Object Oriented Design of Card Deck

I am designing classes for a game with card deck. Please review my code.

- Each Player gets Hand of cards
- The Deck can be shuffled and cards are dealt one at a time from the deck and added to the players hands.

Specifically, I am confused with the good method signature for the method `dealCard` in the `Deck` class. I can either pass the player method to the method like - `void dealCard(Player player)` or get the card and add to hand of player - `Card dealCard()` . Which of these implementation is good as per the practices of OOPS? How do I think in these kind of scenarios? Any advice can be helpful

```
package main;
    //Each card has a value and a suite.
public class Card {
    int value;
    SUITE suite;

    public Card(int value, SUITE suite) {
        this.value = value;
        this.suite = suite;
    }

    public int getValue() {
        return value;
    }

    public SUITE getSuite() {
        return suite;
    }

    public void setValue(int value) {
        this.value = value;
    }

    public void setSuite(SUITE suite) {
        this.suite = suite;
    }

    @Override
    public String toString() {
        return "Card{" +
                "value=" + value +
                ", suite=" + suite +
                '}';
    }
}
```

```
package main;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class Deck {
    List<Card> cardDeck;

    public Deck() {
        this.cardDeck = new ArrayList<Card>();
        for(int value = 1 ; value <= 13 ; value++){
            for(SUITE suite : SUITE.values()){
             cardDeck.add(new Card(value,suite));
            }
        }
    }

    @Override
    public String toString() {
        return "Deck{" +
                "cardDeck=" + cardDeck +
                '}';
    }

    public void shuffle(){
        Random rand = new Random();
        //Generate two random numbers between 0 to 51
        for(int i = 0 ; i < 20 ; i ++){
```

```
    public void dealCard(Player player){
       //Get next card and add to hand of the player
        Card removedCard = cardDeck.remove(0);
        player.getHand().add(removedCard);
    }

    public Card dealCard(){
        Card removedCard = cardDeck.remove(0);
        return removedCard;
    }

    //Size of the deck for testing purpose
    public int getSizeOfDeck(){
        return cardDeck.size();
    }
}
```

```
package main;
import java.util.ArrayList;
import java.util.List;

public class Player {
    List<Card> hand;

    public Player() {
        this.hand = new ArrayList<Card>();
    }

    public List<Card> getHand() {
        return hand;
    }

    @Override
    public String toString() {
        return "Player{" +
                "hand=" + hand +
                '}';
    }
}
```

```
package main;

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

```
package main;
public class Main {
    public static void main(String args[]) {
        Deck deck = new Deck();
        System.out.println(deck);
        System.out.println("Size of deck is: "+deck.getSizeOfDeck());
        deck.shuffle();
        System.out.println("Deck after shuffling is "+deck);

        Player player1 = new Player();
        Player player2 = new Player();

        deck.dealCard(player1);
        System.out.println("Size of deck after dealing a card to player is
"+deck.getSizeOfDeck());
        deck.dealCard(player2);
        System.out.println("Size of deck after dealing a card to player is
"+deck.getSizeOfDeck());

        System.out.println("Hand of player 1 is "+player1.getHand());
        System.out.println("Hand of player 2 is "+player2.getHand());

