

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
```

Superclass's header file.  
This is often <UIKit/UIKit.h>.

```
@interface Spaceship : Vehicle
```

Class name

Superclass

@end

Spaceship.m

```
#import "Spaceship.h"
```

Importing our own header file.

```
@implementation Spaceship
```

Note, superclass not specified here.

@end

# Objective-C

Spaceship.h

```
#import "Vehicle.h"

@interface Spaceship : Vehicle

// declaration of public methods
```

@end

Spaceship.m

```
#import "Spaceship.h"

@implementation Spaceship

// implementation of public and private methods
```

@end

# Objective-C

Spaceship.h

```
#import "Vehicle.h"

@interface Spaceship : Vehicle

// declaration of public methods
```

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods
```

Don't forget the ().

No superclass here either.

@end

# Objective-C

Spaceship.h

Spaceship.m

```
#import "Vehicle.h"  
#import "Planet.h"
```

We need to import Planet.h for method declaration below to work.

```
@interface Spaceship : Vehicle
```

```
// declaration of public methods
```

The full name of this method is orbitPlanet:atAltitude:

```
- (void)orbitPlanet:(Planet *)aPlanet  
    atAltitude:(double)km;
```

Lining up the colons makes things look nice.

It takes two arguments.  
Note how each is preceded by its own keyword.

It does not return any value.

```
@end
```

```
#import "Spaceship.h"
```

```
@interface Spaceship()  
// declaration of private methods (as needed)
```

```
@end
```

```
@implementation Spaceship
```

```
// implementation of public and private methods
```

```
@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)

@end

@implementation Spaceship

// implementation of public and private methods
```

No semicolon here.

```
- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}
```

@end

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

- (void)orbitPlanet:(Planet *)aPlanet
    atAltitude:(double)km;

- (void)setTopSpeed:(double)percentSpeedOfLight;
- (double)topSpeed;
```

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)

@end

@implementation Spaceship

// implementation of public and private methods
```

```
- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}
```

@end

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

- (void)orbitPlanet:(Planet *)aPlanet
    atAltitude:(double)km;

- (void)setTopSpeed:(double)percentSpeedOfLight;
- (double)topSpeed;
```

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)

@end

@implementation Spaceship

// implementation of public and private methods

- (void)setTopSpeed:(double)speed
{
    ???
}

- (double)topSpeed
{
    ???
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}

@end
```

# Objective-C

Spaceship.h

Spaceship.m

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;

- (void)setTopSpeed:(double)percentSpeedOfLight;
- (double)topSpeed;
```

This `@property` essentially declares the two "topSpeed" methods below.

`nonatomic` means its setter and getter are not thread-safe. That's no problem if this is UI code because all UI code happens on the main thread of the application.

@end

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

- (void)setTopSpeed:(double)speed
{
    ???
}

- (double)topSpeed
{
    ???
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}

@end
```

@end

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

We never declare both the `@property` and its setter and getter in the header file (just the `@property`).

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)

@end

@implementation Spaceship

// implementation of public and private methods

- (void)setTopSpeed:(double)speed
{
    ???
}

- (double)topSpeed
{
    ???
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}

@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

We almost always use `@synthesize` to create the implementation of the setter and getter for a `@property`. It both creates the setter and getter methods AND creates some storage to hold the value.

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    ???
}

- (double)topSpeed
{
    ???
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}

@end
```

This is the name of the storage location to use.

\_ (underbar) then the name of the property is a common naming convention.

If we don't use = here, `@synthesize` uses the name of the property (which is **bad** so always use =).

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

This is what the methods  
created by `@synthesize`  
would look like.

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)

@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    _topSpeed = speed;
}

- (double)topSpeed
{
    return _topSpeed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}

@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)

@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
```

Most of the time, you can let `@synthesize` do all the work of creating setters and getters

```
- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}
```

@end

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}
```

@end

However, we can create our own if there is any special work to do when setting or getting.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

Here's another `@property`.  
This one is private (because it's in our .m file).

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@property (nonatomic, strong) Wormhole *nearestWormhole;
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}

@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

It's a pointer to an object (of class Wormhole).  
It's **strong** which means that the memory used by this object will stay around for as long as we need it.

All objects are always allocated on the heap.  
So we always access them through a pointer. Always.

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@property (nonatomic, strong) Wormhole *nearestWormhole;
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}

@end
```

# Objective-C

Spaceship.h

Spaceship.m

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

This creates the setter and getter for our new `@property`.

