# Object-Oriented Programming and User Interfaces

COSC346

## Instructors

- Lecturer: David Eyers, Owheo 1.25
    - dme@cs.otago.ac.nz

ousa
otago uni **students'** association

# WANTED CLASS REPS

**SEMESTER TWO 2017**

## Are you

- proactive, friendly and keen to contribute to your learning environment?
- a great communicator who can represent your peers?

## What's in it for you?

- Kudos & Karma
- Great friendships
- Access to FREE professional training opportunities and support
- A reference letter from OUSA for your CV
- Invitations to Class Rep social events throughout the year

**Don't wait any longer... sign up now!**

**Email your lecturer your ID, university email address and name!**

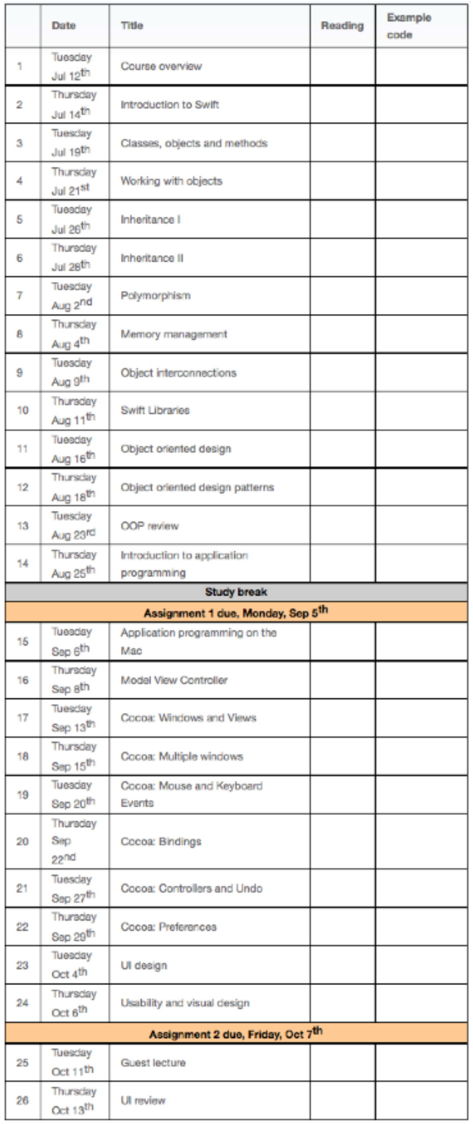
## Schedule

- **Lectures:**
  - Tuesday 13:00–13:50, UOCE Tower Block Room G08, (TG08)
  - Thursday 13:00–13:50, UOCE Tower Block Room G08, (TG08)
- **Labs** (separate streams):
  - Wednesday 10:00 – 11:50, Owheo G.38 (Lab F)
  - Wednesday 12:00 – 13:50, Owheo G.38 (Lab F)
  - There **is a lab** in the first week
- **Tutorials** (separate streams):
  - Tuesday 10:00 – 10:50, Teaching College, T101
  - Friday 10:00 – 10:50, Geology Building, Quad 3
  - There will be **no tutorial** first week

## Grades

- Assignment 1: 20%, due Monday Sep 4th
- Assignment 2: 20%, due Friday Oct 6th
- Final Exam: 60%
- You may work in pairs or individually

