Deck Class
Deck Class

Objective-C

Deck.h

#import <Foundation/Foundation.h>

@interface Deck : NSObject
@end

Deck.m

#import "Deck.h"

@interface Deck()
@end

@implementation Deck

@end

Let's look at another class. This one represents a deck of cards.
Methods With Multiple Arguments

Objective-C

Deck.h

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

@interface Deck : NSObject

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

-(Card *)drawRandomCard;
@end
```

Deck.m

```c
#import "Deck.h"

@interface Deck()
@end

@implementation Deck
@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).
Must Import Card.h

Deck.h

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

@interface Deck : NSObject
-
(void)addCard:(Card *)card atTop:(BOOL)atTop;
-
(Card *)drawRandomCard;
@end
```

Deck.m

```c
#import "Deck.h"

@interface Deck()
@end

@implementation Deck
@end
```
Define Methods in Deck.m

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

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"

@implementation Deck

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

-(Card *)drawRandomCard {
}
@end

Arguments to methods (like the atTop: argument) are never "optional."
Can Define A New `addCard` Method With One Argument

```objective-c
@interface Deck : NSObject
- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;
- (Card *)drawRandomCard;
@end
```

```objective-c
#import "Deck.h"

@implementation Deck

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

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

```objective-c
- (void)addCard:(Card *)card atTop:(BOOL)atTop
{
}
```
Implement New AddCard Method

Deck.h

```objective-c
#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

```objective-c
#import "Deck.h"

@implementation Deck

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

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

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

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).
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 atIndex:(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 atIndex:(BOOL)atTop
{
}

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

So we put the @property declaration we need here in our @implementation.
Implement `addCard:atTop:`:

---

**Objective-C**

```objective-c
@interface Deck : NSObject

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

@implementation Deck

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

@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.
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

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

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

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

### Deck.h

```c
#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

```c
#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
```

The place to put this needed heap allocation is in the **getter** for the cards **property**.
Lazy Instantiation In Getter

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

Objective-C

Deck.m
#import "Deck.h"
@interface Deck()<strong, nonatomic> NSMutatableArray *cards; // of Card
@end

@implementation Deck
-(NSMutatableArray *)cards
{  
    if (!_cards) _cards = [[NSMutatableArray alloc] init];
    return _cards;
}
-(void)addCard:(Card *)card atTop:(BOOL)atTop
{  
    if (atTop) {
        [self.cards insert:card atIndex:0];
    } else {
        [self.cards insert:card atIndex:_cards.count];
    }
}
-(void)addCard{ }
-(Card *)drawRandomCard { }
@end

Stanford CS193p
Fall 2013
Now addCard:atTop: Will Work

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

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

Let's collapse the code we've written so far to make some space.
**drawRandomCard:** Returns A *(Card *)
Implement `drawRandomCard`:

```objective-c
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject
- (void)addCard:(Card *)card atTop:(BOOL)atTop;
- (void)addCard:(Card *)card;
- (Card *)drawRandomCard;
@end

#import "Deck.h"

@interface Deck
@property (strong, nonatomic) NS MutableArray *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
{
    int randomCardIndex = arc4random() % [self.cards count];
    Card *randomCard = self.cards[randomCardIndex];
    [self.cards removeObjectAtIndex:randomCardIndex];
    return randomCard;
}
@end
```

This is the C modulo operator.

These square brackets actually are the equivalent of sending the message `objectAtIndex: to the array.`
Protect Against An Empty Array

Deck.h

```objective-c
#import <Foundation/Foundation.h>
#import "Card.h"

@interface Deck : NSObject

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

-(Card *)drawRandomCard;
@end
```

Deck.m

```objective-c
#import "Deck.h"

@interface Deck()<NSObject

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

@interface Deck

@end

@implementation Deck

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

-(void)addCard:(Card *)card atIndex:(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.
Paul Hegarty

CS193P: iPhone Application Development.
Course Taught at Stanford University, Fall 2013.
Online version available on iTunes U