`@synthesize` does NOT create storage for the object this pointer points to. It just allocates room for the pointer.

We'll talk about how to allocate and initialize the objects themselves next week.

@end

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@property (nonatomic, strong) Wormhole *nearestWormhole;
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
@synthesize nearestWormhole = _nearestWormhole;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}

@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

Now let's take a look at some example coding.  
This is just to get a feel for Objective-C syntax.

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@property (nonatomic, strong) Wormhole *nearestWormhole;
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
@synthesize nearestWormhole = _nearestWormhole;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
}

@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

The "square brackets" syntax is used to send messages.

We're calling topSpeed's getter on ourself here.

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@property (nonatomic, strong) Wormhole *nearestWormhole;
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
@synthesize nearestWormhole = _nearestWormhole;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
    double speed = [self topSpeed];
    if (speed > MAX_RELATIVE) speed = MAX_RELATIVE;
}

@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;

- (void)setTopSpeed:(double)percentSpeedOfLight;
- (double)topSpeed;
```

A reminder of what our getter declaration looks like. Recall that these two declarations are accomplished with the `@property` for `topSpeed` above.

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@property (nonatomic, strong) Wormhole *nearestWormhole;
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
@synthesize nearestWormhole = _nearestWormhole;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
    double speed = [self topSpeed];
    if (speed > MAX_RELATIVE) speed = MAX_RELATIVE;
}

}
```

@end

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

Here's another example of sending a message. It looks like this method has 2 arguments: a Planet to travel to and a speed to travel at. It is being sent to an instance of Wormhole.

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@property (nonatomic, strong) Wormhole *nearestWormhole;
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
@synthesize nearestWormhole = _nearestWormhole;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
    double speed = [self topSpeed];
    if (speed > MAX_RELATIVE) speed = MAX_RELATIVE;
    [[self nearestWormhole] travelToPlanet:aPlanet
                                   atSpeed:speed];
}

@end
```

Square brackets inside square brackets.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

Calling getters and setters is such an important task, it has its own syntax: dot notation.

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@property (nonatomic, strong) Wormhole *nearestWormhole;
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
@synthesize nearestWormhole = _nearestWormhole;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
    double speed = self.topSpeed;
    if (speed > MAX_RELATIVE) speed = MAX_RELATIVE;
    [[self nearestWormhole] travelToPlanet:aPlanet
      atSpeed:speed];
}

@end
```

This is identical to [self topSpeed].

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"

@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
  atAltitude:(double)km;
```

@end

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@property (nonatomic, strong) Wormhole *nearestWormhole;
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
@synthesize nearestWormhole = _nearestWormhole;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet atAltitude:(double)km
{
    // put the code to orbit a planet here
    double speed = self.topSpeed;
    if (speed > MAX_RELATIVE) speed = MAX_RELATIVE;
    [self.nearestWormhole travelToPlanet:aPlanet
                                   atSpeed:speed];
}
```

@end

We can use dot notation here too.

# Dot Notation

## • Dot notation

@property access looks just like C struct member access

```
typedef struct {  
    float x;  
    float y;  
} CGPoint;
```

Notice that we capitalize `CGPoint` (just like a class name).  
It makes our C struct seem just like an object with `@property`s  
(except you can't send any messages to it).

