INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Engenharia Informática / Informática de Gestão
15 de Janeiro de 2021

Data Limite de entrega: 24/01/2021

BARALHO PERFEITO – IN-SHUFFLE SUPORTADO POR UMA ARRAYDEQUE

Alguns casinos, aplicam no seu software de jogo online uma técnica conhecida como o "baralho perfeito – in-Shuffle". Esta técnica consiste, em primeiro lugar, em dividir um baralho de 52 cartas em duas metades de 26 cartas cada. De seguida, mistura-se as metades intercalando as cartas da seguinte forma: Começa-se com a metade superior e vai-se alternando as metades, pegado na carta inferior da metade e colocamo-la em cima de um novo "monte".

Por exemplo, se o nosso baralho original contém as seis cartas 1 2 3 4 5 6, a metade superior é 1 2 3, e a metade inferior é 4 5 6. O 3 na parte inferior da metade superior torna-se a carta inferior no baralho baralhado. Em seguida, colocamos o 6, que está na parte inferior da metade inferior, em cima do monte baralhado. Em seguida, colocamos 2 em cima, depois 5, 1, e finalmente 4. O novo baralho baralhado é então 4 1 5 2 6 3. Observe que a carta que estava no topo do baralho original está agora em segundo lugar no resultado baralhado, e a carta inferior no baralho original está agora em segundo lugar do fundo no baralho baralhado. Este processo de embaralhamento é chamado de "***in-shuffle***" e é alcançado começando pela metade superior quando movemos as cartas para o resultado baralhado.

Por outro lado, se começarmos com a metade inferior, obtemos um chamado "***out-shuffle***", em que a metade superior original e a metade inferior permanecem nas suas posições no baralho baralhado.

Defina uma classe de baralho de cartas usando uma **DEQUE** para conter as cartas. A classe deve definir métodos para executar o baralho perfeito "in-shuffle" e perfeitos "out-shuffle".

Usando a classe criada, contrua um pequeno ***main*** que:

a) Partindo de um baralho inicial, obtenha o baralho perfeito "in-shuffle" e "out-shuffle".
b) Indique quantos "in-shuffles" perfeitos são necessários para voltar a obter um baralho de n cartas na sua ordem original.
c) Indique quantos "out-shuffles" perfeitos são necessários para voltar a obter um baralho de n cartas na sua ordem original?
d) Pode mover a metade superior de um baralho, que está na posição 0, para qualquer posição desejada **m**, realizando uma sequência de baralhados e "out-shuffles", da seguinte forma. Escreve m em binário. Começando com o 1 mais à esquerda e indo para a direita, executa um "in-shuffle" para cada 1 encontrado e um "out-shuffle" para cada 0. Por exemplo, se m é 8, temos 1000 para o seu equivalente binário. Executaríamos um "in-shuffle" seguido por três "out-shuffles" para mover a metade superior original para a posição 8, ou seja, por isso é a nona carta do topo do baralho. Defina um método para executar este "truque" de cartas.

Exemplo de execução para um baralho de 6 cartas de espadas, movendo as 3 cartas de topo:

a)

```
> Task :run
---Testar In-Shuffle---
IN-SHUFFLE
Baralho Original: [E 1] [E 2] [E 3] [E 4] [E 5] [E 6]
Metade Superior: [E 1] [E 2] [E 3]
Metade Inferior: [E 4] [E 5] [E 6]
Baralhado: [E 4] [E 1] [E 5] [E 2] [E 6] [E 3]


---Testar Out-Shuffle---
OUT-SHUFFLE
Baralho Original: [E 1] [E 2] [E 3] [E 4] [E 5] [E 6]
Metade Superior: [E 1] [E 2] [E 3]
Metade Inferior: [E 4] [E 5] [E 6]
Baralhado: [E 1] [E 4] [E 2] [E 5] [E 3] [E 6]
```

b)

```
---Testar Numero de In-Shuffles ate obter o Baralho Original---
--embaralhar #1--
IN-SHUFFLE
Baralho Original: [E 1] [E 2] [E 3] [E 4] [E 5] [E 6]
Metade Superior: [E 1] [E 2] [E 3]
Metade Inferior: [E 4] [E 5] [E 6]
Baralhado: [E 4] [E 1] [E 5] [E 2] [E 6] [E 3]

--embaralhar #2--
IN-SHUFFLE
Baralho Original: [E 4] [E 1] [E 5] [E 2] [E 6] [E 3]
Metade Superior: [E 4] [E 1] [E 5]
Metade Inferior: [E 2] [E 6] [E 3]
Baralhado: [E 2] [E 4] [E 6] [E 1] [E 3] [E 5]

--embaralhar #3--
IN-SHUFFLE
Baralho Original: [E 2] [E 4] [E 6] [E 1] [E 3] [E 5]
Metade Superior: [E 2] [E 4] [E 6]
Metade Inferior: [E 1] [E 3] [E 5]
Baralhado: [E 1] [E 2] [E 3] [E 4] [E 5] [E 6]

Foi necessario 3 in-shuffles para voltar as 6 cartas do baralho original!
```

