Deck Class

Deck Class

Objective-C

Deck.h

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

@interface Deck : NSObject




@end
```

Let's look at another class.
This one represents a deck of cards.

Deck.m

```
#import "Deck.h"

@interface Deck()

@end

@implementation Deck




@end
```

Methods With Multiple Arguments

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;

- (Card *)drawRandomCard;

@end
```

Note that this method has 2 arguments (and returns nothing). It's called "addCard:atTop:".

And this one takes no arguments and returns a Card (i.e. a pointer to an instance of a Card in the heap).

Deck.m

```
#import "Deck.h"

@interface Deck()

@end

@implementation Deck

@end
```

Must Import Card.h

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;


- (Card *)drawRandomCard;

@end
```

We must #import the header file for any class we use in this file (e.g. Card).

Deck.m

```
#import "Deck.h"

@interface Deck()

@end

@implementation Deck




@end
```

Define Methods in Deck.m

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;


- (Card *)drawRandomCard;

@end
```

Deck.m

```
#import "Deck.h"

@interface Deck()

@end

@implementation Deck









- (void)addCard:(Card *)card atTop:(BOOL)atTop
{




}




- (Card *)drawRandomCard { }

@end
```

No Optional Arguments

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;


- (Card *)drawRandomCard;

@end
```

Arguments to methods (like the atTop: argument) are never "optional."

Deck.m

```
#import "Deck.h"

@implementation Deck




- (void)addCard:(Card *)card atTop:(BOOL)atTop
{




}




- (Card *)drawRandomCard { }

@end
```

Can Define A New addCard Method With One Argument

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

Arguments to methods (like the `atTop:` argument) are never "optional."

However, if we want an `addCard:` method without `atTop:`, we can define it separately.

Deck.m

```
#import "Deck.h"

@implementation Deck

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{



}



- (Card *)drawRandomCard { }

@end
```

Implement New AddCard Method

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

Arguments to methods (like the atTop: argument) are never "optional."

However, if we want an addCard: method without atTop:, we can define it separately.

And then simply implement it in terms of the the other method.

Deck.m

```
#import "Deck.h"

@implementation Deck

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{



}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

Need Storage To Hold Cards

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

A deck of cards obviously needs some storage to keep the cards in. We need an `@property` for that. But we don't want it to be public (since it's part of our private, internal implementation).

Deck.m

```
#import "Deck.h"

@interface Deck()

@end

@implementation Deck









- (void)addCard:(Card *)card atTop:(BOOL)atTop
{




}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

Define The Cards Array As Private Property

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

A deck of cards obviously needs some storage to keep the cards in. We need an `@property` for that. But we don't want it to be public (since it's part of our private, internal implementation).

Deck.m

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{



}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

So we put the `@property` declaration we need here in our `@implementation`.

Implement addCard:atTop:

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

Deck.m

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

Now that we have a property to store our cards in, let's take a look at a sample implementation of the addCard:atTop: method.

self.cards is an NSMutableArray ...

... and these are NSMutableArray methods. (insertObject:atIndex: and addObject:).

When Does (cards *) Property Get Allocated?

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

But there's a problem here.
When does the object pointed to by the pointer returned by `self.cards` ever get created?

Deck.m

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

Declaring a `@property` makes space in the instance for the pointer itself, but not does not allocate space in the heap for the object the pointer points to.

Getter For (cards *) Property

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

The place to put this needed heap allocation is in the getter for the cards @property.

Deck.m

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    return _cards;
}


- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

Lazy Instantiation In Getter

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

The place to put this needed heap allocation is in the getter for the cards @property.

Deck.m

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

We'll talk about allocating and initializing objects more later, but here's a simple way to do it.

All properties start out with a value of 0 (called nil for pointers to objects).
So all we need to do is allocate and initialize the object if the pointer to it is nil.
This is called "lazy instantiation".
Now you can start to see the usefulness of a @property.

Now addCard:atTop: Will Work

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

Now the cards property will always at least be an empty mutable array, so this code will always do what we want.

Deck.m

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
    if (atTop) {
        [self.cards insertObject:card atIndex:0];
    } else {
        [self.cards addObject:card];
    }
}

- (void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

- (Card *)drawRandomCard { }

@end
```

Collapse Code To Make Room

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

Let's collapse the code we've written so far to make some space.

Deck.m

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop { ... }
- (void)addCard:(Card *)card { ... }

- (Card *)drawRandomCard
{









}

@end
```

drawRandomCard: Returns A (Card *)

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

Deck.m

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop { ... }
- (void)addCard:(Card *)card { ... }

- (Card *)drawRandomCard
{
    Card *randomCard = nil;

    drawRandomCard simply grabs a card from a
    random spot in our self.cards array.

    return randomCard;
}

@end
```

Implement drawRandomCard:

Objective-C

Deck.h

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

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

Deck.m

```
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop { ... }
- (void)addCard:(Card *)card { ... }

- (Card *)drawRandomCard
{
    Card *randomCard = nil;

    unsigned index = arc4random() % [self.cards count];
    randomCard = self.cards[index];
    [self.cards removeObjectAtIndex:index];

    return randomCard;
}

@end
```

arc4random() returns a random integer.

This is the C modulo operator.

These square brackets actually are the equivalent of sending the message objectAtIndexedSubscript: to the array.

Protect Against An Empty Array

Objective-C

Deck.h

```objc
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;

- (Card *)drawRandomCard;

@end
```

Deck.m

```objc
#import "Deck.h"

@interface Deck()
@property (strong, nonatomic) NSMutableArray *cards; // of Card
@end

@implementation Deck

- (NSMutableArray *)cards
{
    if (!_cards) _cards = [[NSMutableArray alloc] init];
    return _cards;
}

- (void)addCard:(Card *)card atTop:(BOOL)atTop { ... }
- (void)addCard:(Card *)card { ... }

- (Card *)drawRandomCard
{
    Card *randomCard = nil;

    if ([self.cards count]) {
        unsigned index = arc4random() % [self.cards count];
        randomCard = self.cards[index];
        [self.cards removeObjectAtIndex:index];
    }

    return randomCard;
}

@end
```

Calling `objectAtIndexedSubscript:` with an argument of zero on an empty array will **crash** (array index out of bounds)!

So let's protect against that case.

Key References

Paul Hegarty

CS193P: iPhone Application Development.

Course Taught at Stanford University, Fall 2013.

Online version available on iTunes U