# Dot Notation

## • Dot notation

@property access looks just like C struct member access

```
typedef struct {  
    float x;  
    float y;  
} CGPoint;
```

```
@interface Bomb  
@property CGPoint position;  
@end
```

```
@interface Ship : Vehicle  
@property float width;  
@property float height;  
@property CGPoint center;  
- (BOOL)getsHitByBomb:(Bomb *)bomb;  
@end
```

Returns whether the passed bomb would hit the receiving Ship.

# Dot Notation

## • Dot notation

@property access looks just like C struct member access

```
typedef struct {  
    float x;  
    float y;  
} CGPoint;
```

```
@interface Bomb  
@property CGPoint position;  
@end
```

```
@interface Ship : Vehicle  
@property float width;  
@property float height;  
@property CGPoint center;  
- (BOOL)getsHitByBomb:(Bomb *)bomb;  
@end
```

```
@implementation Ship
```

```
@synthesize width, height, center;
```

```
- (BOOL)getsHitByBomb:(Bomb *)bomb  
{
```

```
    float leftEdge = self.center.x - self.width/2;  
    float rightEdge = ...;
```

```
    return ((bomb.position.x >= leftEdge) &&  
            (bomb.position.x <= rightEdge) &&  
            (bomb.position.y >= topEdge) &&  
            (bomb.position.y <= bottomEdge));
```

```
}
```

```
@end
```

Dot notation to reference  
an object's @property.

# Dot Notation

## • Dot notation

@property access looks just like C struct member access

```
typedef struct {
    float x;
    float y;
} CGPoint;

@interface Bomb
@property CGPoint position;
@end

@interface Ship : Vehicle
@property float width;
@property float height;
@property CGPoint center;
- (BOOL)getsHitByBomb:(Bomb *)bomb;
@end
```

```
@implementation Ship
@synthesize width, height, center;

- (BOOL)getsHitByBomb:(Bomb *)bomb
{
    float leftEdge = self.center.x - self.width/2;
    float rightEdge = ...;

    return ((bomb.position.x >= leftEdge) &&
            (bomb.position.x <= rightEdge) &&
            (bomb.position.y >= topEdge) &&
            (bomb.position.y <= bottomEdge));
}
```

@end Dot notation to reference  
an object's @property.

Normal C struct  
dot notation.

# Blocks

- What is a **block**?

A block of code (i.e. a sequence of statements inside `{}`).

Usually included "in-line" with the calling of method that is going to use the block of code.

Very smart about local variables, referenced objects, etc.

- What does it look like?

Here's an example of calling a method that takes a **block** as an argument.

```
[aDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id value, BOOL *stop) {
    NSLog(@"value for key %@ is %@", key, value);
    if ([@"ENOUGH" isEqualToString:key]) {
        *stop = YES;
    }
}];
```

This `NSLog()`s every `key` and `value` in `aDictionary` (but stops if the `key` is `ENOUGH`).

- Blocks start with the magical character caret `^`

Then it has (optional) arguments in parentheses, then `{`, then code, then `}`.

# Blocks

- When do we use blocks in iOS?

  - Enumeration

  - View Animations (more on that later in the course)

  - Sorting (sort this thing using a **block** as the comparison method)

  - Notification (when something happens, execute this **block**)

  - Error handlers (if an error happens while doing this, execute this **block**)

  - Completion handlers (when you are done doing this, execute this **block**)

- And a super-important use: Multithreading

  - With Grand Central Dispatch (GCD) API

# Grand Central Dispatch

- GCD is a C API
- The basic idea is that you have queues of operations
  - The operations are specified using blocks.
  - Most queues run their operations serially (a true "queue").
  - We're only going to talk about serial queues today.
- The system runs operations from queues in separate threads
  - Though there is no guarantee about how/when this will happen.
  - All you know is that your queue's operations will get run (in order) at some point.
  - The good thing is that if your operation blocks, only that queue will block.
  - Other queues (like the main queue, where UI is happening) will continue to run.
- So how can we use this to our advantage?
  - Get blocking activity (e.g. network) out of our user-interface (main) thread.
  - Do time-consuming activity concurrently in another thread.