# Course Overview: Lectures



| | Date | Title | Reading | Example code |
|---|---|---|---|---|
| 1 | Tuesday Jul 12th | Course overview | | |
| 2 | Thursday Jul 14th | Introduction to Swift | | |
| 3 | Tuesday Jul 19th | Classes, objects and methods | | |
| 4 | Thursday Jul 21st | Working with objects | | |
| 5 | Tuesday Jul 26th | Inheritance I | | |
| 6 | Thursday Jul 28th | Inheritance II | | |
| 7 | Tuesday Aug 2nd | Polymorphism | | |
| 8 | Thursday Aug 4th | Memory management | | |
| 9 | Tuesday Aug 9th | Object interconnections | | |
| 10 | Thursday Aug 11th | Swift Libraries | | |
| 11 | Tuesday Aug 16th | Object oriented design | | |
| 12 | Thursday Aug 18th | Object oriented design patterns | | |
| 13 | Tuesday Aug 23rd | OOP review | | |
| 14 | Thursday Aug 25th | Introduction to application programming | | |
| **Study break** | | | | |
| **Assignment 1 due, Monday, Sep 5th** | | | | |
| 15 | Tuesday Sep 6th | Application programming on the Mac | | |
| 16 | Thursday Sep 8th | Model View Controller | | |
| 17 | Tuesday Sep 13th | Cocoa: Windows and Views | | |
| 18 | Thursday Sep 15th | Cocoa: Multiple windows | | |
| 19 | Tuesday Sep 20th | Cocoa: Mouse and Keyboard Events | | |
| 20 | Thursday Sep 22nd | Cocoa: Bindings | | |
| 21 | Tuesday Sep 27th | Cocoa: Controllers and Undo | | |
| 22 | Thursday Sep 29th | Cocoa: Preferences | | |
| 23 | Tuesday Oct 4th | UI design | | |
| 24 | Thursday Oct 6th | Usability and visual design | | |
| **Assignment 2 due, Friday, Oct 7th** | | | | |
| 25 | Tuesday Oct 11th | Guest lecture | | |
| 26 | Thursday Oct 13th | UI review | | |

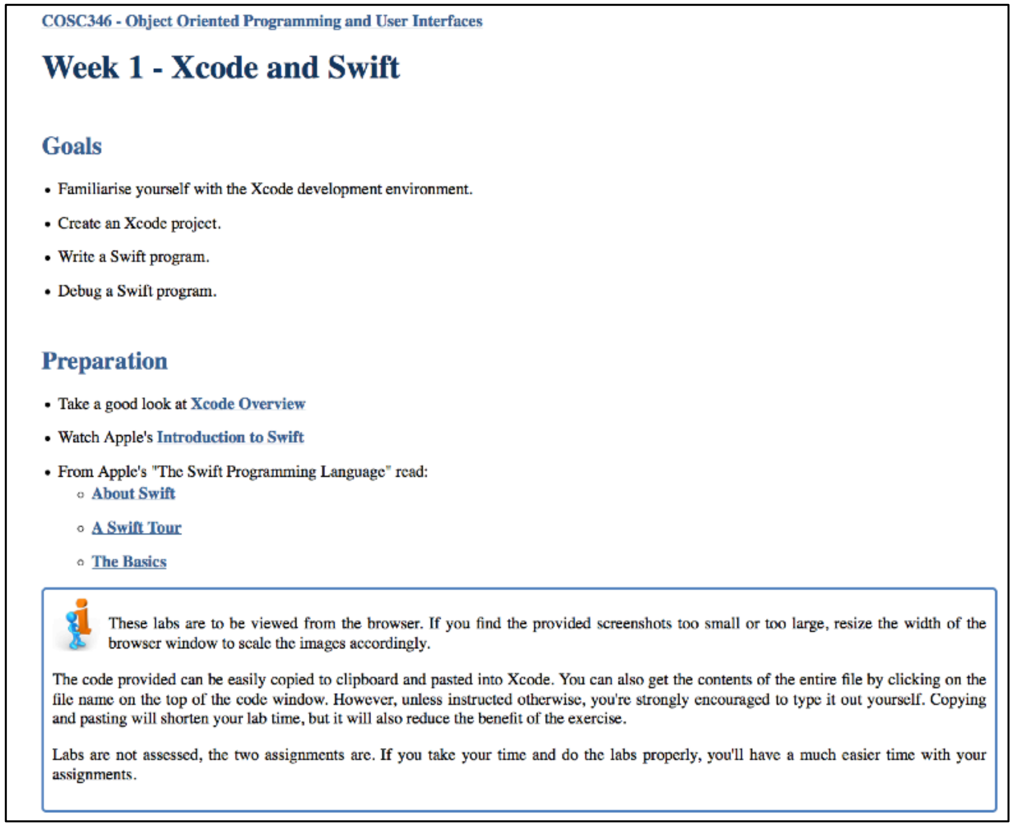
- Object Oriented Programming
  - General concepts: abstraction, encapsulation, inheritance, polymorphism, coupling, cohesion
  - Swift language and Foundation Framework
  - Swift development tools - Xcode
  - Object oriented design principles
- User Interfaces
  - Cocoa Environment and Xcode
  - Interface design principles: usability, basics of graphic design

