

Objects, Methods, Classes, and Properties

A primer to Object Oriented programming in
Objective-C

Why object oriented?

- Nearly all modern software is object oriented
- Helps organize code
- It's fun!

What is an object?

- Objects are the **nouns** of OOP
- “I want to set the **image** of that **view** to **this image**”



Visual Element or Concept

- Set the **color** of that **view** to **red**
- Set the **name** of that **person** to Brutus
- The **address** of that **restaurant** is **240 E 13th St.**
- The **color** of this **bottle** is **green**

Objects allow us to do 2 things

- Hold other objects
 - This **person's first name** is **Winston**
- Do stuff
 - This **view** should scroll to the **200** pixel mark

Methods


- Methods are the **verbs** of OOP
- “**Set** the scroll location to 120”
- “**Push** this view onto the stack”
- “**Animate** this to center screen”

Properties

- Properties are data about an object
- They are like a local variable, except they can be used anywhere by that object

An object you might create

Chat 'n Chew Restaurant

Property Name	Type	Value
name	NSString*	@ "Chat 'n Chew"
cuisine	NSString*	@ "American"
image	UIImage*	

Restaurant Methods

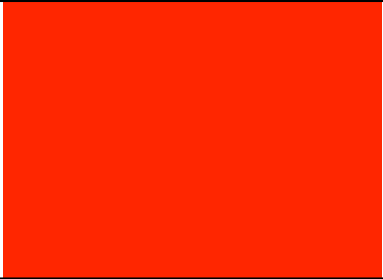
- The verbs of “What do we need to do with a restaurant”
- – (float) calculatePriceOfDinner
- – (void) addReview:(Review*)review
- – (void) deleteReview:(Review*)review

void and nil

- Both mean “nothing”
- void is used in place of a **Type**, when a method returns nothing
- nil is used in place of an **Object**, so that a variable can point at nothing

Another example object

A Label

Property Name	Type	Value
text	NSString*	@ "One Fine Evening"
textColor	UIColor*	
fontSize	float	18

Example Code - Restaurant

```
Restaurant* chatNChew = [Restaurant new];
```

Create a new restaurant
and set it to the
chatNChew variable

```
titleLabel.text = chatNChew.name;
```

Ask for the name property
of chatNChew, set the title
label on the view

```
subtitleLabel.text = chatNChew.cuisine;
```

Ask for the cuisine property
of chatNChew, set the
subtitle label on the view

```
float priceOfDinner = [chatNChew calculatePriceOfDinner];
```

Call the
calculatePriceOfDinner
method on chatNChew

```
priceLabel.text = [NSString stringWithFormat:@"%$.2f"];
```

Set the price on the
priceLabel

Almost everything is an object!

- Almost?
 - int, float, BOOL, double, etc. are all basic data types
 - C Structures are also supported, but used less frequently
- Data types and C-Structs **only hold data**: they can't have methods.

Challenge: Turn this into an object the computer understands





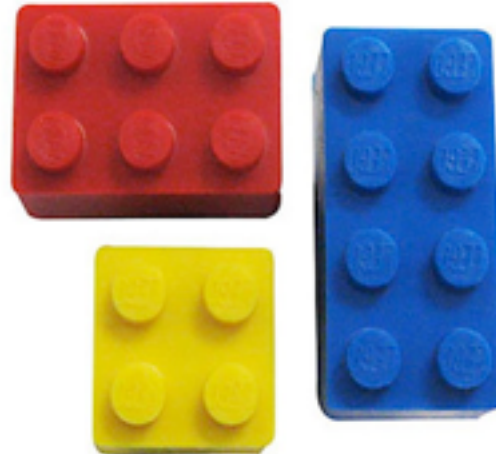
Tips

- Use properties to describe the object
 - What physical properties does it possess?
 - Pretend you are describing it to someone who has never seen one.
- Use methods to describe what the object can do
 - What other objects might need to interact with this object?

