se ainda tem dica disponível na carta. Retornando verdadeiro caso tenha uma dica, e falso caso contrário.

Justificativa do método: Esse método verifica se uma dica é ação ou frase, o que nos ajuda a evitar que um jogador dê palpite sem poder dar.
public boolean verificaDica(Dica dica): Verifica se dica é ação ou frase para poder pedir palpite. Se verifica retornar true, pode pedir palpite, caso contrário, pula o jogador atual, já que a única ação que temos é “perca a vez”.

Justificativa do método: Essa função vai servir quando quisermos calcular o quanto um jogador que acertou vai andar.
public int dicasUsuaisSize(ICarta carta): Contabiliza a quantidade de dicas que ainda podem ou poderiam ser usadas.

Justificativa do método: No jogo real, os jogadores podem ver as dicas que já foram usadas. Nesse caso, usamos esse método para mostrar quais numerações de dicas que eles ainda podem usar.
public void mostraDicasDisponiveis(ICarta carta): Usamos uma lista em dicas disponíveis ao invés de um arraylist, para não perder as numerações das dicas. Esse método printa os números de dicas que estão disponíveis.

ControllerCarta:

Justificativa do método: Utilizado para ser aplicado o padrão singleton.
public static ControllerCarta getInstance(): retorna uma instância estática do ControllerCarta. A fim de que no nosso jogo, haja apenas um objeto ControllerCarta, que pode ser chamado em outras classes.

Justificativa do método: Esse método vai nos ajudar a criar a carta. Para que posteriormente, a gente crie uma lista de cartas para passar para o banco de cartas.
public ICarta criaCarta(IDica dicas[], String tipo, String resposta): Cria uma carta com os argumentos passados para o método e a retorna.

Justificativa do método: Esse método retira a carta que será sorteada das cartas usuais e adiciona em cartas usadas. Retornando verdadeiro, caso dê certo usar a carta.
public boolean usaCarta(Jogo jogo, ICarta carta): Percorre as casas usuais do banco de cartas, quando encontra a carta que foi passada por parâmetro, remove a carta do arraylist de cartas usuais e adiciona em cartas usadas.

Justificativa do método: Queremos que os métodos nas classes sejam só aqueles que tem maior relação possível. Para esse caso, só a carta tem a resposta, então o colocamos no ControllerCarta. Note que, o mediador chama esse método, porque no jogo real é o mediador que olha se o palpite do jogador foi certo.

public boolean verificaResposta(ICarta c, String resposta): Verifica se a resposta do jogador está correta)

Justificativa do método: Esse método está nessa classe, a fim de reduzir o código da classe que executa o jogo.

public ArrayList<ICarta> addCartasDoJogo(): Método que retorna um ArrayList das cartas, prontas para incluir no jogo.

ControllerBancoDeCartas:

Justificativa do método: Utilizado para ser aplicado o padrão singleton.

public static ControllerBancoDeCartas getInstance(): retorna uma instância estática do ControllerBancoDeCartas. A fim de que no nosso jogo, haja apenas um objeto ControllerBancoDeCartas, que pode ser chamado em outras classes.

Justificativa do método: Já que todas as cartas pertencem ao banco, o melhor encaixe para esse método foi na classe banco de cartas.

public ICarta sorteio(BancoDeCartas bancoDeCartas): Realizar um sorteio entre as cartas usuais do banco de cartas.

ControllerJogador:

Justificativa do método: Utilizado para ser aplicado o padrão singleton.

public static ControllerJogador getInstance(): retorna uma instância estática do ControllerJogador. A fim de que no nosso jogo, haja apenas um objeto ControllerJogador, que pode ser chamado em outras classes.

Justificativa do método: Posteriormente, vamos usá-lo para auxiliar na criação de lista de jogadores.

public Jogador criarJogador(String id): Método que recebe um id e retorna um jogador.

