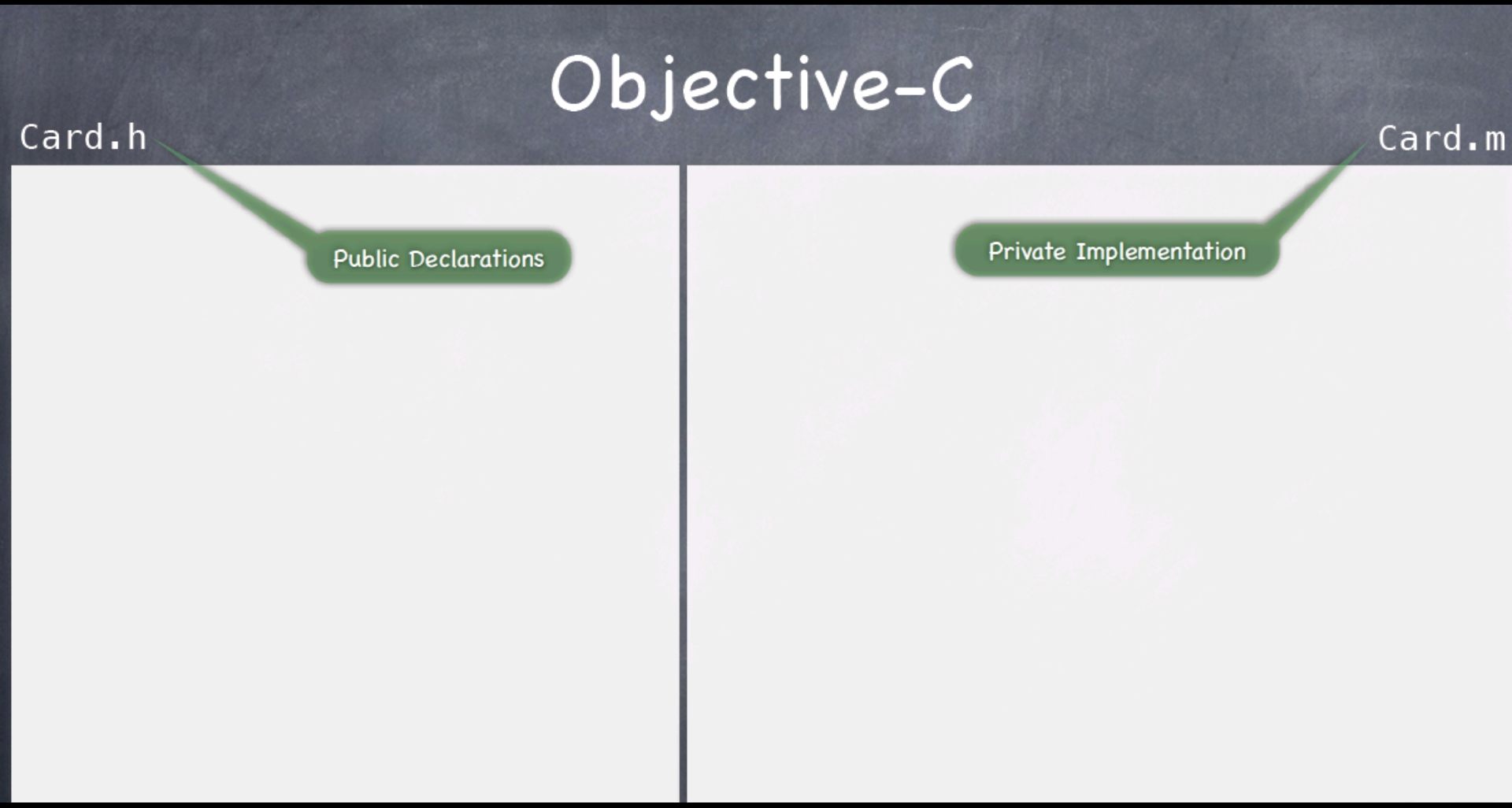
