The Setup

I want to start doing something a little bit different.

Moving forward, I'll try to do this setup work, in a separate video, when it makes sense to do so.

I'll introduce this code as an optional challenge, for those who really like the independent work.

If you feel like you need extra time with some of the basics, definitely take the challenge.

If you feel like you don't need this review, and aren't interested in the challenge, then feel free to skip this video.

The Setup

I'll review this code in much briefer detail, at the start of the video where I introduce it, and the source will be available as a download for you, in the resources section of that video.

My code will include the use of a record, a nested enum on that record, and the use of List functions you've already been exposed to.

I'll be using ascii codes to display the suits of the cards.

The Setup

In this video, I want to set up a Card class, which will be used to create a deck of playing cards.

I'll be using these cards, and decks of cards, to demonstrate many of the methods on java.util.Collections.

The Card will have three fields:

- a Suit, meaning Club, Diamond, Heart, or Spade.
- a face field, which will be a String, containing either the number of the card, or the face value of the card, Jack, Queen, King or Ace.
- a rank, an integer.

The Setup

The Card should override the toString method, and print the card with the face value (abbreviated, if a face card), the ascii character of the suit, and the rank in parentheses.

I'm including the ascii characters that will print out each suit as a printable character.

CLUB = 9827
DIAMOND = 9830
HEART = 9829
SPADE = 9824

The Setup

The output shown here, shows all the cards in a standard deck of playing cards, sorted by suit and rank.

2♣(0) 3♣(1) 4♣(2) 5♣(3) 6♣(4) 7♣(5) 8♣(6) 9♣(7) 10♣(8) J♣(9) Q♣(10) K♣(11) A♣(12)

2♦(0) 3♦(1) 4♦(2) 5♦(3) 6♦(4) 7♦(5) 8♦(6) 9♦(7) 10♦(8) J♦(9) Q♦(10) K♦(11) A♦(12)

2♥(0) 3♥(1) 4♥(2) 5♥(3) 6♥(4) 7♥(5) 8♥(6) 9♥(7) 10♥(8) J♥(9) Q♥(10) K♥(11) A♥(12)

2♠(0) 3♠(1) 4♠(2) 5♠(3) 6♠(4) 7♠(5) 8♠(6) 9♠(7) 10♠(8) J♠(9) Q♠(10) K♠(11) A♠(12)

The Setup

The card should have the following public static methods to assist anyone using this class:

- getNumericCard which should return an instance of a Card, based on the suit and number passed to it.
- getFaceCard which should return an instance of a Card, based on the suit and abbreviation (J, Q, K, A) passed to it.
- getStandardDeck which should return a list of cards you'd find in a standard deck. See the previous slide for a full set of cards.

The Setup

- printDeck, which should take a description, a list of card, and a row count. This method will print the cards out in the number of rows passed.
- Finally, the card should have an overloaded printDeck method, that will print "Current Deck" as the description, and use 4, as the number of rows to be printed.

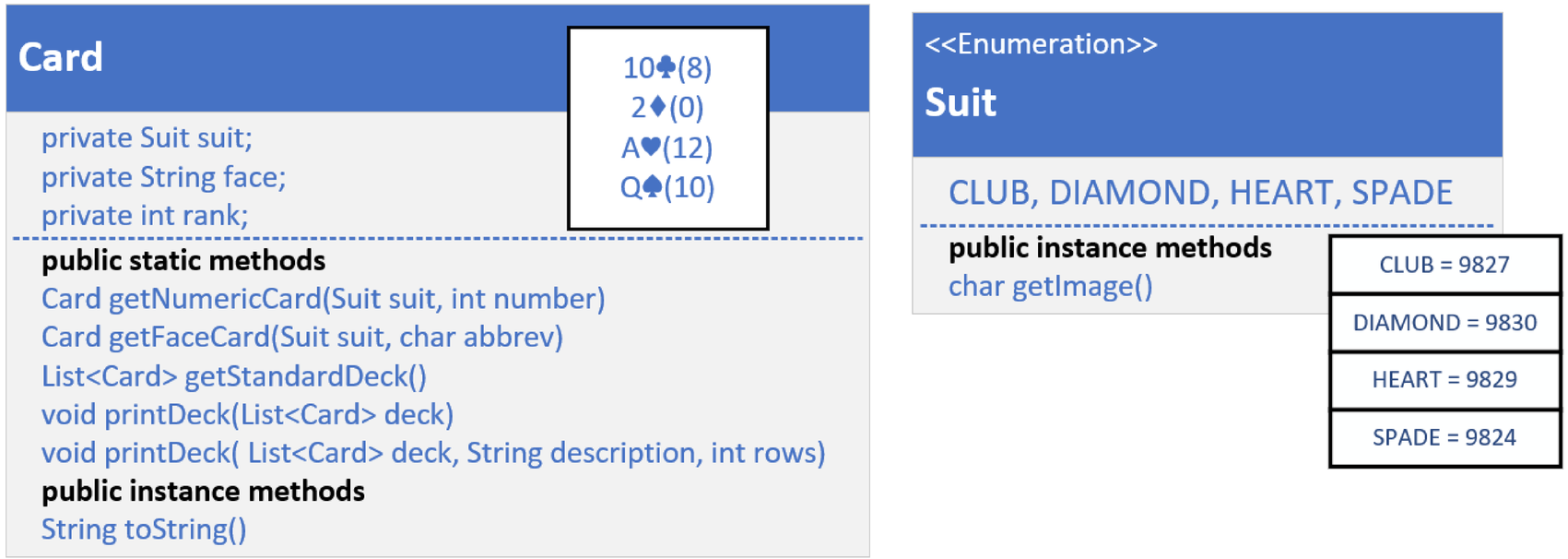
I'll introduce it with a brief overview in the next video, and it will be in the resources section of that video, as well as this one.

The class diagram for my solution

This slide shows my own plan, or the class diagram I'll be coding towards.

I'm going to create a Card class.

In my case I'm just going to make it a record.



The class diagram for my solution

Using a record gives me built in immutability, if all my attributes are simple data types, like primitives and Strings.

Maybe you decided to create a Deck class, to contain your cards, and that's a good idea too.

For the examples ahead, I'm just going to use List as my Deck container.