        player1.getHand().add(deck.dealCard());
    }
}
```

java object-oriented playing-cards

edited Apr 25 '17 at 20:04
200_success
120k 14 139 392

asked Apr 25 '17 at 17:31
mc20
431 3 14

4 For reference, it's "suit", not "suite". – cHao Apr 25 '17 at 18:22

2 Very bad shuffle. – paparazzo Apr 26 '17 at 11:37

1 Also, there is a bug in shuffle. You assume the Deck has 52 cards, which can throw an out of bound error if any cards are drawn. You may also combine 2 card sets into the deck, allowing more than 52 cards (like in casinos) – Tezra Apr 26 '17 at 13:29

So much to say, in so little spa**c**e. This could be answered in so many ways, mainly due to why you need to model the cards. Without a proper context, the requirements will change and can thusly change the entire model. So lets pretend you're aiming to build one (or several) card game(s).

Separation of Concerns

There are many important OOP principles, but I'm mainly going to address the Separation of Concerns (or Concern related to Computer Science, as it often is what you need to consider first: *Why do you have the classes/objects, and what is their intended purpose?*

Somewhat simplified one could say that: *An object/class shouldn't need to concern itself with stuff outside of itself*. This could allow for the internal of any class to be replaced, as long as the public API (the public methods to call it) aren't changed. I'm going to try to exemplify this related to the various classes you've provided.

`Card`

In other words, a `Card` shouldn't need to concern itself who might use it, and how they might use it. And the basic method of handling a card should remain somewhat constant.

More specific, it seems kind of strange that a `Card` is able to change its value or suit. What kind of a gaming card is able to do that? If it is the King of Diamonds, then it usually stays that way for ever.

To me, it would be more natural for the cards to have a method defining how the output could look like. For example to choose alternative textual variations: "value: 13, suit: Spade", "King of Spade", "K♠" or even "🂮". The last two examples from: Wikipedia's Playing Cards in Unicode.

Possibly the parent `Card` class should be rather empty, and allow for subclasses to specify the criteria of that card. Like ordinary playing cards, or a Uno deck, or Pokemon deck, and so on.

`Deck`

The `Deck` consists of `Card`s, but other than basic handling of cards, it shouldn't know too much about the cards. And it shouldn't really know anything about a `Player` or a `Game` using the Deck.

And this answers your question on `dealCard()`. It shouldn't know anything about the player, and basically it should only remove a `Card` from the deck, and return it to whomever who might use it as they please.

Other interesting functions for the `Deck` could be re-insert a card into the deck, possibly verifying that the card is legal in this kind of deck. Imaging solitaire games like Spider (using only the suite of Spade), or games using double decks.

`Player`, or maybe `Hand`?

I'm not sure what I would put in a `Player` class, but in the context of a card game, I think I would rather have a `Hand` class to describe my current set of cards.

In this context, it would be natural to add/remove cards to the hand. And to display it in various alternatives. It could possibly have some generic methods for describing the order of a hand compared to another hand.

For subclassing one could also imagine that the hand could calculate the overall value of the hand, for example the Poker value, or whether it contained sets or series, I could imagine have methods for calculating the value of my hand, for example the Poker Hand value, or whether I had any series or sets. And the subclasses could specify which hands were the most valuable related to each other.

Code review

- In general your code looks nice, with proper indentation and brace usage. I would prefer a little more consistency in commenting the methods/classes at a given place related to the definition. You could/should consider using Javadoc, or a given documentation standard for describing your classes.
- A `shuffle()` method which only shuffles 20 cards, by swapping 2 each time, seems like a really lousy way to shuffle a deck. I'm sure you can do better than this. Maybe loop through the deck once, and switch the card placement to somewhere else in the deck?
- Better presentation alternatives would be nice, I think. See comment on `Card` class above.
- A minor detail on the `SUIT` enum. I would order it as `SPADE`, `HEART`, `DIAMOND` and `CLUB`, as that is the common ordering of value of cards in a lot of card games. Not entirely thrilled about the Uppercasing, but not sure what's the common method in Java to denote globals like these.

PPS! Another interesting principle are the SOLID principle, but it might be difficult to wrap your head around if you're very new to programming.

edited Apr 26 '17 at 14:05

Toby Speight
14.4k 1 29 72

answered Apr 25 '17 at 18:33

holroy
10.7k 1 15 50

12 As is often the case with random numbers, it's easy to get wrong. For that reason I would recommend using a known algorithm for shuffling, like for example Fisher-Yates. – Emily L. Apr 25 '17 at 22:43

4 @EmilyL. just call Collections.shuffle – Pete Kirkham Apr 26 '17 at 12:23

3 @PeteKirkham My comment was mostly in reply to `Maybe loop through the deck once, and switch the card placement to somewhere else in the deck?` to point out that it's harder than you think to come up with a good algo. Of course, `Collections.shuffle` . should be used (which probably is Fisher-Yates). – Emily L. Apr 26 '17 at 13:17

To use some appropriate jargon, *dealing* a card should consist of two operations, the first of which is *drawing* a card from the deck. Clarity should ensue by renaming the function from *deal* to *draw*! – Hurkyl Apr 27 '17 at 4:00

Adding to holroy 's great answer..

I think it would be better if the Deck constructor did not instantiate the Cards, but rather accept Cards:

```
public Deck(final List<Card> cards) {
    this.cardDeck = cards;
}
```

Scope of the object rand can be widened, I do not think you need to create a new Random object every time, and this should give you a better distribution. (I am not sure about this last point I made about better distribution, as far as I know it should be the case.)