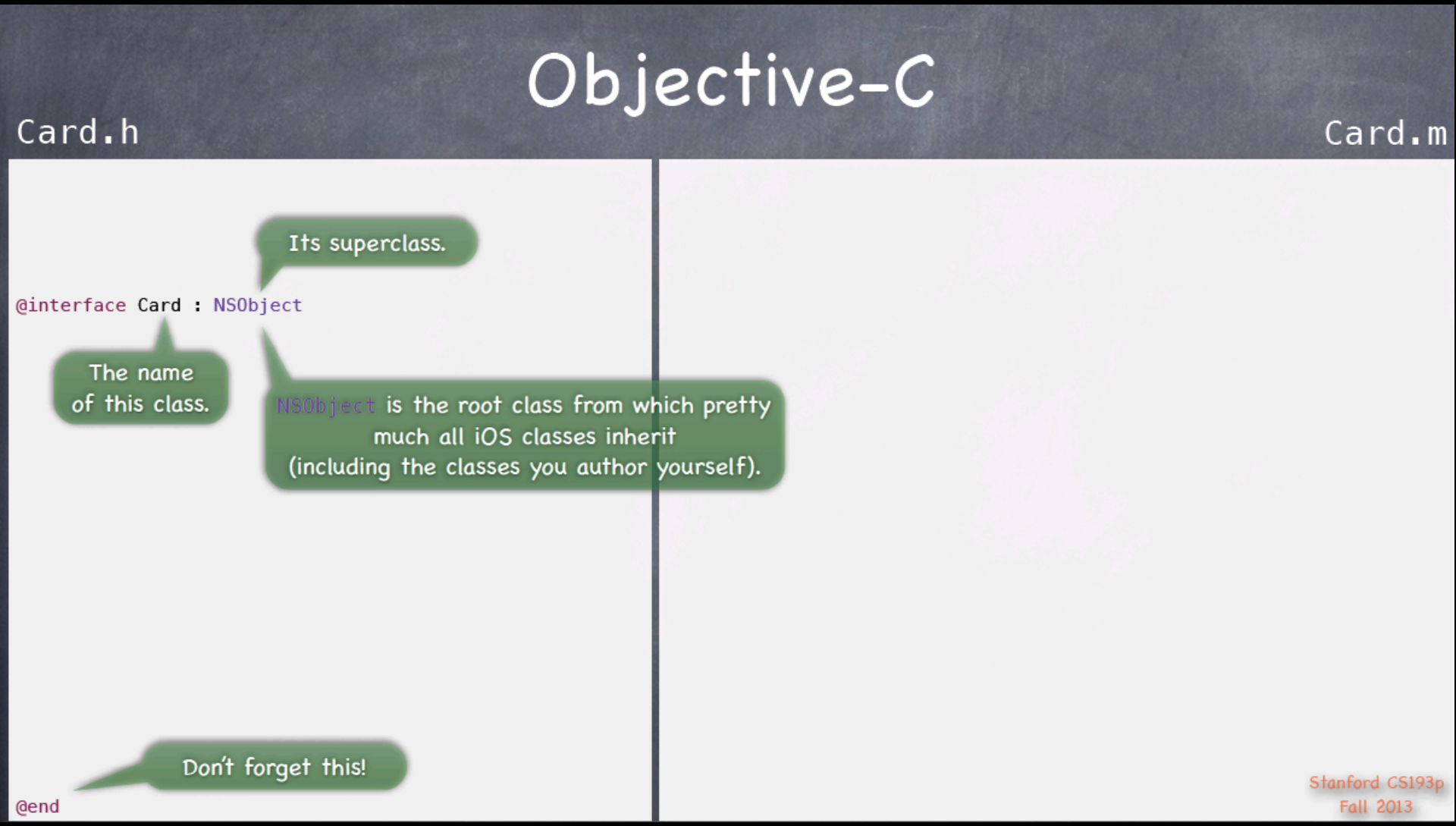
Card Class

