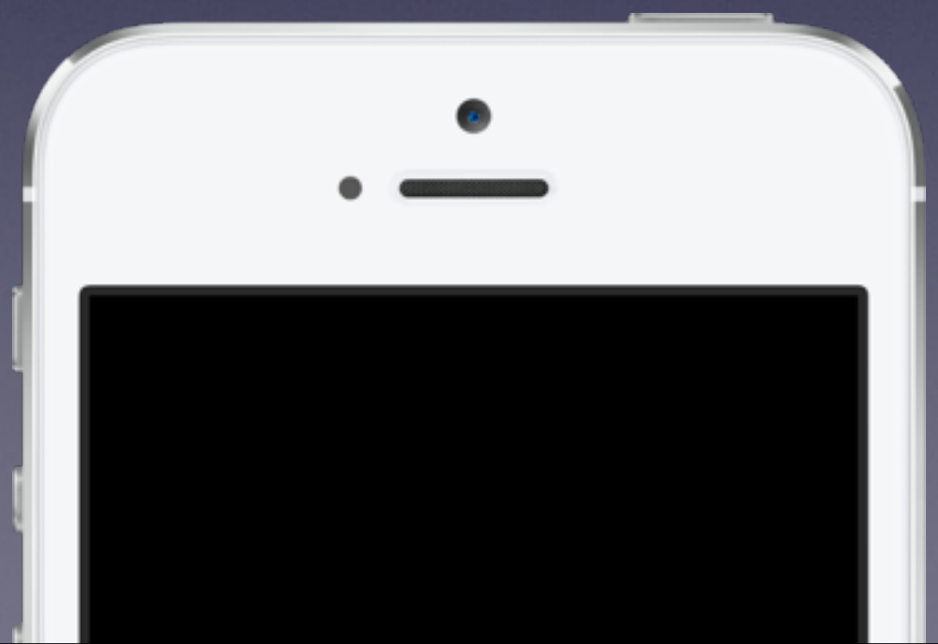
COMS W3101: Programming for iOS

Michael Vitrano



Course Goals

- Walk away with the toolset to build real world apps
- Focus on a foundation in Objective-C and core iOS concepts and frameworks
- Exposure to real-world problem domains and app categories
- A resource for transitioning from academic coding to professional coding

Course Prerequisites

- Fluency in at least one programming language
- Strong understanding of Object-Oriented programming
- Familiarity with Model-View-Controller architecture

Course Structure

- Each class is going to be a mixture of lecture and demo
- Will be posted to the course Github page
    - http://columbia-w3103-ios.github.io/
- Will focus on the building blocks of an iOS app

- One course long project
    - We're going to build a note taking app
    - There will be a set of features that your app must have
    - Over the course of the semester you'll gain the skills you need