Justificativa do método: A classe jogo precisa receber uma lista de jogadores. Esse método entrega isso.

public ArrayList<Jogador> addJogadoresNoJogo(int n): Cria um ArrayList de jogadores numerados de 0 à n, onde o 0 representa nosso mediador(computador) que vamos usar para apenas andar casas e os jogadores de 1 à n são os jogadores que realmente vão jogar.

Justificativa do método: Precisamos do controle da vez, já que o número de jogadores é um parâmetro que pode ser variado de acordo com o que o usuário passa.

public Jogador controlaVez(ArrayList<Jogador> jogadors, Jogador ultimoJogador): acessa o último jogador e retorna qual o próximo a jogar.

Justificativa do método: Esse método nos auxilia a mostrar a situação atual do tabuleiro.
public void mostraJogadores(ArrayList<Jogador> jogadors): Mostra em formato de texto a lista de jogadores.

ControllerTabuleiro:

Justificativa do método: Utilizado para ser aplicado o padrão singleton.
public static ControllerTabuleiro getInstance(): retorna uma instância estática do ControllerTabuleiro. A fim de que no nosso jogo, haja apenas um objeto ControllerTabuleiro, que pode ser chamado em outras classes.

Justificativa do método: Com esse método não vamos precisar criar um ArrayList de Casas na classe ControllerJogo. Esse método já inicializa da forma adequada para nós.
public Tabuleiro iniciaTabuleiro(): Retorna um tabuleiro que possui as casas vazias.

Justificativa do método: Já faz todas as alterações nas casas em que precisamos e retorna o número da casa. Isso nos beneficia porque vamos poder verificar se a casa atual já é a casa da linha de chegada.
public int andaNoTabuleiro(Tabuleiro tabuleiro, ICarta carta, Jogador jogador): Calcula qual será a nova casa do jogador, o remove da casa antiga e adiciona na nova casa. Retorna o novo número da casa

Justificativa do método: o cálculo do jogador e do computador andar são diferentes. Por isso, a necessidade desse método.
public int computadorAndaNoTabuleiro(Tabuleiro tabuleiro, ICarta carta, Jogador computador): Calcula qual será a nova casa do computador, o remove da casa antiga e adiciona na nova casa. Retorna o novo número da casa

Justificativa do método: Toda vez que algum jogador anda a gente precisa verificar se o jogador chega na linha de chegada.
public boolean linhaDeChegada(Tabuleiro tabuleiro, int numeroDaCasa, Jogador jogador): Recebe o número da casa gerada pelo método anterior e retorna verdadeiro caso o número seja o último do tabuleiro. Ou seja, indica true caso a casa seja a linha de chegada e false caso contrário. O método chama o método notifique caso o jogador tenha chegado na linha de chegada.

Justificativa do método: Precisamos mostrar o tabuleiro toda vez que algum jogador anda no tabuleiro.
public void mostraTabuleiro(Tabuleiro tabuleiro): Mostra a situação atual do tabuleiro.

Justificativa do método: Usado para implementar o padrão observador junto com o método linhaDeChegada()
public void notifique(Jogador jogador): Recebe o jogador e informa que ele ganhou o jogo.

ControllerJogo:

public void jogo(): Método responsável por fazer o jogo acontecer.

Justificativa do método: É necessário que as rodadas tenham uma ordem e controle. Esse método faz iso.
public Jogador rodada(Jogo jogo, ICarta carta, Jogador atual, Jogador computador): Método responsável por controlar a rodada de uma carta.

Justificativa do método: É uma forma de diminuir o código jogo()
public Jogo inicializa(int qJogadores): Método responsável por inicializar o jogo, criando as cartas, criando o arraylist de jogadores, iniciando banco de cartas e tabuleiro. Com todos esses parâmetros retornar um objeto Jogo.

Principal: Classe que executa ControllerJogo.jogo()

5. Decisões do projeto.

Em relação ao jogo: O jogo funciona de uma forma bem simples. Nessa versão, o computador sempre é o mediador. O jogo recebe a quantidade de jogadores e já os inicia na casa 0 do tabuleiro. O tabuleiro foi programado para ter 11 casas e as casas foram numeradas de 0 à 10. O tabuleiro é mostrado durante o jogo, a fim de que o jogador acompanhe as posições.

