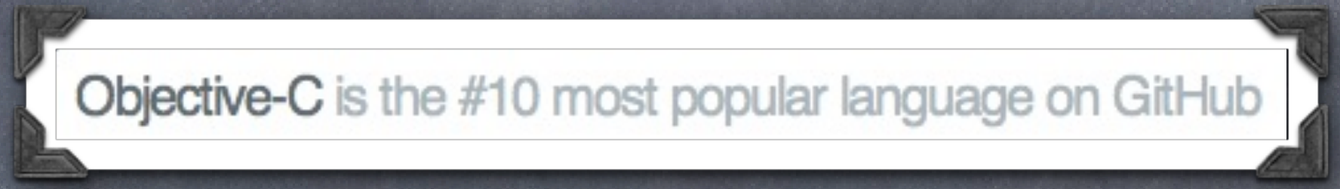
iOS App Development

Matthias Tretter, @myell0w

What will I learn?



- How to build iOS apps
  - Objective-C
  - Cocoa Touch
- Real-life Object-Oriented Programming
  - MVC, Delegation, Target-Action, Closures

Prerequisites

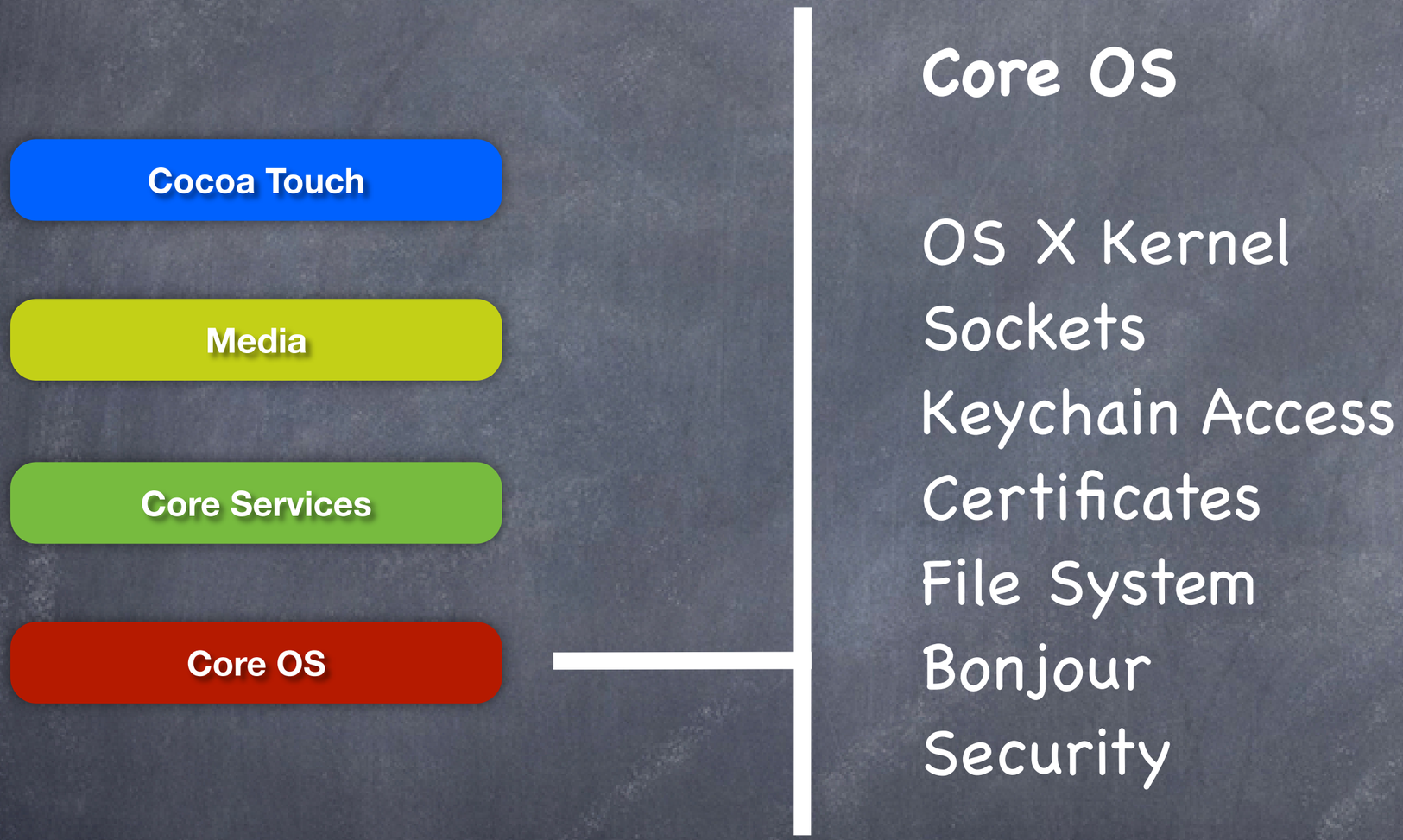
- C Programming
- Object-Oriented Programming
  Class (description/template for an object)
  Instance (manifestation of a class)
  Message (sent to object to make it act)
  Method (code invoked by a Message)
  Instance Variable (object-specific storage) Superclass/Subclass (Inheritance)
  Protocol/Interface (non-class-specific methods)

