# PlayingCard Class

# Create A Subclass of Card

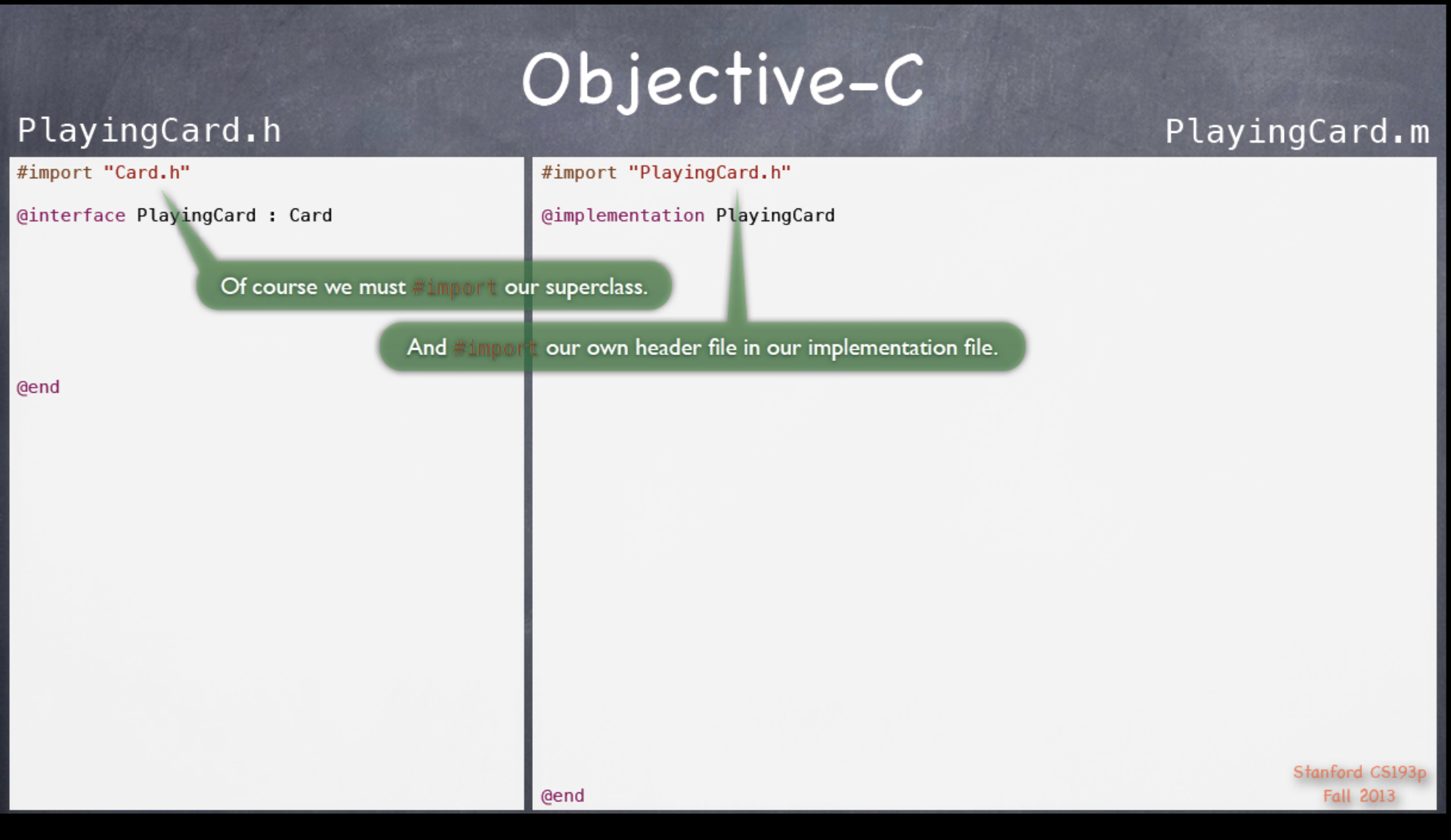
Objective-C

PlayingCard.h

PlayingCard.m

Let's see what it's like to make a subclass of one of our own classes.
In this example, a subclass of `Card` specific to a playing card (e.g. A♠).

# Make Sure To Have Correct Imports



# Define The Properties

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard



@end
```

A PlayingCard has some properties that a vanilla Card doesn't have.
Namely, the PlayingCard's suit and rank.

# More About PlayingCard Properties

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard


@end
```

We'll represent the suit as an NSString that simply contains a single character corresponding to the suit (i.e. one of these characters: ♠♣♥♦).
If this property is nil, it'll mean "suit not set".

We'll represent the rank as an integer from 0 (rank not set) to 13 (a King).

NSUInteger is a typedef for an unsigned integer.

We could just use the C type unsigned int here.
It's mostly a style choice.
Many people like to use NSUInteger and NSInteger in public API and unsigned int and int inside implementation.
But be careful, int is 32 bits, NSInteger might be 64 bits.
If you have an NSInteger that is really big (i.e. > 32 bits worth) it could get truncated if you assign it to an int.
Probably safer to use one or the other everywhere.