2 Files For Every Class



Header File ".h"



Implementation File ".m"

Objective-C

Card.h

```
@interface Card : NSObject

@end
```

Card.m

```
@implementation Card

@end
```

Note, superclass is not specified here.

#import

Objective-C

Card.h

```
#import <Foundation/NSObject.h>

@interface Card : NSObject

@end
```

Superclass's header file.

Card.m

```
@implementation Card

@end
```

Superclass in iOS

Objective-C

Card.h

```
#import <Foundation/Foundation.h>


@interface Card : NSObject


@end
```

Card.m

```
@implementation Card


@end
```

If the superclass is in iOS itself, we import the entire "framework" that includes the superclass.
In this case, Foundation, which contains basic non-UI objects like NSObject.

@import

Objective-C

Card.h

```
@import Foundation;

@interface Card : NSObject

@end
```

In fact, in iOS 7 (only), there is special syntax for importing an entire framework called @import.

Card.m

```
@implementation Card

@end
```

Must Import Our Own Header File

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject



@end
```

Card.m

```
#import "Card.h"



@implementation Card



@end
```

Our own header file must be imported into our implementation file.

Private Declarations

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject



@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card



@end
```

Private declarations can go here.

Our First Property

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong) NSString *contents;



@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card



@end
```

In iOS, we don't access instance variables directly.
Instead, we use an `@property` which declares two methods: a "setter" and a "getter".
It is with those two methods that the `@property`'s instance variable is accessed
(both publicly and privately).

This particular `@property` is a pointer.
Specifically, a pointer to an object whose class is (or inherits from) `NSString`.

ALL objects live in the heap (i.e. are pointed to) in Objective-C!
Thus you would never have a property of type "`NSString`" (rather, "`NSString *`").

Because this `@property` is in this class's header file, it is public.
Its setter and getter can be called from outside this class's `@implementation` block.

strong or weak

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong) NSString *contents;



@end
```

`strong` means:
"keep the object that this property points to
in memory until I set this property to `nil` (zero)
(and it will stay in memory until everyone who has a `strong`
pointer to it sets their property to `nil` too)"

`weak` would mean:
"if no one else has a `strong` pointer to this object,
then you can throw it out of memory
and set this property to `nil`
(this can happen at any time)"

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card



@end
```

atomic or nonatomic

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;



@end
```

nonatomic means:
"access to this property is not thread-safe".
We will always specify this for object pointers in this course.
If you do not, then the compiler will generate locking code that will complicate your code elsewhere.

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card



@end
```

synthesize

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;



@end
```

This is the @property implementation that the compiler generates automatically for you (behind the scenes).
You are welcome to write the setter or getter yourself, but this would only be necessary if you needed to do something in addition to simply setting or getting the value of the property.

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card

@synthesize contents = _contents;

- (NSString *)contents
{
    return _contents;
}

- (void)setContents:(NSString *)contents
{
    _contents = contents;
}



@end
```

This @synthesize is the line of code that actually creates the backing instance variable that is set and gotten.
Notice that by default the backing variable's name is the same as the property's name but with an underbar in front.

Hidden Getter & Setter

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;









@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card
```

Because the compiler takes care of everything you need to implement a property, it's usually only one line of code (the @property declaration) to add one to your class.

```
@end
```

Primitive Properties

Objective-C

Card.h

```
#import <Foundation/Foundation.h>

@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic) BOOL chosen;
@property (nonatomic) BOOL matched;

@end
```

Notice no strong or weak here.
Primitive types are not stored in the heap, so there's no need to specify how the storage for them in the heap is treated.

Let's look at some more properties.
These are not pointers.
They are simple BOOLs.

Properties can be any C type.
That includes int, float, etc., even C structs.

C does not define a "boolean" type.
This BOOL is an Objective-C typedef.
It's values are YES or NO.

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card

@end
```

Behind The Scenes

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic) BOOL chosen;
@property (nonatomic) BOOL matched;



@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card

@synthesize chosen = _chosen;
@synthesize matched = _matched;

- (BOOL)chosen
{
    return _chosen;
}
- (void)setChosen:(BOOL)chosen
{
    _chosen = chosen;
}

- (BOOL)matched
{
    return _matched;
}
- (void)setMatched:(BOOL)matched
{
    _matched = matched;
}

@end
```

Here's what the compiler is doing behind the scenes for these two properties.

Change Getter Name

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;



@end
```

It is actually possible to change the name of the getter that is generated. The only time you'll ever see that done in this class (or anywhere probably) is boolean getters.

This is done simply to make the code "read" a little bit nicer. You'll see this in action later.

Card.m

```
#import "Card.h"

@interface Card()
@end
@implementation Card

@synthesize chosen = _chosen;
@synthesize matched = _matched;

- (BOOL)isChosen
{
    return _chosen;
}
- (void)setChosen:(BOOL)chosen
{
    _chosen = chosen;
}

- (BOOL)isMatched
{
    return _matched;
}
- (void)setMatched:(BOOL)matched
{
    _matched = matched;
}

@end
```

Note change in getter method.

Note change in getter method.

Getter & Setter
Still Hidden

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;



@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card



@end
```

Remember, unless you need to do something besides setting or getting when a property is being set or gotten, the implementation side of this will all happen automatically for you.