OOP

Swift

# Course Overview: Labs

- On the course webpage
- Not assessed
- First lab tomorrow



COSC346 - Object Oriented Programming and User Interfaces

## Week 1 - Xcode and Swift

### Goals

- Familiarise yourself with the Xcode development environment.
- Create an Xcode project.
- Write a Swift program.
- Debug a Swift program.

### Preparation

- Take a good look at Xcode Overview
- Watch Apple's Introduction to Swift
- From Apple's "The Swift Programming Language" read:
  - About Swift
  - A Swift Tour
  - The Basics

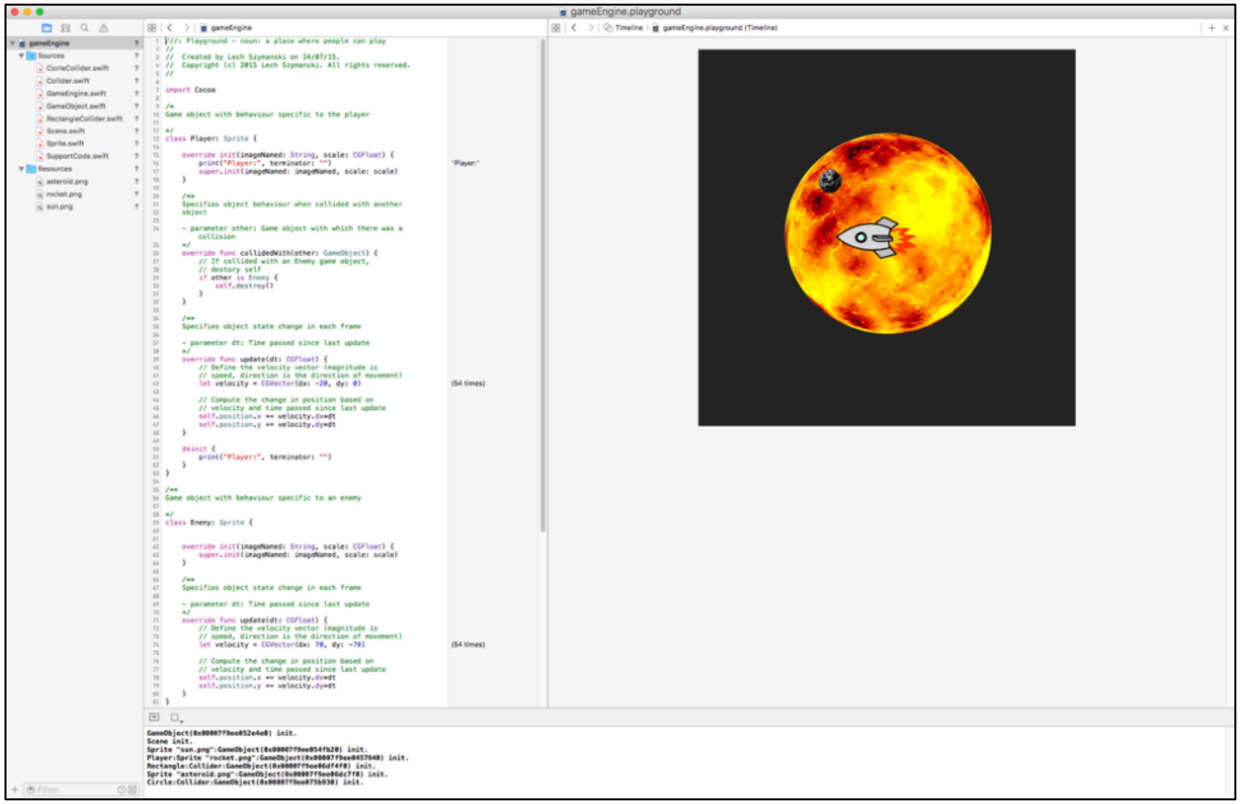
These labs are to be viewed from the browser. If you find the provided screenshots too small or too large, resize the width of the browser window to scale the images accordingly.