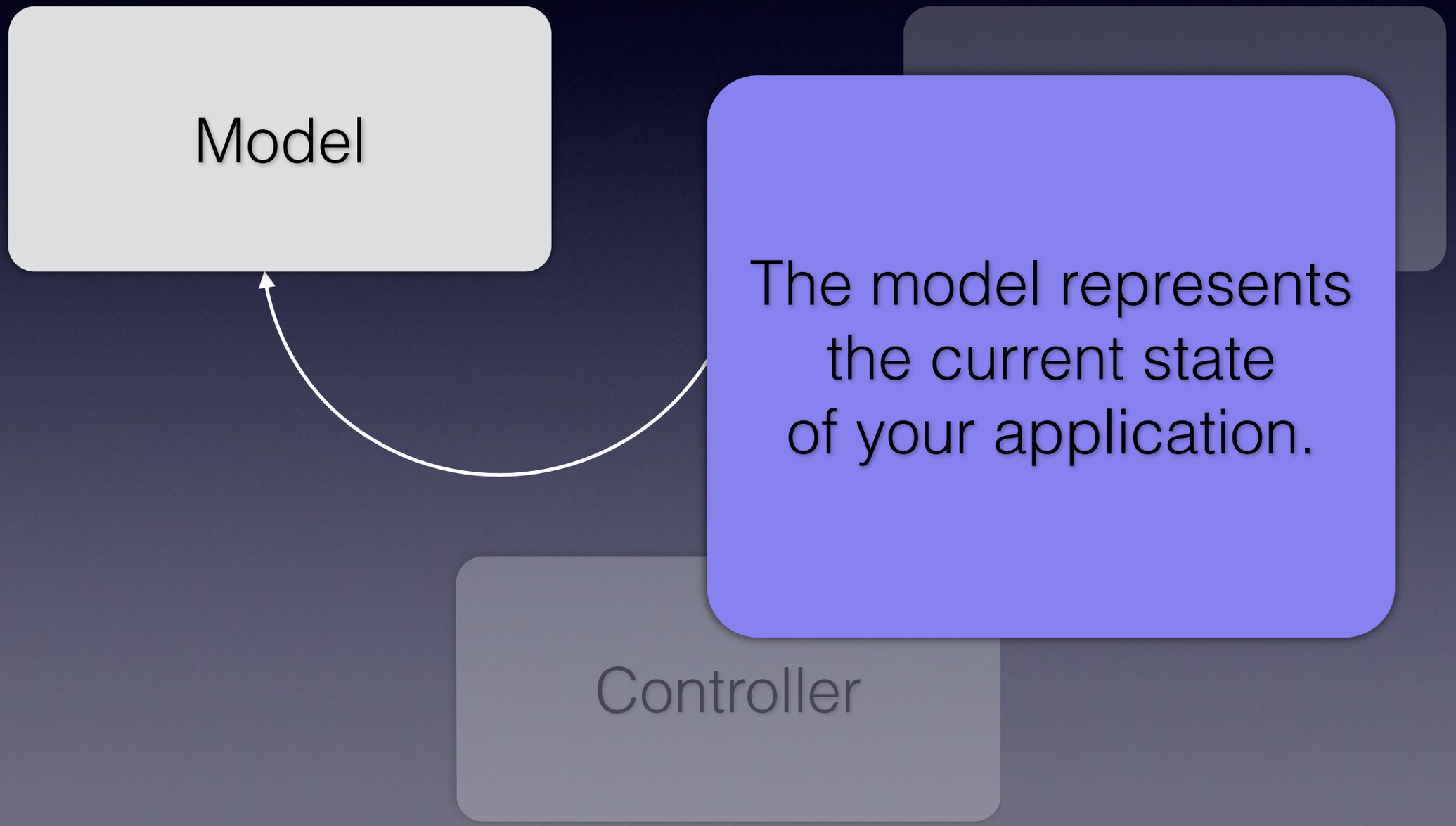
Model-View-Controller

Model

View

Controller

Model-View-Controller



Model-View-Controller

The view is how your application displays its state to the user.