Classes

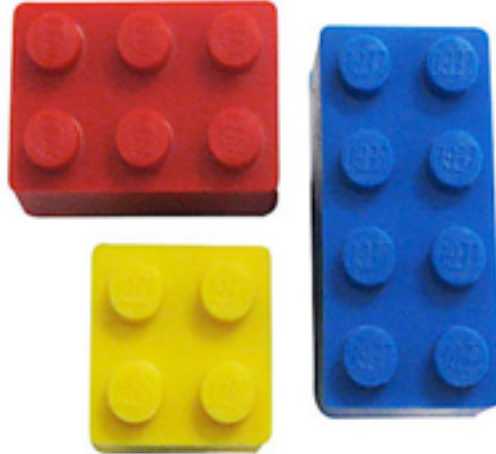
- Q: How do we write an object in code?
- A: You don't! You **create** it when the application runs!

Example



- We write a **Class** to define the properties and methods that objects of that **kind** of class will have.
- Classes are the blueprints of objects

How we might create 3 bricks



```
Brick* yellowBrick = [Brick new];  
yellowBrick.color = [UIColor yellowColor];  
yellowBrick.width = 2;  
yellowBrick.height = 2;  
[yellowBrick placeOnBrick:ground];
```

```
Brick* redBrick = [Brick new];  
redBrick.color = [UIColor redColor];  
redBrick.width = 2;  
redBrick.height = 3;  
[redBrick placeOnBrick:yellowBrick];
```

```
Brick* blueBrick = [Brick new];  
blueBrick.color = [UIColor blueColor];  
blueBrick.width = 2;  
blueBrick.height = 4;  
[blueBrick placeOnBrick:redBrick];
```

**Notice how we both
set properties and
modify state by calling
methods.**

NSString

- We want to represent this:
- *“I had a delicious sandwich for lunch”*
- What other things might we need to do know about this text?
- What might we need to do to this text?

NSString

NSString is a Class that represents strings.

```
NSString* whatIAte = @"I had a delicious sandwich for  
lunch";
```

is a shortcut to creating a new string object.

```
int stringLength = whatIAte.length;
```

```
NSString* aBetterLunch = [whatIAte  
stringByAppendingString:@" and a macaron"];
```

UIColor

- UIColor is a class that represents a color
- You can create a

What's with the *?

- It simply means “pointer”
- You don't need to worry about it much
- But you need to make sure you use it whenever you are talking about that type, but not **to** that type

Use *

```
NSString* someText = @"ss";
```

```
- (NSString*)stringByAppendingString:(NSString*)string
```

Don't use *

```
[NSArray new];
```

Class Methods

- You can talk **to** a type?
- It's like asking the blueprint
- `[NSString new]` is a great example: it returns an `NSString*`
- `[UIColor redColor]` is another, it returns a `UIColor*` set to red
- “String, make me a new instance of you”
- There is only ever **one** of each **class**
- There can be **unlimited objects** that are kinds of that class