The code provided can be easily copied to clipboard and pasted into Xcode. You can also get the contents of the entire file by clicking on the file name on the top of the code window. However, unless instructed otherwise, you're strongly encouraged to type it out yourself. Copying and pasting will shorten your lab time, but it will also reduce the benefit of the exercise.

Labs are not assessed, the two assignments are. If you take your time and do the labs properly, you'll have a much easier time with your assignments.

# Course Overview: Tutorials

- As needed



```swift
//: Playground - noun: a place where people can play
//
//  Created by Lech Szymanski on 24/07/15.
//  Copyright (c) 2015 Lech Szymanski. All rights reserved.
//

import Cocoa

/*
Game object with behaviour specific to the player

*/
class Player: Sprite {

    override init(imageNamed: String, scale: CGFloat) {
        print("Player:", terminator: "")
        super.init(imageNamed: imageNamed, scale: scale)
    }

    /**
    Specifies object behaviour when collided with another
    object

    - parameter other: Game object with which there was a
        collision
    */
    override func collidedWith(other: GameObject) {
        // If collided with an Enemy game object,
        // destory self
        if other is Enemy {
            self.destroy()
        }
    }

    /**
    Specifies object state change in each frame

    - parameter dt: Time passed since last update
    */
    override func update(dt: CGFloat) {
        // Define the velocity vector (magnitude is
        // speed, direction is the direction of movement)
        let velocity = CGVector(dx: -20, dy: 0)

        // Compute the change in position based on
        // velocity and time passed since last update
        self.position.x += velocity.dx*dt
        self.position.y += velocity.dy*dt
    }

    deinit {
        print("Player:", terminator: "")
    }
}

/**
Game object with behaviour specific to an enemy

*/
class Enemy: Sprite {


    override init(imageNamed: String, scale: CGFloat) {
        super.init(imageNamed: imageNamed, scale: scale)
    }

    /**
    Specifies object state change in each frame

    - parameter dt: Time passed since last update
    */
    override func update(dt: CGFloat) {
        // Define the velocity vector (magnitude is
        // speed, direction is the direction of movement)
        let velocity = CGVector(dx: 70, dy: -70)

        // Compute the change in position based on
        // velocity and time passed since last update
        self.position.x += velocity.dx*dt
        self.position.y += velocity.dy*dt
    }
}
```

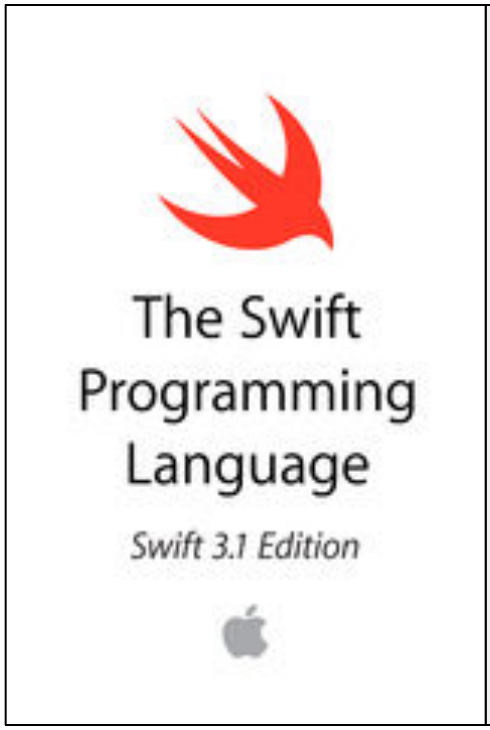
```
GameObject(0x00007f9ee052e4e0) init.
Scene init.
Sprite "sun.png":GameObject(0x00007f9ee054fb20) init.
Player:Sprite "rocket.png":GameObject(0x00007f9ee0457040) init.
Rectangle:Collider:GameObject(0x00007f9ee06df4f0) init.
Sprite "asteroid.png":GameObject(0x00007f9ee06dc7f0) init.
Circle:Collider:GameObject(0x00007f9ee075b030) init.
```