View

Controller

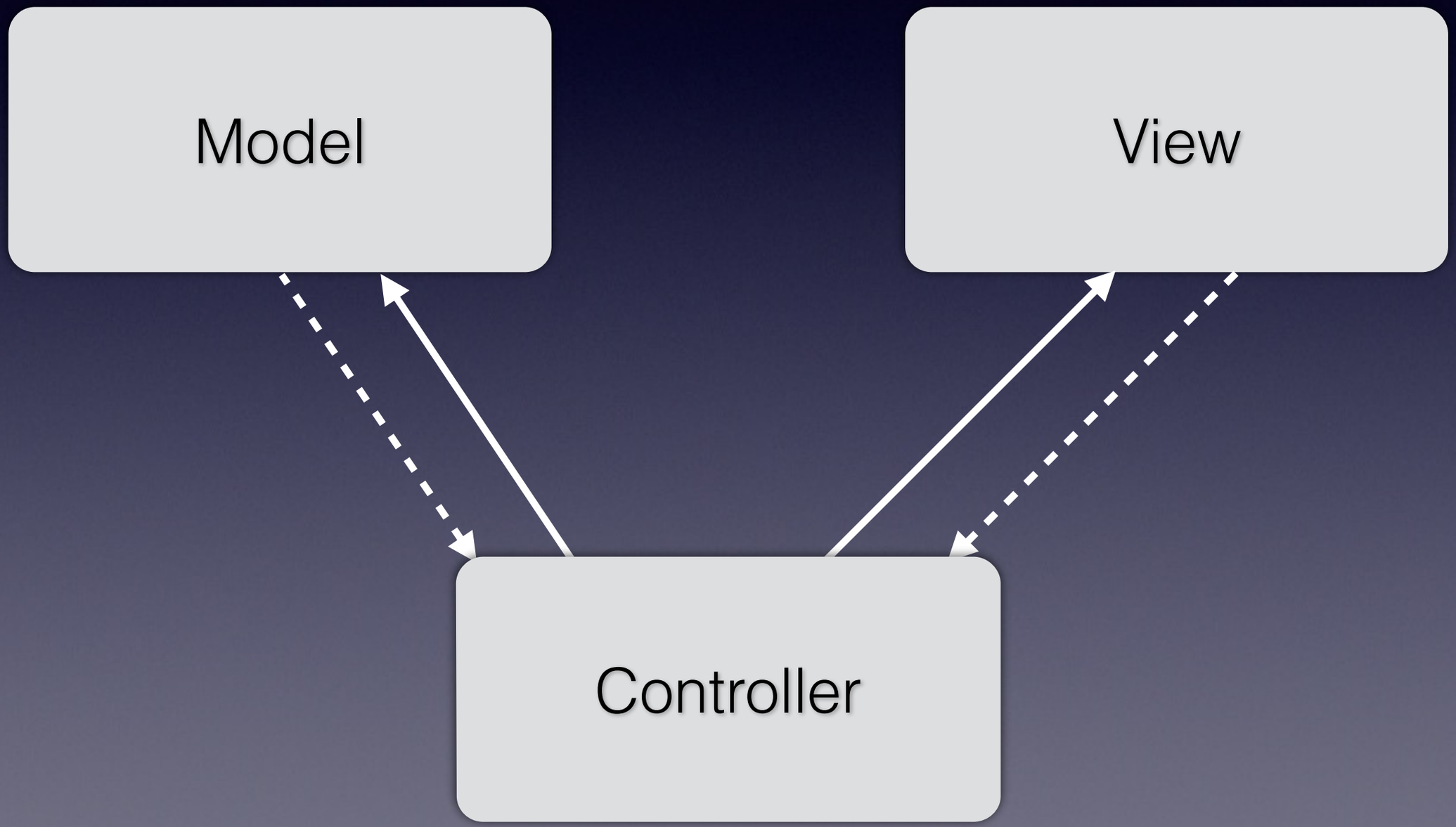
Model-View-Controller

Model

The controller mediates the communication between the model and the view.

Controller

Model-View-Controller



Model-View-Controller

Controllers need to have direct knowledge about the view.

View

Controller

Model-View-Controller

View

Controller

Views communicate with controllers via Delegate and Target-Action Patterns.

Model-View-Controller

Model

Controllers must have direct knowledge about the model.

Vie

Controller

Model-View-Controller

Model

Vie

Models broadcast information about changes with NSNotificationCenter.

Controller

MSVPowerPlant.h — Edited

MSVPowerPlant.h › @interface MSVPowerPlant

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

@interface MSVPowerPlant : NSObject

// This is the basic declaration of a class called MSVPowerPlant
// it inherits from NSObject. Objects are defined in header files (.h)

@end
```

Counterparts › MSVPowerPlant.m › @implementation MSVPowerPlant

```objc
#import "MSVPowerPlant.h"

@implementation MSVPowerPlant

// For every class defined, there is also an implementation
// which lives in the implementation file (.m)

@end
```

MSVPowerPlant.h — Edited

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

@interface MSVPowerPlant : NSObject

// Your class's public interface is defined here, in the header

@end
```

```objc
#import "MSVPowerPlant.h"

@interface MSVPowerPlant ()

// Your class's internal interface is declared inside of the
// implementation file in a class extension

@end

@implementation MSVPowerPlant

// The implementation of your public and private methods go
// here in the implementation file

@end
```

MSVPowerPlant.h

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

// We need to let the compiler know that there exists
// a class called MSVCity that we are going to use in our
// sendPowerToCity:amount: method. We do that with the @class
// directive
@class MSVCity;

@interface MSVPowerPlant : NSObject

// We define a method like this:

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;

// This method takes a city object and an amount of power
// to send it. It returns void.

@end
```

MSVPowerPlant.m

```objc
#import "MSVPowerPlant.h"

// We need to import the MSVCity header so that
// we can know the a MSVCity object's interface
#import "MSVCity.h"

@interface MSVPowerPlant ()

@end

@implementation MSVPowerPlant

// We implement methods like this:

- (void)sendPowerToCity:(MSVCity *)city amount:(double)powerInWatts
{
    // We call methods like this:
    [city addPower:powerInWatts];
}

@end
```

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

@class MSVCity;

@interface MSVPowerPlant : NSObject

// Properties are how we define storage for an object to
// keep state in. When declaring a property, the compiler automatically
// generates a accessor and setter for us as well as an instance variable
// to store that information in.

@property (nonatomic) double currentPower;

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;


@end
```

```objc
#import "MSVPowerPlant.h"
#import "MSVCity.h"

@interface MSVPowerPlant ()

@end

@implementation MSVPowerPlant

- (void)sendPowerToCity:(MSVCity *)city amount:(double)powerInWatts
{
    // The accessor method is just the name of the property. In ObjC
    // we do not prepend 'get' to the beginning of our accessors
    double currentPower = [self currentPower];

    if (currentPower >= powerInWatts) {

        // The setter method is the name of the property prepended by
        // 'set'

        [self setCurrentPower:(currentPower - powerInWatts)];

        [city addPower:powerInWatts];
    }
}

@end
```

MSVPowerPlant.h

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

@class MSVCity;

@interface MSVPowerPlant : NSObject

// Some properties should not be writable to outsiders.
// We restrict the writablitity of a property using 'readonly'.
// This means that only the accessor method is available outside of the class.