```
private final Random rand = new Random();
public void shuffle(){
    //Generate two random numbers between 0 to 51
    for(int i = 0 ; i < 20 ; i ++){
        int firstCard = rand.nextInt(52);
        int secondCard = rand.nextInt(52);
        Collections.swap(cardDeck,firstCard,secondCard);
    }
}
```

Instead of blindly exposing the hand of the player as you do below:

```
public List<Card> getHand() {
    return hand;
}
```

I think it would be better to expose functionality you need like:

```
public class Player {
    private final List<Card> hand;

    public Player() {
        this.hand = new ArrayList<Card>();
    }

    public void dealCards(Card... cards) {
        // implementation here..
    }

    public boolean holdsCard(Card card) {
        // implementation here..
    }
}
```

Basically, whenever you see a method chain as follows..

```
player.getHand().add(removedCard);
```

you can think about the design you have.

It is usually a good practice to use private instance variables, where as you seem to have default access on them. (So access modifier = default access.)

With Java, each Random object created on the same thread has a unique seed, even if it's made in the same millisecond as another. – Nic Hartley Apr 26 '17 at 16:53

For your shuffle algorithm, unless you have a solid background in statistics I'd recommend using an established algorithm. Since your algorithm only swaps 20 pairs of cards, you guarantee that at least 12 cards will be remain in their initial positions. In the Fischer-Yates shuffle, every card is given an equal chance to permute, resulting in a fair shuffle.

On a separate note, your SUIT enum should be defined in the card class where it is primarily used. There is no situation where you would need to use SUIT without Card, and this will remove the entanglement of the Card and Main classes.

answered Apr 26 '17 at 3:42

kcazllerraf
41 2

1 I'd even suggest to use the `Collections.suffle()` helper method... – Timothy Truckle Apr 26 '17 at 7:51

Or just don't shuffle. Just select a random entry from the collection when you draw. – Taemyr Apr 26 '17 at 7:54

I am using `SUIT` in `Deck` class to create initial 52 cards. If I move `SUIT` to `CARD` , I guess it wont be possible – mc20 Apr 26 '17 at 8:22

@TimothyTruckle Yes, no point in reinventing the wheel – JollyJoker Apr 26 '17 at 12:06

@mc20 You should move the Suit enum to the Card class while keeping it publicly accessible. Since you are already using Card in Deck it is not an issue to also use Suit in Deck. – kcazllerraf Apr 27 '17 at 1:40

I will also add in...

Think about packaging

A Suit is useless without a Card, so Suit should be in the same package or a sub-package as Card. A Player can use a card, but doesn't need to have a card; So a Player should be in a different package. This is important for permission levels, as you might want to make the card Set methods Protected. Moving Players to a new package would give Deck permission to re-write cards, but not players.

Think "What do you need"/"What does this look like"

I would also make an argument that you don't need a Deck or hand, and that both are just pools of cards. The semantics are in the variable name.

On Shuffle

I would say change

```
for(int i = 0 ; i < 20 ; i ++){
    int firstCard = rand.nextInt(52);
```

To

```
for(int firstCard  = 0 ; firstCard  < cardDeck.length() ; firstCard++){
    int secondCard = rand.nextInt(cardDeck.length());
```

This insures that in the shuffle, each card is touched at least once. Since Deck isn't thread safe, I would also make the Random object a global so that you can make it once at instantiation and reuse it. Also, You should iterate of the length of the deck, not a static 52, as all we really know about the deck is that it contains 0+ cards.

You also need to think about "Do I care about threads?". If 2 Threads share a Deck, They could potentially Duplicate or Drop cards. Making Deck Thread-safe also adds overhead though, so you should only accommodate it as much as you think it matters.

edited Apr 26 '17 at 13:26

answered Apr 25 '17 at 20:23

Tezra
181 1 11

The revised code still doesn't ensure that each card is touched at least once, because over all the iterations there is a very high probability that cards will be multiply selected. Instead it would be better to shuffle from one deck to the next - and then every card would be definitely touched. Implementation is trivial. – Konchog Apr 26 '17 at 6:46

@Konchong Every card is touched at least once. A card is either in it's original position when the index hits that position(in which case it will be touched as firstCard) or it has been moved at an earlier index(in which case it was

The ranks, suits, and cards should all be immutable. I used enums for rank, suit, and card. sample code is here. i used an abstract Cards class which held an array of cards and extended that to a deck class and a hand class.

edited May 23 '17 at 11:33
Community ♦
1

answered Apr 25 '17 at 23:11

Ray Tayek
129 4

What is the reason behind making them immutable and how to do that? – mc20 Apr 26 '17 at 8:23

there really is only one club, one ace, and one aceOfClubs. they should never change. an easy way to do that is to make them enums. if they are enums, it is easy to add other state and/or behaviour. – Ray Tayek Apr 26 '17 at 9:00