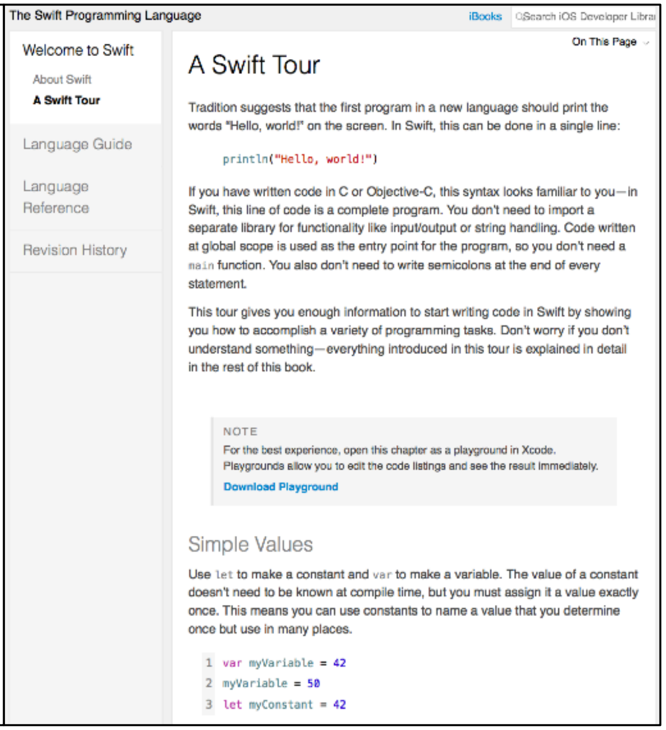
# Reading

The Swift Programming Language (2017)

Apple Inc.

- Up to date
- Free
- HTML, iBook format





A Swift Tour

Tradition suggests that the first program in a new language should print the words "Hello, world!" on the screen. In Swift, this can be done in a single line:

```
println("Hello, world!")
```

If you have written code in C or Objective-C, this syntax looks familiar to you—in Swift, this line of code is a complete program. You don't need to import a separate library for functionality like input/output or string handling. Code written at global scope is used as the entry point for the program, so you don't need a `main` function. You also don't need to write semicolons at the end of every statement.

This tour gives you enough information to start writing code in Swift by showing you how to accomplish a variety of programming tasks. Don't worry if you don't understand something—everything introduced in this tour is explained in detail in the rest of this book.

NOTE

For the best experience, open this chapter as a playground in Xcode. Playgrounds allow you to edit the code listings and see the result immediately.

Download Playground

Simple Values

Use `let` to make a constant and `var` to make a variable. The value of a constant doesn't need to be known at compile time, but you must assign it a value exactly once. This means you can use constants to name a value that you determine once but use in many places.

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

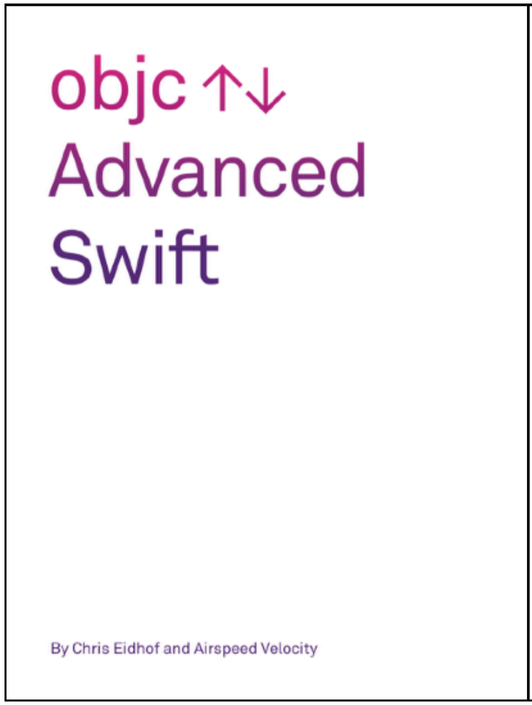
# Reading

Advanced Swift (2016)