Public Method Declaration

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (                          chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(Card *)card;



@end
```

Enough properties for now.
Let's take a look at defining methods.

Here's the declaration of a public method called `match:` which takes one argument (a pointer to a `Card`) and returns an `integer`.

What makes this method public?
Because we've declared it in the header file.

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card



@end
```

Public Method Implementation

Objective-C

Card.h

```
#import <Foundation/Foundation.h>




@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(Card *)card;




@end
```

Here's the declaration of a public method called `match:` which takes one argument (a pointer to a `Card`) and returns an integer.

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card




- (int)match:(Card *)card
{
    int score = 0;

    return score;
}


@end
```

`match:` is going to return a "score" which says how good a match the passed `card` is to the `Card` that is receiving this message. 0 means "no match", higher numbers mean a better match.

Call A Method With [] or .

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(Card *)card;









@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card









- (int)match:(Card *)card
{
    int score = 0;


    if ([card.contents isEqualToString:self.contents]) {
        score = 1;
    }


    return score;
}



@end
```

There's a lot going on here!
For the first time, we are seeing the "calling" side of properties (and methods).

For this example, we'll return 1 if the passed card has the same contents as we do or 0 otherwise (you could imagine more complex scoring).

“.” Notation For Getters and Setters Only

Objective-C

Card.h

```
#import <Foundation/Foundation.h>




@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(Card *)card;









@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card




- (int)match:(Card *)card
{
    int score = 0;


    if ([card.contents isEqualToString:self.contents]) {
        score = 1;
    }


    return score;
}




@end
```

Notice that we are calling the “getter” for the contents @property (both on our self and on the passed card). This calling syntax is called “dot notation.” It’s only for setters and getters.

"[]" Notation For Everything Else

Objective-C

Card.h

```
#import <Foundation/Foundation.h>



@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(Card *)card;



@end
```

Recall that the contents property is an NSString.

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card



- (int)match:(Card *)card
{
    int score = 0;

    if ([card.contents isEqualToString:self.contents]) {
        score = 1;
    }

    return score;
}

@end
```

isEqualToString: is an NSString method which takes another NSString as an argument and returns a BOOL (YES if the 2 strings are the same).

Also, we see the "square bracket" notation we use to send a message to an object.
In this case, the message isEqualToString: is being sent to the NSString returned by the contents getter.

Match Multiple Cards Declaration

Objective-C

Card.h

```
#import <Foundation/Foundation.h>




@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(NSArray *)otherCards;




@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card




- (int)match:(NSArray *)otherCards
{
    int score = 0;


    if ([card.contents isEqualToString:self.contents]) {
        score = 1;
    }


    return score;
}




@end
```

We could make `match:` even more powerful by allowing it to match against multiple cards by passing an array of cards using the `NSArray` class in Foundation.

Match Multiple Cards Implementation

Objective-C

Card.h

```
#import <Foundation/Foundation.h>




@interface Card : NSObject

@property (strong, nonatomic) NSString *contents;

@property (nonatomic, getter=isChosen) BOOL chosen;
@property (nonatomic, getter=isMatched) BOOL matched;

- (int)match:(NSArray *)otherCards;









@end
```

Card.m

```
#import "Card.h"

@interface Card()

@end

@implementation Card

- (int)match:(NSArray *)otherCards
{
    int score = 0;

    for (Card *card in otherCards) {
        if ([card.contents isEqualToString:self.contents]) {
            score = 1;
        }
    }

    return score;
}

@end
```

We'll implement a very simple match scoring system here which is to score 1 point if ANY of the passed otherCards' contents match the receiving Card's contents.
(You could imagine giving more points if multiple cards match.)

Note the for-in looping syntax here.
This is called "fast enumeration."
It works on arrays, dictionaries, etc.

Key References

Paul Hegarty

CS193P: iPhone Application Development.

Course Taught at Stanford University, Fall 2013.

Online version available on iTunes U