INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Computer Engineering / Computer Management
Jan 15, 2021

Delivery deadline: 24/01/2021

PERFECT SHUFFLE - IN-SHUFFLE SUPPORTED BY AN ARRAYDEQUE

Some casinos, apply in their online gambling software a technique known as the "perfect shuffle – in-shuffler". This technique is based, on first, dividing a deck of 52 cards into two halves of 26 cards each. Next, we merge the halves by interleaving the cards as follows. Beginning with the top half and alternating halves, we take the bottom card from a half and place it on top of a new deck.

For example, if our deck contains the six cards 1 2 3 4 5 6, the top half is 1 2 3, and the bottom half is 4 5 6. The 3 at the bottom of the top half becomes the bottom card in the shuffled deck. We then place the 6, which is at the bottom of the bottom half, on top of the shuffled deck. Next, we place 2 on top, then 5, 1, and finally 4. The shuffled deck is then 4 1 5 2 6 3. Notice that the card that was on top of the original deck is now second in the shuffled result, and the bottom card in the original deck is now second from the bottom in the shuffled deck. This shuffle is called an in-shuffle and is achieved by beginning with the top half when we move cards into the shuffled result.

If we begin with the bottom half, you get an out-shuffle, whereby the original top card and bottom card remain in their positions in the shuffled deck.

Define a class of playing-card decks by using a deque to contain the cards. The class should define methods to perform perfect in-shuffles and perfect out-shuffles.

Using the class created, build a little main that:

a) Beginning from the original deck, write the perfect in-shuffle and the perfect out-shuffle.
b) Beginning from the original deck, write the and how many perfect out-shuffles are needed to return a deck of n cards to its original order.
c) Write the and how many perfect in-shuffles are needed to return a deck of n cards to its original order.
d) You can move a deck's top card, which is at position 0, to any desired position m by performing a sequence of in-shuffles and out-shuffles, as follows. You write m in binary. Beginning with the leftmost 1 and proceeding to the right, you perform an in-shuffle for each 1 encountered and an out-shuffle for each 0. For example, if m is 8, we have 1000 for its binary equivalent. We would perform one in-shuffle followed by three out-shuffles to move the original top card to position 8, that is, so it is the ninth card from the top of the deck. Define a method to perform this card trick.

Running example for a deck of 6 cards of spades, moving the top 3 cards:

a)

```
---Testing In-Shuffle---
IN-SHUFFLE
Original Deck: [S 1] [S 2] [S 3] [S 4] [S 5] [S 6]
Top Half: [S 1] [S 2] [S 3]
Bottom Half: [S 4] [S 5] [S 6]
Shuffled: [S 4] [S 1] [S 5] [S 2] [S 6] [S 3]


---Testing Out-Shuffle---
OUT-SHUFFLE
Original Deck: [S 1] [S 2] [S 3] [S 4] [S 5] [S 6]
Top Half: [S 1] [S 2] [S 3]
Bottom Half: [S 4] [S 5] [S 6]
Shuffled: [S 1] [S 4] [S 2] [S 5] [S 3] [S 6]
```

b)

```
---Testing Number of In-Shuffles to Return to  Original Order---
--Shuffle #1--
IN-SHUFFLE
Original Deck: [S 1] [S 2] [S 3] [S 4] [S 5] [S 6]
Top Half: [S 1] [S 2] [S 3]
Bottom Half: [S 4] [S 5] [S 6]
Shuffled: [S 4] [S 1] [S 5] [S 2] [S 6] [S 3]

--Shuffle #2--
IN-SHUFFLE
Original Deck: [S 4] [S 1] [S 5] [S 2] [S 6] [S 3]
Top Half: [S 4] [S 1] [S 5]
Bottom Half: [S 2] [S 6] [S 3]
Shuffled: [S 2] [S 4] [S 6] [S 1] [S 3] [S 5]

--Shuffle #3--
IN-SHUFFLE
Original Deck: [S 2] [S 4] [S 6] [S 1] [S 3] [S 5]
Top Half: [S 2] [S 4] [S 6]
Bottom Half: [S 1] [S 3] [S 5]
Shuffled: [S 1] [S 2] [S 3] [S 4] [S 5] [S 6]

It takes 3 in-shuffles to return a deck of 6 cards to its original Deck!
```

c)

```
---Testing Number of Out-Shuffles to Return to  Original Deck---
--Shuffle #1--
OUT-SHUFFLE
Original Deck: [S 1] [S 2] [S 3] [S 4] [S 5] [S 6]
Top Half: [S 1] [S 2] [S 3]
Bottom Half: [S 4] [S 5] [S 6]
Shuffled: [S 1] [S 4] [S 2] [S 5] [S 3] [S 6]

--Shuffle #2--
OUT-SHUFFLE
Original Deck: [S 1] [S 4] [S 2] [S 5] [S 3] [S 6]
Top Half: [S 1] [S 4] [S 2]
Bottom Half: [S 5] [S 3] [S 6]
Shuffled: [S 1] [S 5] [S 4] [S 3] [S 2] [S 6]

--Shuffle #3--
OUT-SHUFFLE
Original Deck: [S 1] [S 5] [S 4] [S 3] [S 2] [S 6]
Top Half: [S 1] [S 5] [S 4]
Bottom Half: [S 3] [S 2] [S 6]
Shuffled: [S 1] [S 3] [S 5] [S 2] [S 4] [S 6]

--Shuffle #4--
OUT-SHUFFLE
Original Deck: [S 1] [S 3] [S 5] [S 2] [S 4] [S 6]
Top Half: [S 1] [S 3] [S 5]
Bottom Half: [S 2] [S 4] [S 6]
Shuffled: [S 1] [S 2] [S 3] [S 4] [S 5] [S 6]

It takes 4 out-shuffles to return a deck of 6 cards to its original order!
```