c)

```
---Testar Numero de Out-Shuffles ate obter o Baralho Original---
--embaralhar #1--
OUT-SHUFFLE
Baralho Original: [E 1] [E 2] [E 3] [E 4] [E 5] [E 6]
Metade Superior: [E 1] [E 2] [E 3]
Metade Inferior: [E 4] [E 5] [E 6]
Baralhado: [E 1] [E 4] [E 2] [E 5] [E 3] [E 6]

--embaralhar #2--
OUT-SHUFFLE
Baralho Original: [E 1] [E 4] [E 2] [E 5] [E 3] [E 6]
Metade Superior: [E 1] [E 4] [E 2]
Metade Inferior: [E 5] [E 3] [E 6]
Baralhado: [E 1] [E 5] [E 4] [E 3] [E 2] [E 6]

--embaralhar #3--
OUT-SHUFFLE
Baralho Original: [E 1] [E 5] [E 4] [E 3] [E 2] [E 6]
Metade Superior: [E 1] [E 5] [E 4]
Metade Inferior: [E 3] [E 2] [E 6]
Baralhado: [E 1] [E 3] [E 5] [E 2] [E 4] [E 6]

--embaralhar #4--
OUT-SHUFFLE
Baralho Original: [E 1] [E 3] [E 5] [E 2] [E 4] [E 6]
Metade Superior: [E 1] [E 3] [E 5]
Metade Inferior: [E 2] [E 4] [E 6]
Baralhado: [E 1] [E 2] [E 3] [E 4] [E 5] [E 6]

Foi necessario 4 out-shuffles para voltar as 6 cartas do baralho original!
```

d)

```
---Testar Mover as 3 Cartas de Topo---
MOVENDO 3 POSI□□ES
Baralho Original: [E 1] [E 2] [E 3] [E 4] [E 5] [E 6]
3 □ equivalente a 11 em binario.

IN-SHUFFLE
Baralho Original: [E 1] [E 2] [E 3] [E 4] [E 5] [E 6]
Metade Superior: [E 1] [E 2] [E 3]
Metade Inferior: [E 4] [E 5] [E 6]
Baralhado: [E 4] [E 1] [E 5] [E 2] [E 6] [E 3]

IN-SHUFFLE
Baralho Original: [E 4] [E 1] [E 5] [E 2] [E 6] [E 3]
Metade Superior: [E 4] [E 1] [E 5]
Metade Inferior: [E 2] [E 6] [E 3]
Baralhado: [E 2] [E 4] [E 6] [E 1] [E 3] [E 5]

Depois de movido: [E 2] [E 4] [E 6] [E 1] [E 3] [E 5]

Feito.
```

Trabalho a desenvolver

Implementar em Java uma solução para o problema enunciado que cumpra integralmente as especificações a seguir descritas.

Considerações a ter em conta na implementação

- Deverá ser desenvolvida uma aplicação para a consola no IDE NetBeans, JDK 1.7 ou posterior.
- O nome do projeto a criar deverá ter o formato g#NSn_NSn, contendo o nº do grupo atribuído no 1º trabalho (#), o nome (N), sobrenome (S) e nº (n) dos dois elementos do grupo. (Exemplo: g62AnaSilva2045_ RuiAlves2183). Em nenhum outro sítio deve constar qualquer elemento que permita identificar os autores do trabalho.
- Não precisam de incluir quaisquer comentários *javadoc*.
- A estrutura de dados a utilizar, deve ter por base a **DEQUE** desenvolvida nas aulas da unidade curricular e cujo código foi implementado em "myCollections" – sem qualquer alteração.
- A classe a desenvolver ***Baralho***, deve seguir a estrutura apresentada no **ANEXO A**, mas podem acrescentar outros métodos que considerem importantes para a solução do problema.
- A classe ***Carta*** e o ficheiro java dos enumeradores ***Naipe***, devem ser utilizadas tal como fornecidas no **ANEXO B** e não podem alterar o que quer que seja no código dessas classes.
- Nos métodos que não conseguirem implementar, devem simplesmente incluir a seguinte linha de código:
  throw new UnsupportedOperationException("Método não implementado!");