# Override The Getter For contents Property

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{

    return [NSString stringWithFormat:@"%d%@", self.rank, self.suit];
}



@end
```

Users of our PlayingCard class might well simply access suit and rank properties directly. But we can also support our superclass's contents property by overriding the getter to return a suitable (no pun intended) NSString.

Even though we are overriding the implementation of the contents method, we are not re-declaring the contents property in our header file. We'll just inherit that declaration from our superclass.

# stringWithFormat Method

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{

    return [NSString stringWithFormat:@"%d%@", self.rank, self.suit];
}



@end
```

The method `stringWithFormat:` is an `NSString` method that's sort of like using the C function `printf` to create the string.

Note we are creating an `NSString` here in a different way than `alloc/init`. We'll see more about "class methods" like `stringWithFormat:` a little later.

Users of our `PlayingCard` class might well simply access `suit` and `rank` properties directly. But we can also support our superclass's `contents` property by overriding the getter to return a suitable (no pun intended) `NSString`.

Even though we are overriding the implementation of the `contents` method, we are not re-declaring the `contents` property in our header file. We'll just inherit that declaration from our superclass.

# Limitation of Current Implementation

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{

    return [NSString stringWithFormat:@"%d%@", self.rank, self.suit];
}



@end
```

Calling the getters of our two properties (rank and suit) on ourself.

But this is a pretty bad representation of the card (e.g., it would say 11♣ instead of J♣ and 1♥ instead of A♥).

# A Fix To The Problem

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}



@end
```

We'll create an `NSArray` of `NSString`s, each of which corresponds to a given rank.
Again, `0` will be "rank not set" (so we'll use `?`).
`11`, `12` and `13` will be `J` `Q` `K` and `1` will be `A`.

Then we'll create our "J♠" string by appending (with the `stringByAppendingString:` method) the suit onto the end of the string we get by looking in the array.

# More About @

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}



@end
```

Notice the @[ ] notation to create an array.

Here's the array-accessing [] notation again (like we used with `self.cards[index]` earlier).

Also note the @" " notation to create a (constant) `NSString`.

All of these notations are converted into normal message-sends by the compiler.
For example, `@[ ... ]` is `[[NSArray alloc] initWithObjects:...]`.
`rankStrings[self.rank]` is `[rankStrings objectAtIndexedSubscript:self.rank]`.

# Modify suit Getter To Return "?" When Not Set

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}
```

This is nice because a "not yet set" rank shows up as ?.

But what about a "not yet set" suit?
Let's override the getter for suit to make a suit of nil return ?.

Yet another nice use for properties versus direct instance variables.

```
- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

# Modify suit Setter To Protect Against Invalid Content

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}



- (void)setSuit:(NSString *)suit
{
    if ([@[@"♥",@"♦",@"♠",@"♣"] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

Let's take this a little further and override the setter for `suit` to have it check to be sure no one tries to set a `suit` to something invalid.

# Sending Message To An NSArray Created by @[]

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}









- (void)setSuit:(NSString *)suit
{
    if ([@[@"♥",@"♦",@"♠",@"♣"] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

Notice that we can embed the array creation as the target of this message send. We're simply sending `containsObject:` to the array created by the `@[]`.

`containsObject:` is an `NSArray` method.

# Problem With Implementing BOTH The Setter And The Getter For A Property

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;


@end
```

But there's a problem here now.
A compiler warning will be generated if we do this.
Why?
Because if you implement BOTH the setter and the getter for a property, then you have to create the instance variable for the property yourself.

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}









- (void)setSuit:(NSString *)suit
{
    if ([@[@"♥",@"♦",@"♠",@"♣"] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

# Must Synthesize The suit Property

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;


@end
```

But there's a problem here now.
A compiler warning will be generated if we do this.
Why?
Because if you implement BOTH the setter and the getter for a property, then you have to create the instance variable for the property yourself.

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
```

Luckily, the compiler can help with this using the `@synthesize` directive.

If you implement only the setter OR the getter (or neither), the compiler adds this `@synthesize` for you.

```
- (void)setSuit:(NSString *)suit
{
    if ([@[@"♥",@"♦",@"♠",@"♣"] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

# Only Access _PropertyName From Setter & Getter

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
```

You should only ever access the instance variable directly ...

```
- (void)setSuit:(NSString *)suit
{
    if ([@[@"♥",@"♦",@"♠",@"♣"] containsObject:suit]) {
        _suit = suit;
    }
}
```

... in the property's setter ...

```
- (NSString *)suit
{
    return _suit ? _suit : @"?";
}
```

... in its getter ...

... or in an initializer (more on this later).

```
@end
```

# Class Methods

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

All of the methods we've seen so far are "instance methods".
They are methods sent to instances of a class.
But it is also possible to create methods that are sent to the class itself.
Usually these are either creation methods (like `alloc` or `stringWithFormat:`) or utility methods.