C. Eidhof, A. Velocity

Objc.io

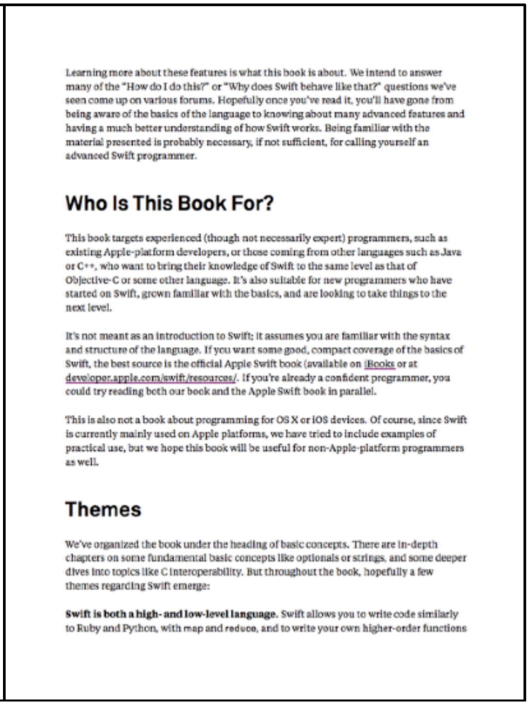
- For programmers that come from other languages (such as Java)
- E-book formats: PDF, mobi



objc ↑↓

Advanced Swift

By Chris Eidhof and Airspeed Velocity



Learning more about these features is what this book is about. We intend to answer many of the "How do I do this?" or "Why does Swift behave like that?" questions we've seen come up on various forums. Hopefully once you've read it, you'll have gone from being aware of the basics of the language to knowing about many advanced features and having a much better understanding of how Swift works. Being familiar with the material presented is probably necessary, if not sufficient, for calling yourself an advanced Swift programmer.

## Who Is This Book For?

This book targets experienced (though not necessarily expert) programmers, such as existing Apple-platform developers, or those coming from other languages such as Java or C++, who want to bring their knowledge of Swift to the same level as that of Objective-C or some other language. It's also suitable for new programmers who have started on Swift, grown familiar with the basics, and are looking to take things to the next level.

It's not meant as an introduction to Swift; it assumes you are familiar with the syntax and structure of the language. If you want some good, compact coverage of the basics of Swift, the best source is the official Apple Swift book (available on iBooks or at developer.apple.com/swift/resources/. If you're already a confident programmer, you could try reading both our book and the Apple Swift book in parallel.

This is also not a book about programming for OS X or iOS devices. Of course, since Swift is currently mainly used on Apple platforms, we have tried to include examples of practical use, but we hope this book will be useful for non-Apple-platform programmers as well.

## Themes

We've organized the book under the heading of basic concepts. There are in-depth chapters on some fundamental basic concepts like optionals or strings, and some deeper dives into topics like C interoperability. But throughout the book, hopefully a few themes regarding Swift emerge:

**Swift is both a high- and low-level language.** Swift allows you to write code similarly to Ruby and Python, with map and reduce, and to write your own higher-order functions

