Programação Orientada por Objetos

2019/2020

Trabalho de Laboratório – Curso EI

6

Objetivos:
Genéricos e Coleções.

Programas:
Na sequência da implementação de uma aplicação de jogos de cartas, pretendem-se criar novas funcionalidade e adaptar algumas das existentes.

Regras de implementação:

- Criar a aplicação utilizando o IDE **Netbeans.**
- Implementar o código necessário e testar no fim de cada nível.
- Use as convenções de codificação adotadas para a linguagem Java (ver **Notas**).

Implementação:

Nível 1:

- Descarregue e descompacte o ficheiro **CardGames.zip** que acompanha este enunciado. Abra o projeto incluído utilizando o IDE **Netbeans**. Este projeto, desenvolvido num laboratório anterior, contém as definições de cartas de jogar e de baralhos.
- Pretende-se possibilitar a ordenação das cartas pela ordem decrescente do seu valor. Este valor, que é guardado no atributo `value` da classe `Card`, depende do jogo onde a carta é utilizada. Sendo assim, comece por implementar a interface `Comparable` do Java na classe `Card`.
- Na classe `Deck`, crie o método `sortByValue` que ordena as cartas pelo seu valor decrescente usando o método `sort` da classe `Collections`.
- Na classe `Deck`, crie também o método `Iterator<Card> iterator()` que devolve um iterador das cartas incluídas nesse baralho. Neste caso, devolva o iterador da lista `cards` existente na classe, usando o método `iterator()` dessa lista.
- No fim do método principal (`main` que está na classe `CardGames`), teste o método `sortByValue` do baralho `deck2`, mostrando, a seguir, as cartas desse baralho, uma por linha, no formato: "**> <nomedacarta> (<valor>)**". Por exemplo: "`> dama de paus (2)`". Use o iterador criado para percorrer as cartas desse baralho.

Nível 2:

- Na classe `Deck`, implemente o método `shuffle` que baralha as cartas existentes. Na implementação, use o método `shuffle` da classe `Collections`.
- No método principal, teste o método anterior.
- Na classe `Deck`, implemente agora o método `List<Card> getCards()` que retorna uma nova coleção com as cartas existentes no baralho. Não deve retornar a coleção existente.
- No método principal, teste o método anterior mostrando a lista de cartas que obteve a partir do `deck2`.
- Ainda no programa principal, remova todas as cartas de ouros do `deck2` e mostre depois as cartas que ficaram no baralho.

Nível 3:

- Crie a classe genérica `ValuedElement<E>` que associa um elemento a um valor inteiro. Inclua apenas um construtor que recebe o elemento e o valor inteiro e os métodos seletores do elemento e do valor.
- Crie agora a classe `Player` que tem apenas o atributo `name` que guarda o nome do jogador. Defina o construtor que recebe o nome do jogador e o método seletor que devolve o nome do jogador.
- Para guardar a pontuação obtida pelos jogadores num jogo crie agora a classe `GameScores`. Nesta classe defina como atributo uma lista de `ValueElement<Player>` que irá associar uma pontuação a cada jogo de um jogador. Use a interface `List` para o tipo deste atributo.

- No construtor sem argumentos da classe `GameScores` crie uma lista ligada (classe `LinkedList<>`) para o atributo definido.
- Na classe `GameScores` inclua o método `insertScore(Player player, int score)` que adiciona um jogador e a pontuação obtida à lista de pontuações.
- Defina ainda, na classe anterior, o método `toString` que devolve um quadro com as 10 primeiras pontuações no seguinte formato:

```
------ Scores ------
1 - MANUEL:         18
2 - JOAO:           15
3 - ANA:            12
4 -
5 -
6 -
7 -
8 -
9 -
10 -
```

  NOTA: Devem ser sempre mostradas 10 entradas, mesmo que a lista tenha menos ou mais do que dez jogadores, tal como está representado acima. **Opcionalmente**, pode devolver a lista de jogadores ordenada pelo valor decrescente das pontuações. Neste caso, implemente a interface `Comparable` na classe `ValuedElement<E>` que ordena pelo valor e faça a ordenação dentro do método `toString` da classe **GameScores**.
- Para testar o código deste nível, crie um objeto `GameScores`, crie os jogadores mostrados no quadro anterior e adicione-os, com as pontuações mostrados, ao objeto criado. Depois, mostre no ecrã o quadro usando o método `toString` definido.

Nível 4:

- A classe `Deck` utiliza uma lista para guardar as cartas. No entanto, esta lista permite inserir elementos em qualquer posição, não sendo a melhor forma de representação de um baralho de cartas. Sendo assim, pretende-se usar uma coleção que funcione como uma pilha de cartas, onde as cartas são colocadas sempre no topo e retiradas também a partir do topo. Este tipo de coleção é implementado, em Java, com a classe `Stack` que também é um tipo de lista. Não necessita de fazer mais alterações ao código, uma vez que a pilha é um tipo de lista. Nota: atualmente a classe `Stack` caiu em desuso no Java e aconselha-se a utilização de outra classe de coleção que implemente as suas funcionalidades. Desafio (**opcional**): implemente as funcionalidades que são pedidas a seguir utilizando em substituição da classe `Stack`, uma classe que implemente a interface `Queue`. Mais informação sobre a classe `Stack` em: https://www.journaldev.com/13401/java-stack
- Implemente os seguintes métodos na classe `Deck`:
  - `topCard` – devolve a carta que está no topo do baralho, sem a retirar da coleção;
  - `drawCard` – devolve a carta que está no topo do baralho, retirando-a da coleção;
  - `putCard` – coloca uma carta no topo do baralho, acrescentando-a à coleção;
- No programa principal:
  - Guarde as cartas do `deck1` numa lista de cartas e limpe o deck.
  - Num ciclo, adicione ao `deck1` todas as cartas que estão na lista usando o método `putCard` e mostrando que a carta que está no topo do baralho, usando o método `topCard`;
  - Num segundo ciclo, retire as cartas do baralho, usando o método `drawCard` e mostrando a carta retirada;

Nível 5:

- Pretende-se agora ordenar as cartas de uma baralho pelo naipe e a seguir pelo valor da carta. Neste caso deve aparecer primeiro o naipe espadas, com as cartas ordenadas de Ás a 7 (no caso de um baralho da sueca), seguidas de valete, dama e rei e depois, com a mesma ordenação, os naipes de copas, ouros e paus. Para fazer esta ordenação, comece por criar a classe `CardSuitValueComparator` que implementa a interface `Comparator` onde a ordenação deve ser definida.
- De seguida, crie o método `sort()` na classe `Deck` que usa a classe criada para fazer a ordenação das cartas.
- No programa principal, crie um novo baralho de cartas da sueca, baralhe-o e mostre-o. A seguir ordene as cartas usando o método que criou e mostre novamente as cartas que deverão aparecer ordenadas.

Notas:

Para os identificadores siga as convenções adotadas normalmente, em particular:

1) A notação **camelCase** para o nome das variáveis locais e identificadores de atributos e métodos.
2) A notação **PascalCase** para os nomes das classes.
3) Não utilize o símbolo '_', nem abreviaturas nos identificadores.