Final Project

- 4 EH „Fernlehre“
- small iOS app
  - student proposals

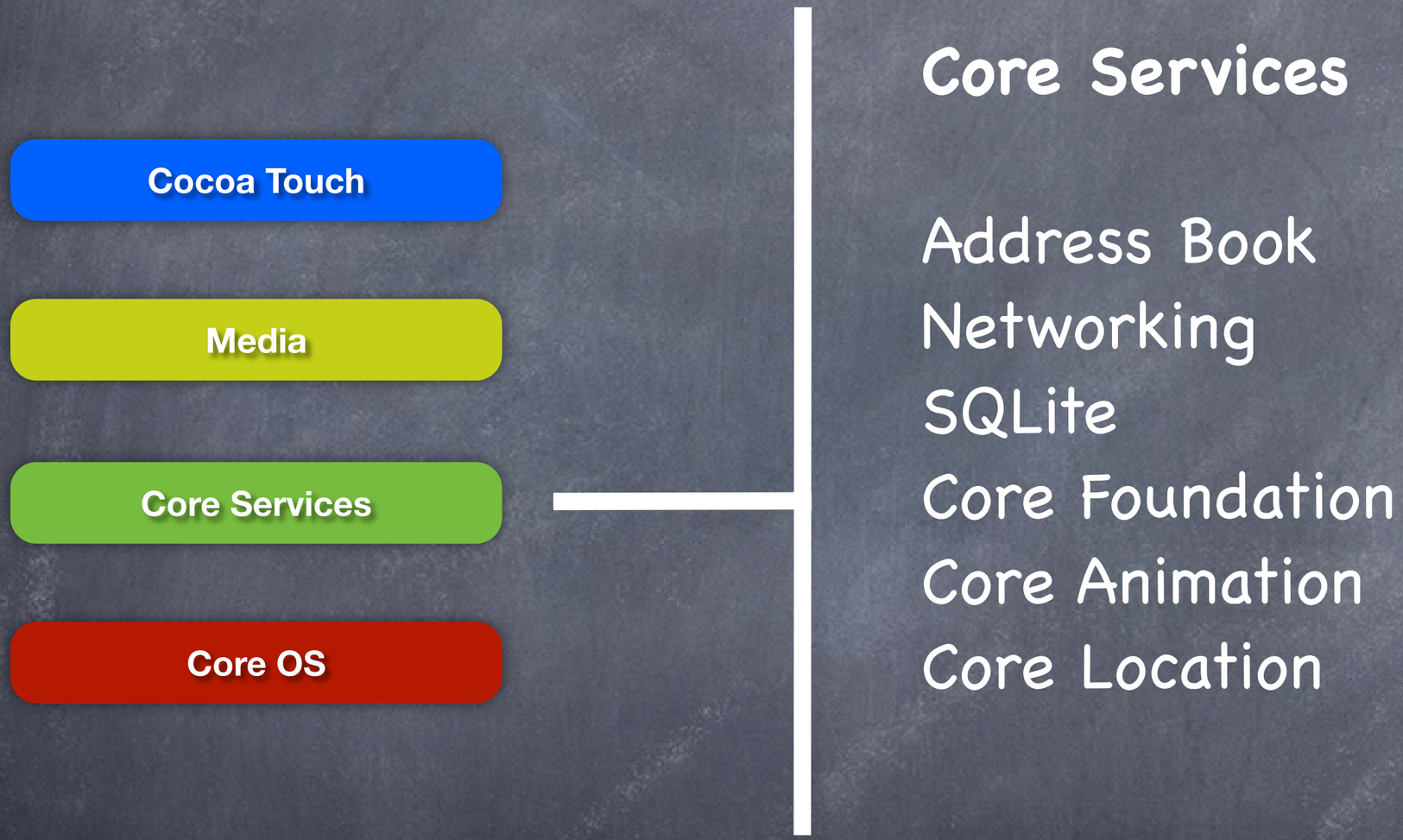
    or

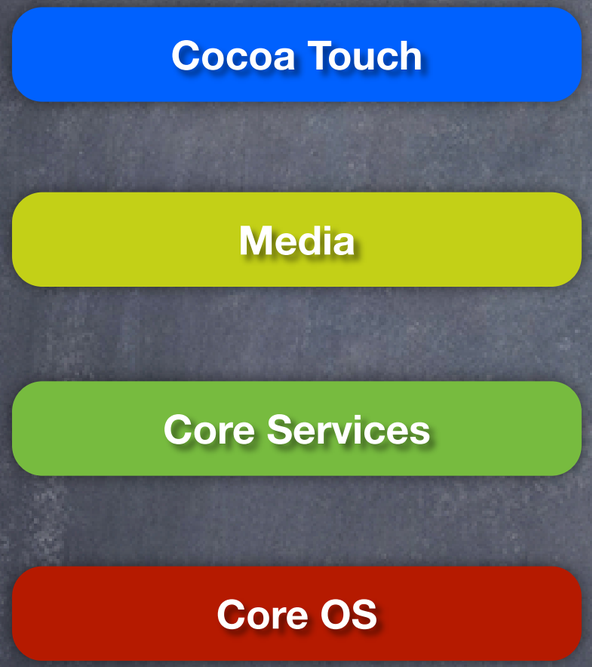
  - fixed project
