# Object-Oriented JavaScript: Deck of Cards

Let's implement a deck of playing cards in an object-oriented way with JavaScript. Then we'll add methods for additional functionality such as a **shuffle** operation, a **reset** operation, and a **deal** operation that will return a random card and remove it from the deck.

This type of task is a very common interview question and example given by teachers when discussing object-oriented programming patterns. JavaScript is such a flexible language that even though it is **prototype-based** it can be used in a roughly class-based manner, especially with the introduction of `class` syntax in ES6 which is a syntactic-sugar over the base `prototype`.

Step 1: Create a Deck

To start, we need to think about our requirements. There are 52 cards in a deck, which can have one of 4 suits and one of 13 values. We can store our `deck` as an array. By iterating over each suit and then each value we can populate it as follows

Step 2: Shuffle a deck

For our first method, we want to randomly shuffle the deck. Mike Bostock, the creator of the D3 visualization library, has a fantastic post discussing why you need to use the Fisher-Yates Shuffle for a truly non-biased shuffle.

Using Bostock's code for a generic `shuffle` function, we can add a `shuffle` method to our class as follows:

Note that we use Object Destructuring in the first part of our `shuffle` method. In other words, the following two lines are equivalent:

We also use another shortcut when declaring the `m` and `i` variables to keep them on one line. The following three approaches are equivalent:

Then later on we use Array Destructuring to swap `i` and `m` before finally returning `this`. We need to remember to call our `shuffle` method on our newly created deck instance `deck1`, too, when we want to use it.

### Step 3: Deal a card

Now we need to add a `deal` method that will return one card and remove it from the deck. Fortunately we can use the `pop` method to do exactly this so our code is quite concise for this one!

At the bottom we create a new instance `deck1` of our `Deck` class, call `shuffle()` on it, and output the result. Then we `deal()` one card and if we look at our `deck1` again we can see that the last card has been removed!

### Step 4: Reset

The last step is to add a `reset()` method that will give us a fresh, randomly shuffled deck of cards. If you think about it, we don't really need new code for this, we just need to rearrange what we already have.

Our `reset` method is really just our original code for creating a basic `Deck`. We just need to separate it out from our `constructor` into its own method. As an additional bonus, we can make it so that whenever `Deck` creates a new object instance, that new object will automatically be shuffled.

Here's what the code looks like:

Now each and every time we create a new `Desk` instance it will be randomly shuffled but we can always `reset` it to an ordered configuration. Plus we can `deal` a card or add additional `shuffles` as desired.