Considerações gerais

- Os grupos formados no primeiro trabalho deverão manter-se.
- Apenas serão aceites para avaliação, trabalhos cuja implementação não apresente qualquer erro de compilação e com um mínimo de funcionalidades perfeitamente operacionais.
- É expressamente proibida a cópia integral ou parcial de código de outras fontes que não a documentação disponibilizada pelos docentes da unidade curricular.
- O trabalho deverá ser entregue por um dos elementos do grupo, dentro do prazo estabelecido, obrigatoriamente no portal de e-learning (em http://virtual.ipb.pt/, escolher <Trabalho Pratico 2> no separador <Atividades>, dentro da área de AED), e em nenhuma situação poderá ser remetido por e-mail.
- Deverá ser submetido o projeto depois de compactado pelo próprio NetBeans. Para o efeito, no menu File do NetBeans, selecionar Export Project->To ZIP (confirmar que o arquivo fica com a extensão ".zip").
- O trabalho apenas poderá ser submetido com um atraso máximo de 5 dias, sendo subtraído, ainda assim, um valor à sua nota por cada dia de atraso.
- Não serão permitidas resubmissões (quando submeter, certifique-se de que se trata da versão final).
- Os alunos poderão ter que defender os seus trabalhos, presencialmente ou por videoconferência, em data a marcar pelo docente, mostrando ter capacidade de implementar o código, compreendê-lo e explicá-lo.
- Para esclarecimentos adicionais sobre o trabalho, usar o fórum de discussão criado na plataforma http://virtual.ipb.pt/: AED (20/2.4) > Fóruns > Trabalhos Práticos > Trabalho Prático 2.
- Para informação adicional sobre o tema podem consultar:
  http://jwilson.coe.uga.edu/emt725/Shuffle/Shuffle.html

ANEXO A

Ficheiro **Baralho.java**

```
import myCollections.ArrayDeque;

public class Baralho {
   private ArrayDeque<Carta> conteudo;

   public Baralho(){
          (…)
   }                                  // fim de construtor

   public Baralho(int numeroDeCartas){
          (…)
   }                                  // fim de construtor

   /** Devolve o conteúdo do baralho.
      @return  o conteúdo do baralho.
   */
   public ArrayDeque<Carta> getConteudo(){
          (…)
   }

   public boolean nextBaralho(…){
          (…)
   }

   /** Executa o baralho perfeito in-shuffle.
   */
   public void inShuffle(….){
          (…)
   }                                  // fim de inShuffle

   /** Executa o baralho perfeito out-shuffle.
   */
   public void outShuffle(….){
          (…)
   }                                  // fim de outShuffle

   /** Mover a carta de topo do baralho um determinado número de
      posições para baixo do baralho.
       @param posicao o número de posições a mover.
       @param messagens quando as mensagens devem ou não ser escritas.
   */
   public void moveTopo(int posicao, boolean messagens){
          (…)
   }                                  // fim de moveTopo

}                  // fim de Baralho
```

ANEXO B

Ficheiro (completo) **Naipe.java**

```
package aedTrabalho_n2;

/**
* @author docentesAED
 */
public enum Naipe {ESPADAS, PAUS, OUROS, COPAS}
```

Ficheiro (completo) **Carta.java**

```
package aedTrabalho_n2;

public class Carta {
   private Naipe naipe;    // naipe da carta
   private int valor = 0;    // o valor da carta

   public Carta(Naipe n, int v){
      setNaipe(n);
       setValor(v);
   } // fim de construtor

   // Definir o naipe da carta.
   private void setNaipe(Naipe oNaipe){
       naipe = oNaipe;   }

   /** Devolve o naipe da carta como uma constante do enumerador Naipe.
      @return  O naipe da carta. */
   public Naipe getNaipe(){
      return naipe;   }

   // Define o valor / número da carta.
   private void setValor (int oValor){
       valor = oValor;   }

   /** Devolve o valor da carta como inteiro.
       @return  o valor da carta. */
   public int getValor(){
       return valor;   }

   public String toString(){
       char naipeCar = ' '; // Versão com caracter para o naipe

       switch (naipe) {
        case ESPADAS:
           naipeCar = 'E';
           break;
        case PAUS:
                   naipeCar = 'P';
                   break;
        case COPAS:
                   naipeCar = 'C';
                   break;
        case OUROS:
                   naipeCar = 'O';
                   break;
      } // end switch

       String valorString = "" + valor;

       return "[" + naipeCar + " " + valorString + "]";
   }
} // fim de Carta
```