- Submission: https://github.com/FHCampus-ITTBA13/



Core OS

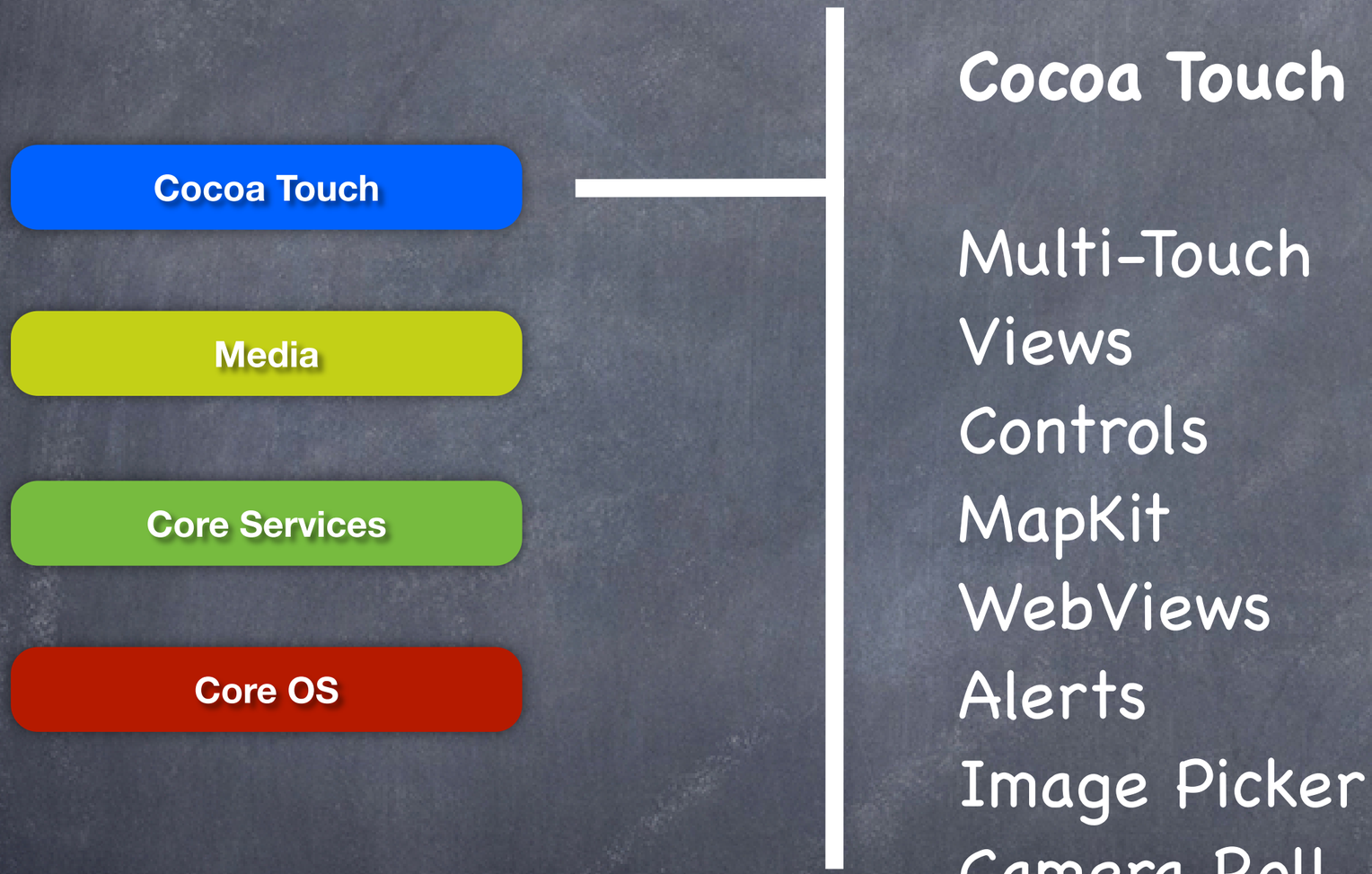
OS X Kernel
Sockets
Keychain Access
Certificates
File System
Bonjour
Security