@property (nonatomic, readonly) double currentPower;

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;


@end
```

Counterparts › MSVPowerPlant.m

```objc
#import "MSVPowerPlant.h"
#import "MSVCity.h"

@interface MSVPowerPlant ()

// Since we still need to set the current power when sending power
// to a city, we need to have a writable property interal to the class.
// We do this by re-defining the property as writable inside the class extension

@property (nonatomic) double currentPower;

@end

@implementation MSVPowerPlant

- (void)sendPowerToCity:(MSVCity *)city amount:(double)powerInWatts
{
    // The accessor method is just the name of the property. In ObjC
    // we do not prepend 'get' to the beginning of our accessors
    double currentPower = [self currentPower];

    if (currentPower >= powerInWatts) {

        // The setter method is the name of the property prepended by
        // 'set'

        [self setCurrentPower:(currentPower - powerInWatts)];

        [city addPower:powerInWatts];
    }
}

@end
```

MSVPowerPlant.h — Edited

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

@class MSVCity;

@interface MSVPowerPlant : NSObject

@property (nonatomic, readonly) double currentPower;

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;


@end
```

```objc
#import "MSVPowerPlant.h"
#import "MSVCity.h"

@interface MSVPowerPlant ()

@property (nonatomic) double currentPower;

@end

@implementation MSVPowerPlant

- (void)sendPowerToCity:(MSVCity *)city amount:(double)powerInWatts
{
    // Properties are so important to the language that there is special syntax for accessing
    // and setting them.

    // Use dot-notation to access a property. This is equivalent to calling the accessor
    // with the bracket notation.
    double currentPower = self.currentPower;

    if (currentPower >= powerInWatts) {

        // You can also use dot-notation to write to a property. This is equavalent to calling
        // the setter with bracket notation
        self.currentPower = currentPower - powerInWatts;

        [city addPower:powerInWatts];
    }
}

@end
```

MSVPowerPlant.h — Edited

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

@class MSVCity;

@interface MSVPowerPlant : NSObject

@property (nonatomic, readonly) double currentPower;

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;


@end
```

```objc
#import "MSVPowerPlant.h"
#import "MSVCity.h"

@interface MSVPowerPlant ()

@property (nonatomic) double currentPower;

@end

@implementation MSVPowerPlant

// You can override the generated accessors and setters by providing
// your own method implementation

- (void)setCurrentPower:(double)currentPower
{
    // _currentPower is the automatically generated instance variable for the currentPower
    // property. All automatically generated instance variables will be of the form _(Property Name)

    currentPower = MAX(currentPower, 0);
    _currentPower = currentPower;
}

- (void)sendPowerToCity:(MSVCity *)city amount:(double)powerInWatts
{
    double currentPower = self.currentPower;

    if (currentPower >= powerInWatts) {

        self.currentPower = currentPower - powerInWatts;

        [city addPower:powerInWatts];
    }
}

@end
```

MSVPowerPlant.h — Edited

MSVPowerPlant.h › currentPower

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

@class MSVCity;

@interface MSVPowerPlant : NSObject

@property (nonatomic, readonly) double currentPower;

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;


@end
```

Counterparts › MSVPowerPlant.m › No Selection

```objc
#import "MSVPowerPlant.h"
#import "MSVCity.h"

@interface MSVPowerPlant ()

@property (nonatomic) double currentPower;

@end

@implementation MSVPowerPlant

// When we override both the getter and setter, we must tell the compiler
// that we still want it to create an instance variable for it. The @syntesize
// directive says "I want to create an instance variable named _currentPower for the
// property currentPower". _currentPower will be of type double because of the type in
// the property definition

@synthesize currentPower = _currentPower;

- (void)setCurrentPower:(double)currentPower
{
    currentPower = MAX(currentPower, 0);
    _currentPower = currentPower;
}

- (double)currentPower
{
    return _currentPower;
}

- (void)sendPowerToCity:(MSVCity *)city amount:(double)powerInWatts
{
    double currentPower = self.currentPower;

    if (currentPower >= powerInWatts) {

        self.currentPower = currentPower - powerInWatts;

        [city addPower:powerInWatts];
    }
}

@end
```