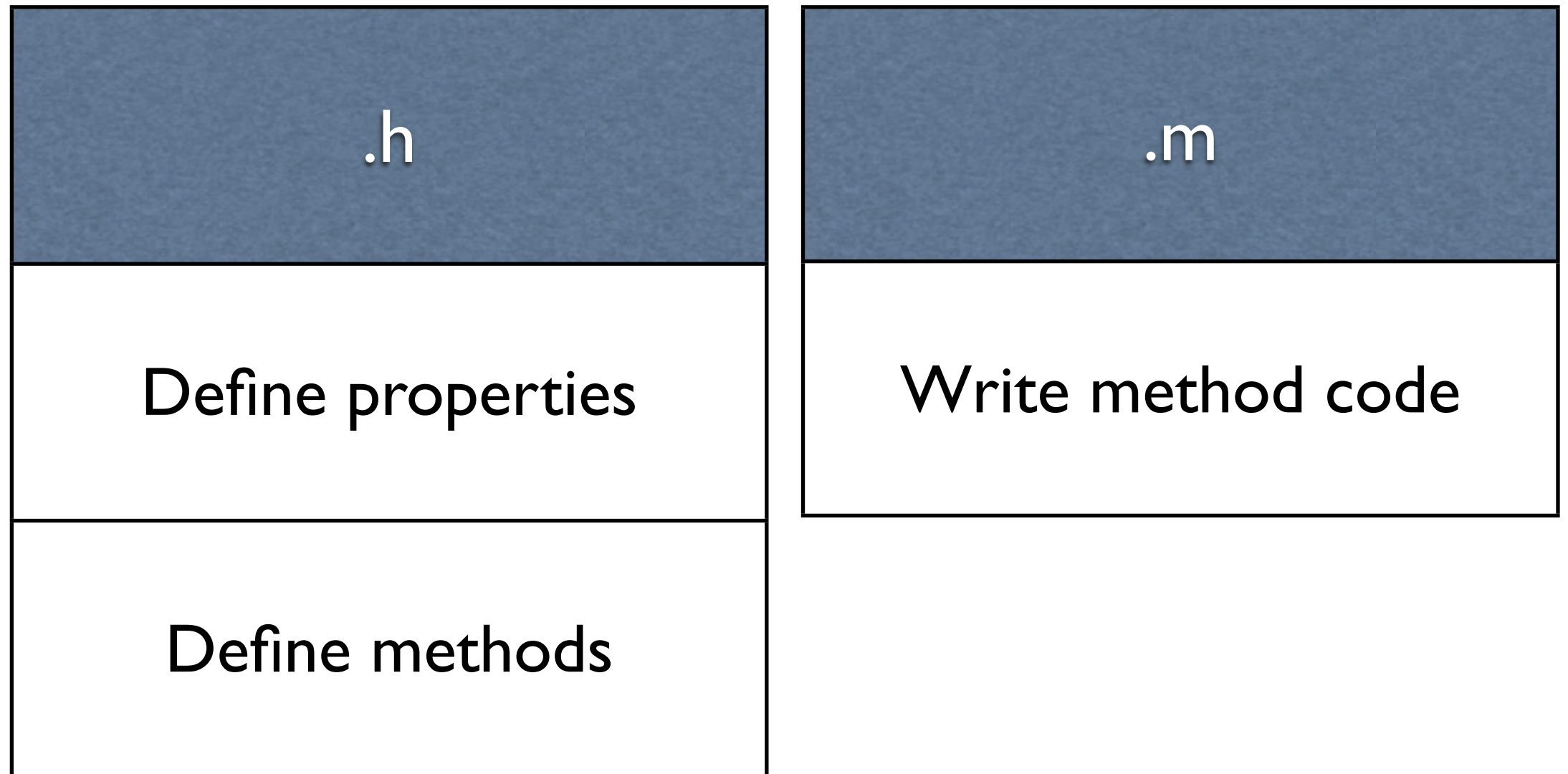
Writing a Class

- Don't focus on the values, focus on the types
- Write properties to hold data
- Write methods to do stuff

.h and .m files

- Objective-C has both .h files, called **header** or **interface** files, and .m files, called **implementation** files.
- Header files are the description of your class to other classes
- Implementation files are the execution of your class

Brick Class



Brick.h

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

```
@interface Brick : NSObject Interface and superclass
```

```
@property UIColor* color;
```

```
@property int width; Properties
```

```
@property int height;
```

- (void)cutInHalf;
- (int)numberOfPegs; **Method declarations**
- (void)placeOnBrick:(Brick*)otherBrick;

```
@end End the interface (no curly braces)
```

Brick.m

@implementation Brick **Implementation**

```
- (void)cutInHalf
{
    self.height = self.height / 2;
}

- (int)numberOfPegs
{
    return self.height * self.width;
}

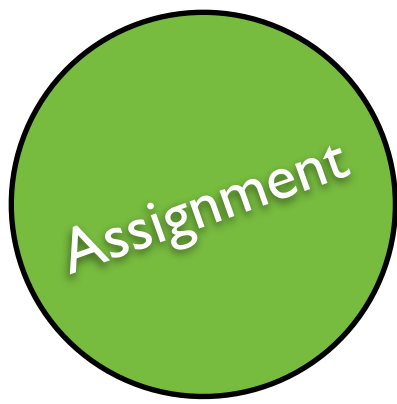
- (void)placeOnBrick:(Brick *)otherBrick
{
    //Here we would do whatever is needed to attach bricks together
}
```

Implement methods

@end **End the implementation**

Tips

- When accessing a property from within the same class, use **self.propertyName**
- When accessing a property from another class, use **variableName.propertyName**
- When accessing a method from within the same class, use **[self methodName]**
- When accessing a method from another class, use **[variableName methodName]**
- Classes should start with a capital letter
- Methods and properties should start with a lowercase letter

