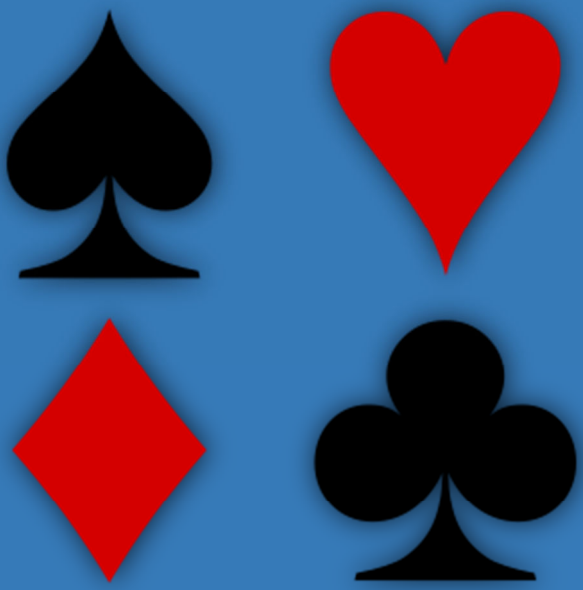

Escola de Ciências e Tecnologia

Disciplina: Estruturas de Dados e Algoritmos I

Docentes: Lígia Ferreira e João Coelho

Solitário

Primeiro Trabalho Prático

Trabalho Elaborado Por:

Marlene Oliveira Nº 25999

Ano Lectivo 2012/2013

Descrição das Classes do Jogo

Foram completados todos os métodos de todas as classes do jogo, à excepção da implementação da Carta, uma vez que foi usada a implementação fornecida sem terem sido feitas alterações importantes. Uma descrição do funcionamento dos métodos de cada classe pode ser consultada de seguida.

Parte 1 – As Cartas e o Baralho

Dado que a classe Carta.java já era fornecida, não foi criada uma classe para o efeito, sendo utilizada a implementação dada. A única alteração que foi feita nesta classe teve como objectivo final corrigir a ordem das cartas na enumeration Valor, que tinha a ordem trocada (em vez de ter a ordem "Rei, Dama, Valete", tinha a ordem "Rei, Valete, Dama").

A classe Carta.java utiliza enumerations para armazenar as constantes correspondentes aos Naipes das cartas e aos Valores das cartas. Uma enumeration pode ser vista como uma lista de constantes fixas [1], sendo por isso que os nomes dos seus elementos se encontram escritos com letra maiúscula.

A classe Baralho.java deverá fornecer um baralho de 52 cartas. A estrutura de dados que irá armazenar as cartas será uma Lista Duplamente Ligada. No construtor da classe Baralho.java são criadas as 52 cartas e adicionadas à referida estrutura de dados. A referida classe contém ainda os métodos get(), toString() e verfica(Carta x).

O método get() retorna uma carta aleatória do baralho, ficando o mesmo com menos uma carta. Para que tal fosse possível, começa-se por calcular um inteiro aleatório que se encontre no intervalo [0,tamanho da lista] e de seguida remove-se da lista o elemento que se encontre na posição correspondente ao inteiro calculado. Finalmente retorna-se o elemento removido da lista.

O método toString() serve apenas para fornecer uma representação das cartas do baralho e, sendo assim, retorna a string resultante da invocação do método toString() da lista duplamente ligada.

Exemplo da representação do baralho retornada pelo método toString():

```
[AS de ESPADAS ; DOIS de ESPADAS ; TRES de ESPADAS ; QUATRO de ESPADAS ; CINCO de ESPADAS ; SEIS de ESPADAS ; SETE de ESPADAS ; OITO de ESPADAS ; NOVE de ESPADAS ; DEZ de ESPADAS ; VALETE de ESPADAS ; DAMA de ESPADAS ; REI de ESPADAS ; AS de OUROS ; DOIS de OUROS ; TRES de OUROS ; QUATRO de OUROS ; CINCO de OUROS ; SEIS de OUROS ; SETE de OUROS ; OITO de OUROS ; NOVE de OUROS ; DEZ de OUROS ; VALETE de OUROS ; DAMA de OUROS ; REI de OUROS ; AS de COPAS ; DOIS de COPAS ; TRES de COPAS ; QUATRO de COPAS ; CINCO de COPAS ; SEIS de COPAS ; SETE de COPAS ; OITO de COPAS ; NOVE de COPAS ; DEZ de COPAS ; VALETE de COPAS ; DAMA de COPAS ; REI de COPAS ; AS de PAUS ; DOIS de PAUS ; TRES de PAUS ; QUATRO de PAUS ; CINCO de PAUS ; SEIS de PAUS ; SETE de PAUS ; OITO de PAUS ; NOVE de PAUS ; DEZ de PAUS ; VALETE de PAUS ; DAMA de PAUS ; REI de PAUS ; ]
```