Cocoa Touch
Media
Core Services
Core OS
Core Services
Address Book
Networking
SQLite
Core Foundation
Core Animation
Core Location



Media

Core Audio
JPEG/PNG/TIFF
PDF
Quartz
OpenGL ES
Video Playback
Audio Recording/Mixing


Cocoa Touch
Media
Core Services
Core OS
Cocoa Touch
Multi-Touch
Views
Controls
MapKit
WebViews
Alerts
Image Picker
Camera Roll

Platform Components

- Tools

Xcode 4

Instruments

- Language

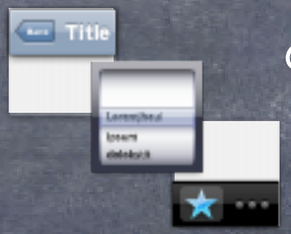
```
[display setTextColor:[UIColor blackColor]];
```

- Frameworks

Foundation

Core Data

Map Kit

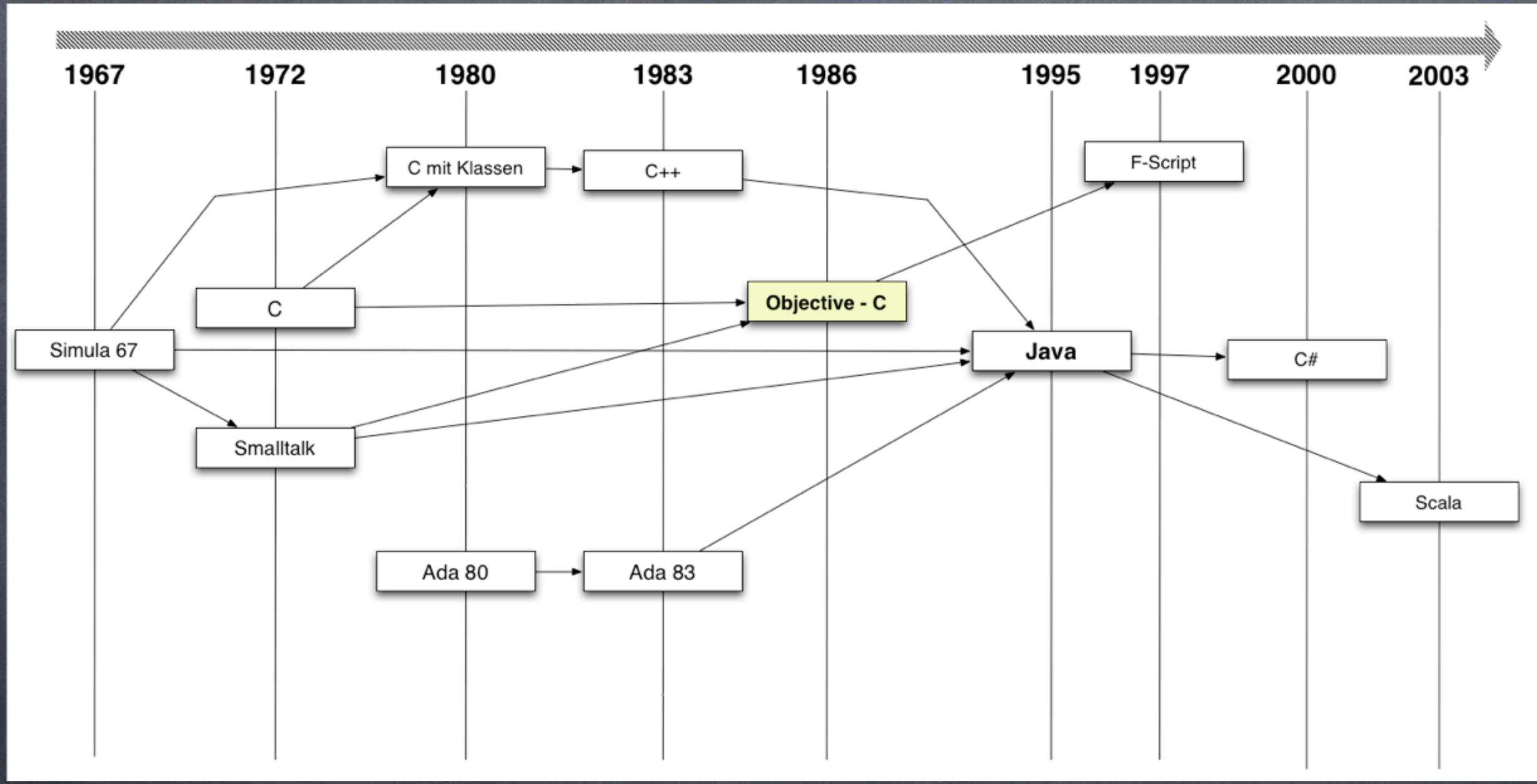


UIKit

Core Motion

- Design Strategies

MVC



Source: http://en.wikipedia.org/wiki/Timeline_of_programming_languages

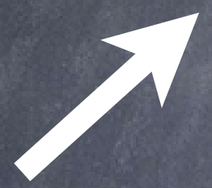
[Objective-C description]

- Object-Oriented Superset of C
- Shortly after C++
- dynamic, message passing
- 2007: Objective-C 2.0
- 2011: ARC
- 2012: Literals

```
// Klassenbezeichnung und Ableitung
@interface Bruch : NSObject < Protokoll >
// Instanzvariablen, für jede Instanz angelegt.
{
    NSInteger zaehler;
    NSInteger nenner;
}
// Eigenschaften der Instanzen
@property( assign ) NSInteger zaehler;
@property( assign ) NSInteger nenner;

// Methoden:
- (void) printLn;
- (float) floatValue;
@end
```

[Objective-C showSyntax]



```
- (BOOL)willAppOfDeveloper:(NSString *)developer
        getRejectedWhenUsingPrivateAPI:(SEL)aPrivateAPI {

    if (self.usesPrivateAPI) {
        @try {
            [self performSelector:@selector(aPrivateAPI)
                       withObject:[NSNumber numberWithInt:5]];
        }
        @catch (NSException *exception) {
            return YES;
        }

        return ![developer isEqualToString:@"Tim Cook"];
    }

    return NO;
}
```

Objective-C

- New language to learn!
  - Strict superset of C
  - Adds syntax for classes, methods, etc.
  - A few things to "think differently" about (e.g. properties, dynamic binding)
- Most important concept to understand today: Properties
  - Usually we do not access instance variables directly in Objective-C.
  - Instead, we use "properties."
  - A "property" is just the combination of a getter method and a setter method in a class.
  - The getter has the name of the property (e.g. "myValue")
  - The setter's name is "set" plus capitalized property name (e.g. "setMyValue:")
  - (To make this look nice, we always use a lowercase letter as the first letter of a property name.)
  - We just call the setter to store the value we want and the getter to get it. Simple.
- This is just your first glimpse of this language!
  - We'll go much more into the details next week.
  - Don't get too freaked out by the syntax at this point.

Objective-C

Spaceship.h

Spaceship.m

Objective-C

Spaceship.h

```
#import "Vehicle.h"

@interface Spaceship : Vehicle


@end
```

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

Class name

Superclass

Spaceship.m

```
#import "Spaceship.h"

@implementation Spaceship


@end
```

Importing our own header file.

Note, superclass not specified here.

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




@end
```

Don't forget the ().

No superclass here either.

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
```

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

The full name of this method is orbitPlanet:atAltitude:

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.

Spaceship.m

```
#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

```
#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;




@end
```

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.

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;


@end
```

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

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;

@end
```

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.

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;




@end
```

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

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;



@end
```

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

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;

@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
```

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.

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
```

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.

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;




@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
```

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

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;


}

@end
```

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

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

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;



@end
```

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

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;

@end
```

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`.

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;




@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
```

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

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.

Further Literature

- „The Objective-C Programming Language“, Apple
- „Programming with Objective-C“, Apple
  http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html#//apple_ref/doc/uid/TP40011210
- „Learn Objective-C on the Mac“, Scott Knaster & Mark Dalrymple
  http://www.apress.com/9781430218159
- Slides
  https://github.com/FHCampus-ITTBA13/Slides