d)

```
---Testing Shifting Top Card---
SHIFTING 3 POSITIONS
Original Deck: [S 1] [S 2] [S 3] [S 4] [S 5] [S 6]
3 is equivalent to 11 in binary.

IN-SHUFFLE
Original Deck: [S 1] [S 2] [S 3] [S 4] [S 5] [S 6]
Top Half: [S 1] [S 2] [S 3]
Bottom Half: [S 4] [S 5] [S 6]
Shuffled: [S 4] [S 1] [S 5] [S 2] [S 6] [S 3]

IN-SHUFFLE
Original Deck: [S 4] [S 1] [S 5] [S 2] [S 6] [S 3]
Top Half: [S 4] [S 1] [S 5]
Bottom Half: [S 2] [S 6] [S 3]
Shuffled: [S 2] [S 4] [S 6] [S 1] [S 3] [S 5]

After the shift: [S 2] [S 4] [S 6] [S 1] [S 3] [S 5]

Done.
```

Work to be developed

Implement in Java a solution to the problem that that fully complies with the following specifications:

Considerations to be taken into consideration in the implementation

- The console application must be developed in IDE NetBeans, JDK 1.7 or later.
- The name of the project to be created must have the format g#NSn_NSn, containing the group number assigned on the 1st practical assignment (#), the name (N), Surname (S) and nº (n) of the two students of the group. (Exemple: g62AnaSilva2045_ RuiAlves2183). In no other place should there be any element that allows the identification of the authors of the practical assignment.
- There is no need to include any javadoc comments.
- The data structure to be used, must be based on the **DEQUE** developed in the DSA classes and whose code was implemented in "myCollections" – without any change.
- The class to be developed ***Deck***, should follow the structure presented in **ATTACHEMENT A**, but you may add other methods that consider important for solving the problem.
- The class ***Card*** and the java file ***Suit*** enumerators must be used as provided in **ATTACHEMENT B** and cannot change anything in the code of those classes.
- In the methods that you are unable to implement, they should simply include the following line of code:
  throw new UnsupportedOperationException("Method not implemented!");

Generic considerations

- The groups formed in the first practical assignment should be maintained.
- Will be accepted for evaluation, only practical assignments whose implementation does not present any compilation error and with a minimum of perfectly operational functionalities.
- It is expressly forbidden to copy all or part of the code from other sources than the documentation made available by the teachers.
- The assignment must be delivered, by one student of the group, in the e-learning portal, mandatorily, within the deadline set, (at http://virtual.ipb.pt/, choose <Trabalho Pratico 2> on tab <Atividades>, inside the AED area), and in no case can be sent by e-mail.
- The project must be submitted after being compacted by NetBeans itself. To this purpose, in the NetBeans File menu, select Export Project->To ZIP (confirm that the file is left with the extension ".zip").
- The project can only be submitted with a maximum delay of 5 days, and still subtracted a value from the grade for each day of delay.
- No resubmissions will be allowed (when submitting, make sure it is the final version).
- Students may have to defend their work, in person or by videoconference, on a date to be set by the teacher, showing that they have the ability to implement the code, understand it and explain it.
- For further clarification on the work, use the discussion forum created on the http://virtual.ipb.pt/: AED (20/2.4) > Fóruns > Trabalhos Práticos > Trabalho Prático 2.

ATTACHEMENT A

File **Deck.java**

```
import myCollections.ArrayDeque;

public class Deck
{
	private ArrayDeque<Card> contents;

	public Deck() {
		(…)
	}				// end default constructor

	public Deck(int numberOfCards) {
		(…)
	}				// end constructor

	/** Returns the contents of the deck.
	 @return  The contents of the deck. */

	public ArrayDeque<Card> getContents() {
		(…)
	} //

	public boolean nextShufler(…) {
		(…)
	}

	/** Performs a perfect in-shuffle.
	*/
	public void inShuffle(…) {
		(…)
	}				// end inShuffle

	/** Performs a perfect out-shuffle.
	*/
	public void outShuffle(…) {
		(…)
	}				// end outShuffle

	/** Shifts the top card of the deck a given number of positions down in the deck.
	@param position The number of positions to move the top card by.
	 @param messages Whether or not messages should be displayed.
	*/
	public void moveTop(int position, boolean messages) {
		(…)
	} // end shiftFirst


} // end Deck
```

ATTACHEMENT B

File (full) **Suit.java**

```
package aedTrabalho_n2;

public enum Suit {SPADE, CLUB, DIAMOND, HEART}
```

File (full) **Card.java**

```
public class Card{
	private Suit suit;
	private int value = 0;

	public Card(Suit s, int v){
		setSuit(s);
		setValue(v);
	} // end constructor

	// Sets the suit of the card.
	private void setSuit(Suit theSuit){
		suit = theSuit;
	} // end setSuit

	/** Returns the suit of the card as a constant from the Suit enum.
	 @return  The suit of the card.                    */
	public Suit getSuit(){
		return suit;
	} // end getSuit

	// Sets the value / number of the card.
	private void setValue (int theValue){
		value = theValue;
	} // end setValue

	/** Returns the value of the card as an int.
	 @return  The value of the card.                   */
	public int getValue(){
		return value;
	} // end getValue

	public String toString(){
		char suitChar = ' ';    // A character version of the suit
		switch (suit) {
		case SPADE:
			suitChar = 'S';
			break;
		case CLUB:
			suitChar = 'C';
			break;
		case HEART:
			suitChar = 'H';
			break;
		case DIAMOND:
			suitChar = 'D';
			break;
		}
		String valueString = "" + value;

		return "[" + suitChar + " " + valueString + "]";
	} // end toString
} // end Card
```