---

[1] "An enum type is a type whose fields consist of a fixed set of constants." Fonte: http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

O método contains(Carta x) verifica se uma carta se encontra no baralho, retornado true se a carta se encontra no mesmo e false caso contrário. Este método invoca o método contains implementado na classe das Listas Duplamente Ligadas para verificar se a carta se encontra de facto no baralho.

Parte 2 – Zona A

A zona A consiste no baralho de 24 cartas que o jogador pode consultar durante o jogo e do qual pode retirar cartas para efectuar jogadas válidas. A classe ZonaA.java tem como variáveis de instância duas stacks (uma correspondente à pilha de cartas viradas para baixo e outra correspondente à pilha de cartas viradas para cima) com tamanho máximo igual ao número máximo de cartas que podem estar nesta zona (24 cartas). O construtor desta classe recebe como argumento uma lista de cartas, cujos elementos serão armazenados nas pilhas criadas para o efeito. Inicialmente todas as cartas se encontram na pilha das cartas escondidas (viradas para baixo). A classe ZonaA.java contém os métodos virar() e retirar().

O método virar() retira uma carta da pilha das cartas escondidas e coloca-a no topo da pilha das cartas viradas para cima (visíveis). Caso a pilha das cartas escondidas se encontre vazia e a pilha das cartas viradas para cima possua elementos, as cartas desta segunda pilha são transferidas para a pilha das cartas escondidas, permitindo que seja virada uma carta. Se já não existem cartas para virar, é lançada uma excepção. Este método retorna a carta que se encontra no topo das cartas viradas para cima, ou seja, retorna a carta virada.

O método retirar() retira a carta que se encontra no topo da pilha das cartas viradas para cima, ou seja, remove a carta da pilha (utilizando o método pop da pilha) e retorna-a. Caso já não existam cartas para retirar, é lançada uma excepção.

Parte 3 – Zona B

A zona B corresponde à zona do jogo onde se encontram as quatro pilhas de cartas correspondentes a cada um dos naipes. As cartas colocadas nesta zona devem seguir a ordem A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, D, K. Na implementação desta zona foi utilizado um array de quatro pilhas que representa a zona B. Cada uma das pilhas deste array tem tamanho máximo 13, ou seja, cada uma das pilhas só pode conter as 13 cartas que constituem um naipe. No construtor desta classe são apenas inicializadas as quatro pilhas, que se encontram vazias inicialmente. A classe ZonaB.java tem os métodos put(Carta x) e put(Carta x, int i).

O método put(Carta x) coloca uma carta numa das quatro pilhas correspondentes ao naipe da carta. Para cada uma das pilhas verifica-se se estas se encontram vazias. Em caso afirmativo e se a carta dada como argumento for um Ás, a carta é colocada na pilha. Se a pilha se encontrar vazia e a carta não for um Ás, é lançada uma excepção. Caso as pilhas não se encontrem vazias, se o naipe da carta a ser colocada na pilha é igual ao da carta no topo da

pilha e se o valor da carta é o seguinte (valor da carta no topo + 1), a carta é colocada na pilha. Caso o valor da carta não seja o seguinte ao valor da carta no topo da pilha, é lançada uma excepção que indica que a carta que se pretendia colocar na pilha não é a carta seguinte. Caso o naipe seja diferente do da carta no topo da pilha, é lançada uma excepção que indica que os naipes são diferentes.

O método put(Carta x, int i) funciona de um modo semelhante ao anterior, sendo indicada a pilha onde se pretende colocar a carta dada como argumento.

Parte 4 – Zona C

A zona C consiste em 28 cartas distribuídas por 7 pilhas onde se encontram cartas viradas para cima e cartas viradas para baixo. A classe ZonaC.java tem como variável de instância um array bidimensional de stacks com duas linhas e sete colunas. As linhas permitem distinguir as pilhas de cartas viradas para cima das pilhas de cartas viradas para baixo e as colunas permitem distinguir as várias pilhas de cartas. Note-se que a pilha que se encontra na primeira coluna da primeira linha não é utilizada, uma vez que a primeira pilha de cartas não possui cartas viradas para baixo (tem apenas uma carta virada para cima). No construtor da classe que implementa a zona C são inicializadas as várias pilhas com o número de cartas correspondente a cada pilha. Considera-se que, no pior caso, o número máximo de cartas viradas para cima que podem existir corresponde a um naipe, ou seja, a treze cartas. A classe ZonaC.java contém os métodos add(int i, Carta x), add(List<Carta> s, int i), retira(int n, int i) e retira(int i).