Em relação às escolhas tomadas no desenvolvimento:

1. Coesão e acoplamento.
   A coesão acontece quando uma classe assume responsabilidades que não são suas. Além disso, um método pode ser considerado como não coeso quando ele realizar várias funcionalidades.

   Classes coesas:
   - As classes do pacote entidade, são coesas por terem métodos simples e específicos.
   - ControllerBancoDeCartas: É uma classe coesa, pois a única coisa que precisa fazer é realizar um sorteio entre as cartas que pertencem ao banco.
   - ControllerCarta: Possui métodos relacionados apenas a Carta e não possui nenhuma dependência de outra classe.
   - ControllerCasa: A classe possui apenas um método, que é responsável por inicializar as casas no tabuleiro. Colocando suas devidas numerações e pondo os jogadores na primeira casa do tabuleiro.

- ControllerDica: Possui muitos métodos, mas todos são relacionados à dica, como criar dica, retornar dica, remover dica, identificar se tem dica, etc. A classe não possui nenhum acoplamento.
- ControllerJogador: Contém apenas métodos relacionados à Jogador, não dependendo da funcionalidade de nenhuma classe.

O acoplamento é o quanto uma classe depende de outra para funcionar.

Classes que possuem acoplamento:

- ControllerTabuleiro: Podemos considerar que a classe possui um acoplamento médio, visto que precisa de métodos das classes ControllerDicas e ControllerJogador, mas não precisa em todos seus métodos.
- Mediador: É uma classe que foi feita a fim de deixar claro as coisas que o mediador faz no Jogo Perfil, que são: pedir palpite, pedir o número da dica, conferir se o palpite está certo e dizer se a dica é uma frase ou uma ação. Porém a maioria dessas ações já são realizadas em outras classes, porque um dos objetivos deste trabalho foi manter os métodos em classes que se relacionam com os mesmos objetos das ações. Dessa forma, essa classe é fortemente acoplada, porque ela usa muitos métodos de outras classes.
- Controller Jogo: É onde o jogo realmente acontece. Então ele tem um acoplamento fortíssimo, já que depende das funcionalidades de todas as outras classes controllers para acontecer.

2. Injeção de dependência.

A injeção de dependência foi usada na classe Jogo a fim de evitar o alto nível de acoplamento de código dentro da aplicação. Assim, Aumentando a facilidade de implementação de novas funcionalidades.

3. Padrão GOF:

Padrão fábrica: O padrão fábrica encapsula a criação de objeto deixando as subclasses decidirem quais objetos criar. Aplicamos o padrão da seguinte forma: a classe CartaAbstrata é a classe pai das classes CartaPessoa, CartaAno, CartaCoisa e CartaLugar. Na classe CartaAbstrata implementamos a interface ICarta. Tudo isso, seguindo o modelo visto em sala. Dessa forma, a criação de cartas segue o modelo fábrica, fazendo com que a gente não precise declarar o tipo dos objetos filhos.

Padrão observador: O padrão observador é responsável por observar e notificar a mudança de um estado. Pensando nesse princípio, esse padrão foi escolhido para avisar quando um jogador chega na linha de chegada(última casa do tabuleiro). Dessa forma,

notificando a mudança de estado. O padrão foi utilizado através das interfaces Observador e Observável, que tem os protótipos de métodos respectivos, public boolean linhaDeChegada(Tabuleiro tabuleiro, int numeroDaCasa, Jogador jogador) - que toda vez que alguém avança casas ele confere se chegou na linha de chegada e public void notifique(Jogador jogador) - que é chamado pelo método linha de chegada quando alguém atinge a última posição do tabuleiro. Ambas interfaces são implementadas na classe ControllerTabuleiro.

Padrão Singleton: Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto. Sabendo disso, pensamos no padrão Singleton como uma forma de contornar as fortes consequências que o acoplamento feito principalmente na classe ControllerJogo poderia causar, já que por causa desse acoplamento, estaríamos instanciando novos objetos