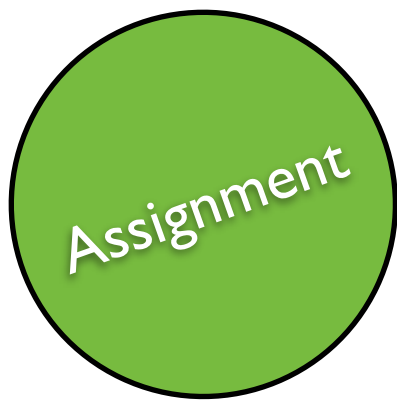


Create the Restaurant Class

- We're going to create a new class, called Restaurant
- We need it to have the following properties:
 - The title of the restaurant
 - The type of cuisine
 - The price of an entree
 - The price of an appetizer
 - The price of dessert
 - The price of a bottle of wine

Create the Restaurant Class

- Don't worry about methods yet, let's just get some properties created.

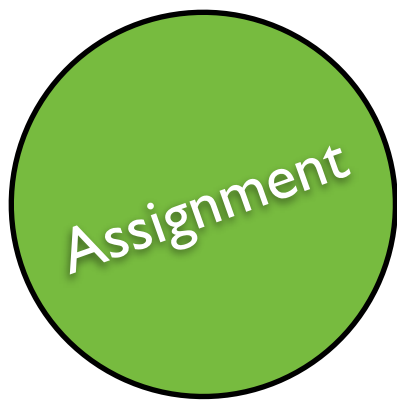


Use the Restaurant class

- In DetailViewController's configureView method:
 - Create a 3 new instances of your Restaurant class
 - Set up properties for your favorite restaurants
 - Choose one of those restaurants and make it's title display on the view

Tips

- Add `#import "Restaurant.h"` line to the top of the `DetailViewController.h` file
- This lets the `DetailViewController` know about the `Restaurant`
- Make sure you use the `*` correctly
- Ensure you choose the correct data types (`int/float`) or classes (`NSString*`) for your properties

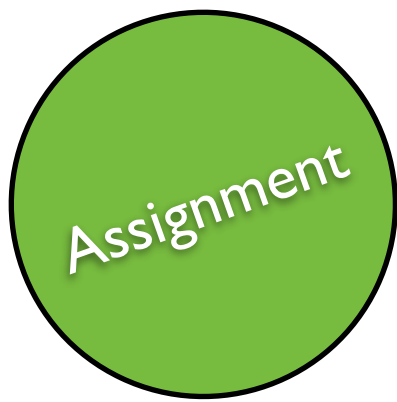


Add a method

- **Copy** `-(float)priceOfDinnerForNumberOfGuests:(int)numberOfGuests` to your **Restaurant class**
- You can delete that method from your **DetailViewController**
- Change the **configureView** method to call the method in your restaurant
- Set the **detailDescriptionLabel** to the output from your new method

IBOutlets

- IBOutlets are just (special) properties for your view controllers
- They allow you to make the connection between the view in the storyboard and your app
- You make the connection to the view, which might be an instance of UILabel, UIView, UIImageView, etc.
- Once you have that connection, you can set whatever properties you want on that view.



Adding additional IBOutlets

- Create IBOutlets for labels corresponding to:
 - Title
 - CuisineType
- (Optional) Add a BOOL to your view called `acceptsCreditCards`, and add a credit card logo to the view, with its hidden property set to your restaurant's value.
- (Optional) Create additional IBOutlets and explore how to set other properties on the views.

Turn in

- Ensure you have created 3 restaurants in your DetailViewController
 - Hook up one of them to the view using IBOutlets
 - Try switching which of your 3 restaurants is being displayed.
 - Can you set up your code so that switching this only requires changing 1 line of code?
-
- Email your zipped assignment to stanciod@newschool.edu

Next week:

- IBActions (Special methods that allow you to hook up buttons and other controls)
- Model-View-Controller
- More User Interface exploration
- Dealing with images
- What do you want to learn about?

Advanced Tasks (Optional)

- Add a UIStepper to allow you to change the number of guests
- Look into IBActions to make this happen
- Add any additional properties and methods to your Restaurant that support your views