O método add(int i, Carta x) permite inserir uma carta dada como parâmetro na pilha indicada por um inteiro que também é fornecido como parâmetro do método. Caso a pilha indicada esteja vazia, a carta só é inserida se o seu valor for REI. Caso contrário é lançada uma excepção que indica que a carta não é um REI. No caso da pilha não se encontrar vazia, a carta é inserida com sucesso se:

- A carta não tiver valor superior ao da carta que se encontra no topo da pilha;
- A carta estiver na ordem correcta, ou seja, se a carta tiver o valor imediatamente anterior ao da carta que se encontra no topo da pilha;
- As cartas não forem do mesmo naipe;
- As cartas forem de naipes de cores diferentes.

Caso algum dos pontos anteriores não se verifique, é lançada uma excepção que indica o porquê da jogada pretendida ser inválida.

O método add(List<Carta> s, int i) funciona de modo semelhante ao anterior, mas permite inserir uma lista de cartas numa pilha. Se durante a inserção de cartas na pilha se verificar uma jogada inválida, é lançada uma excepção.

O método retira(int n, int i) permite retirar n cartas de uma determinada pilha (retira as n cartas da pilha das cartas viradas para cima). Caso se retirem todas as cartas viradas para cima da pilha indicada, é virada uma carta da pilha das cartas viradas para baixo. Se o número

de cartas que se pretende remover for superior ao número de cartas na pilha (superior ao total do número de cartas viradas para cima e de cartas viradas para baixo), é lançada uma excepção. O método retorna uma lista com as n cartas removidas da pilha.

O método retira(int i) funciona de modo semelhante ao anterior, mas permite apenas retirar uma carta de cada vez da pilha indicada pelo inteiro passado como parâmetro do método. Á semelhança do que acontecia com o método anterior, se a pilha de cartas viradas para cima se encontrar vazia, é também virada uma carta da pilha de cartas viradas para baixo. Este método retorna a carta removida da pilha pretendida.

Parte 5 – O jogo

A classe Solitaire.java tem como variáveis de instância as três zonas que constituem o jogo e um baralho de 52 cartas. No construtor desta classe é inicializado o baralho e as zonas do jogo, sendo as cartas do baralho distribuídas pelas diferentes zonas do jogo (24 cartas para a zona A e 28 cartas para a zona C). Esta classe possui os métodos virar(), retirarZonaA(), addZonaB(Carta x), retiraZonaC(int i, int n), retiraZonaC(int i), addZonaC(int i, List<Carta> l) e addZonaC(int i, Carta x).

O método virar() invoca o método equivalente da classe ZonaA.java, permitindo que seja virada uma carta da zona A.

O método retirarZonaA() permite retirar uma carta da zona A, sendo invocado o método equivalente da classe ZonaA.java.

O método addZonaB(Carta x) permite adicionar uma carta a zona B. para que tal seja possível é invocado o método put(Carta x) da classe ZonaB.java.

Os métodos retiraZonaC(int i, int n) e retiraZonaC(int i) permitem retirar n cartas de uma pilha da zona C e a i-ésima carta carta de uma pilha da zona C, respectivamente. Para que tal seja possível, são invocados os métodos retira(int n, int i) e retira(i), respectivamente.

Os métodos addZonaC(int i, List<Carta> l) e addZonaC(int i, Carta x) permitem inserir uma lista de cartas na i-ésima pilha da zona C e inserir uma carta na i-ésima pilha da zona C. Tal como acontece com os métodos anteriores, para efectuar estas operações são invocados os métodos da classe ZonaC.java próprios para o efeito.

Parte 8 – Tipos Abstractos de Dados

Foram utilizados os seguintes tipos abstractos de dados na implementação do jogo:

- Stacks;
- Listas Duplamente Ligadas.

As Stacks utilizadas na implementação do jogo são as StackArray implementadas na aula prática da disciplina.

As Listas Duplamente Ligadas utilizadas na implementação do jogo são as listas criadas na aula teórica, tendo sido acrescentados os métodos necessários de modo a que as mesmas funcionassem correctamente. Foram implementados todos os métodos da interface List<T> providenciada juntamente com o enunciado .