# Grand Central Dispatch

## ● Important functions in this C API

Creating and releasing queues

```
dispatch_queue_t dispatch_queue_create(const char *label, NULL); // serial queue  
void dispatch_release(dispatch_queue_t);
```

Putting blocks in the queue

```
typedef void (^dispatch_block_t)(void);  
void dispatch_async(dispatch_queue_t queue, dispatch_block_t block);
```

Getting the current or main queue

```
dispatch_queue_t dispatch_get_current_queue();  
void dispatch_queue_retain(dispatch_queue_t); // keep it in the heap until dispatch_release  
  
dispatch_queue_t dispatch_get_main_queue();
```

# Grand Central Dispatch

- What does it look like to call these?

Example ... assume we fetched an image from the network (this would be slow).

```
- (void) viewWillAppear:(BOOL)animated
{
    NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
    UIImage *image = [UIImage imageWithData:imageData];
    self.imageView.image = image;
    self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
    self.scrollView.contentSize = image.size;
}
```

# Grand Central Dispatch

- What does it look like to call these?

Example ... assume we fetched an image from the network (this would be slow).

```
- (void) viewWillAppear:(BOOL)animated  
{
```

```
    NSData *imageData = [NSData dataWithContentsOfURL:networkURL];  
    UIImage *image = [UIImage imageWithData:imageData];  
    self.imageView.image = image;  
    self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);  
    self.scrollView.contentSize = image.size;
```

```
}
```

# Grand Central Dispatch

## • What does it look like to call these?

Example ... assume we fetched an image from the network (this would be slow).

```
- (void) viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);

    NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
    UIImage *image = [UIImage imageWithData:imageData];
    self.imageView.image = image;
    self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
    self.scrollView.contentSize = image.size;
}
```

# Grand Central Dispatch

## • What does it look like to call these?

Example ... assume we fetched an image from the network (this would be slow).

```
- (void) viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        UIImage *image = [UIImage imageWithData:imageData];
        self.imageView.image = image;
        self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
        self.scrollView.contentSize = image.size;
    });
}
```

# Grand Central Dispatch

## • What does it look like to call these?

Example ... assume we fetched an image from the network (this would be slow).

```
- (void) viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        UIImage *image = [UIImage imageWithData:imageData];
        self.imageView.image = image;
        self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
        self.scrollView.contentSize = image.size;
    });
}
```

**Problem!** UIKit calls can only happen in the main thread!

# Grand Central Dispatch

## • What does it look like to call these?

Example ... assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];

        UIImage *image = [UIImage imageData:imageData];
        self.imageView.image = image;
        self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
        self.scrollView.contentSize = image.size;
    });
}
```

# Grand Central Dispatch

## • What does it look like to call these?

Example ... assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
}
```

# Grand Central Dispatch

## • What does it look like to call these?

Example ... assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
}
```

**Problem!** This "leaks" the `downloadQueue` in the heap. We have to `dispatch_release` it.

# Grand Central Dispatch

## • What does it look like to call these?

Example ... assume we fetched an image from the network (this would be slow).

```
- (void)viewWillAppear:(BOOL)animated
{
    dispatch_queue_t downloadQueue = dispatch_queue_create("image downloader", NULL);
    dispatch_async(downloadQueue, ^{
        NSData *imageData = [NSData dataWithContentsOfURL:networkURL];
        dispatch_async(dispatch_get_main_queue(), ^{
            UIImage *image = [UIImage imageData:imageData];
            self.imageView.image = image;
            self.imageView.frame = CGRectMake(0, 0, image.size.width, image.size.height);
            self.scrollView.contentSize = image.size;
        });
    });
    dispatch_release(downloadQueue);
}
```

Don't worry, it won't remove the queue from the heap until all blocks have been processed.