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter




- (void)setSuit:(NSString *)suit
{
    if ([@[@"♥",@"♦",@"♠",@"♣"] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

# Our First Class Method

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

@end
```

Class methods start with +
Instance methods start with –

Here's an example of a class utility method which returns an NSArray of the NSStrings which are valid suits (e.g. ♠, ♣, ♥, and ♦).

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

+ (NSArray *)validSuits
{
    return
}

- (void)setSuit:(NSString *)suit
{
    if ([@[@"♥",@"♦",@"♠",@"♣"] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

Since a class method is not sent to an instance, we cannot reference our properties in here (since properties represent per-instance storage).

# Move The NSArray To The validSuits Method

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

Here's an example of a class utility method which returns an NSArray of the NSStrings which are valid suits (e.g. ♠, ♣, ♥, and ♦).

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

+ (NSArray *)validSuits
{
    return @[@"♥",@"♦",@"♠",@"♣"];
}

- (void)setSuit:(NSString *)suit
{
    if ([                          containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

We actually already have the array of valid suits, so let's just move that up into our new class method.

# Call validSuits From The Setter

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

+ (NSArray *)validSuits
{
    return @[@"♥",@"♦",@"♠",@"♣"];
}

- (void)setSuit:(NSString *)suit
{
    if ([[PlayingCard validSuits] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

Now let's invoke our new class method here.

See how the name of the class appears in the place you'd normally see a pointer to an instance of an object?

# Notice How A Class Method Is Invoked

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;



@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

+ (NSArray *)validSuits
{
    return @[@"♥",@"♦",@"♠",@"♣"];
}

- (void)setSuit:(NSString *)suit
{
    if ([[PlayingCard validSuits] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

Now let's invoke our new class method here.

See how the name of the class appears in the place you'd normally see a pointer to an instance of an object?

It'd probably be instructive to go back and look at the invocation of the NSString class method stringWithFormat: a few slides ago.

Also, make sure you understand that stringByAppendingString: above is not a class method, it is an instance method.

# You Can Make A Class Method Public

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;


@end
```

The validSuits class method might be useful to users of our PlayingCard class, so let's make it public.

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter

+ (NSArray *)validSuits
{
    return @[@"♥",@"♦",@"♠",@"♣"];
}

- (void)setSuit:(NSString *)suit
{
    if ([[PlayingCard validSuits] containsObject:suit]) {
        _suit = suit;
    }
}

- (NSString *)suit
{
    return _suit ? _suit : @"?";
}

@end
```

# Collapse Methods To Make Room

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;


@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }



@end
```

# Our Second Class Method

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;


@end
```

Let's move our other array (the strings of the ranks) into a class method too.

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings =
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }

+ (NSArray *)rankStrings
{
    return @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
}




@end
```

# Invoke Our New Class Method

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;


@end
```

We'll leave this one private because the public API for the rank is purely numeric.

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = [PlayingCard rankStrings];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }

+ (NSArray *)rankStrings
{
    return @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
}

@end
```

And now let's call that class method.

Note that we are not required to declare this earlier in the file than we use it.

# Our Third Class Method Is Also Public

Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;
+ (NSUInteger)maxRank;

@end
```

But here's another class method that might be good to make public.

So we'll add it to the header file.

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = [PlayingCard rankStrings];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }

+ (NSArray *)rankStrings
{
    return @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
}

+ (NSUInteger)maxRank { return [[self rankStrings] count]-1; }

@end
```

# Use Our New Class Method In The rank Setter

Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;
+ (NSUInteger)maxRank;

@end
```

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = [PlayingCard rankStrings];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }

+ (NSArray *)rankStrings
{
    return @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
}

+ (NSUInteger)maxRank { return [[self rankStrings] count]-1; }

- (void)setRank:(NSUInteger)rank
{
    if (rank <= [PlayingCard maxRank]) {
        _rank = rank;
    }
}

@end
```

And, finally, let's use maxRank inside the setter for the rank @property to make sure the rank is never set to an improper value.

# That's It For PlayingCard

## Objective-C

PlayingCard.h

```
#import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;
@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;
+ (NSUInteger)maxRank;

@end
```

That's it for our PlayingCard. It's a good example of array notation, @synthesize, class methods, and using getters and setters for validation.

PlayingCard.m

```
#import "PlayingCard.h"

@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = [PlayingCard rankStrings];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@synthesize suit = _suit; // because we provide setter AND getter
+ (NSArray *)validSuits { ... }
- (void)setSuit:(NSString *)suit { ... }
- (NSString *)suit { ... }

+ (NSArray *)rankStrings
{
    return @[@"?",@"A",@"2",@"3",...,@"10",@"J",@"Q",@"K"];
}

+ (NSUInteger)maxRank { return [[self rankStrings] count]-1; }

- (void)setRank:(NSUInteger)rank
{
    if (rank <= [PlayingCard maxRank]) {
        _rank = rank;
    }
}

@end
```

# Key References

**Paul Hegarty**

CS193P: iPhone Application Development.

Course Taught at Stanford University, Fall 2013.

Online version available on iTunes U