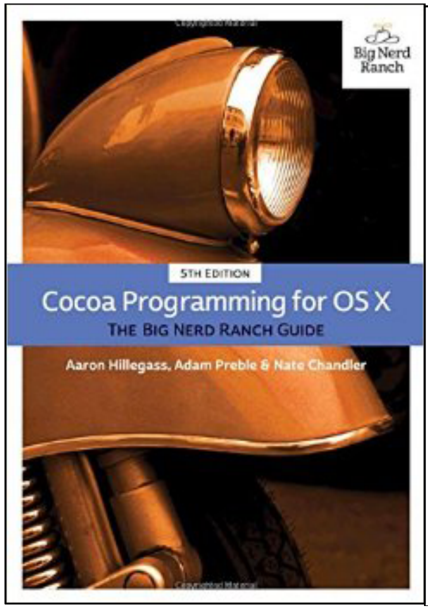
# Reading

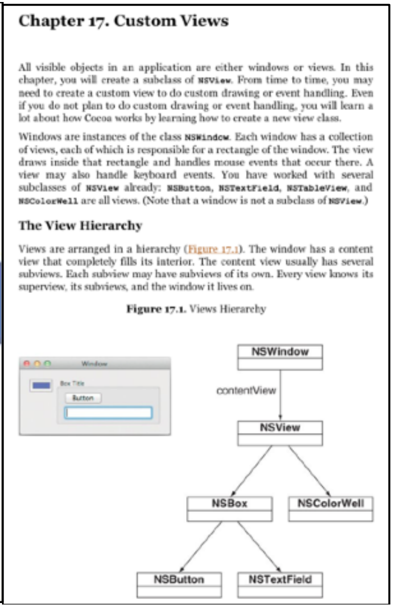
*Cocoa Programming for Mac OS X*, 5th ed. (2015)

A. Hillegass, A. Preble, N. Chandler

Big Nerd Ranch Guides

- Updated for Xcode 6
- Updated for Swift 2.x
- Excellent examples
- Still probably one of the best books on Cocoa development
- Hardcopy in the lab


Big Nerd Ranch
5TH EDITION
Cocoa Programming for OS X
THE BIG NERD RANCH GUIDE
Aaron Hillegass, Adam Preble & Nate Chandler



**Chapter 17. Custom Views**

All visible objects in an application are either windows or views. In this chapter, you will create a subclass of `NSView`. From time to time, you may need to create a custom view to do custom drawing or event handling. Even if you do not plan to do custom drawing or event handling, you will learn a lot about how Cocoa works by learning how to create a new view class.

Windows are instances of the class `NSWindow`. Each window has a collection of views, each of which is responsible for a rectangle of the window. The view draws inside that rectangle and handles mouse events that occur there. A view may also handle keyboard events. You have worked with several subclasses of `NSView` already: `NSButton`, `NSTextField`, `NSTableView`, and `NSColorWell` are all views. (Note that a window is not a subclass of `NSView`.)

**The View Hierarchy**

Views are arranged in a hierarchy (Figure 17.1). The window has a content view that completely fills its interior. The content view usually has several subviews. Each subview may have subviews of its own. Every view knows its superview, its subviews, and the window it lives on.

**Figure 17.1.** Views Hierarchy

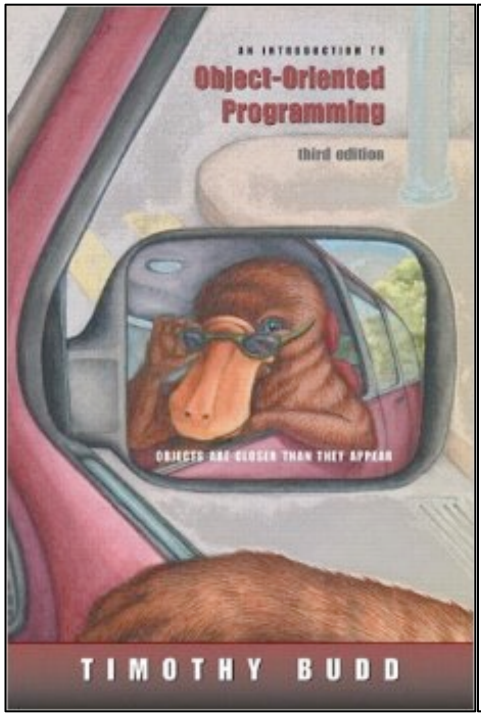
## Reading

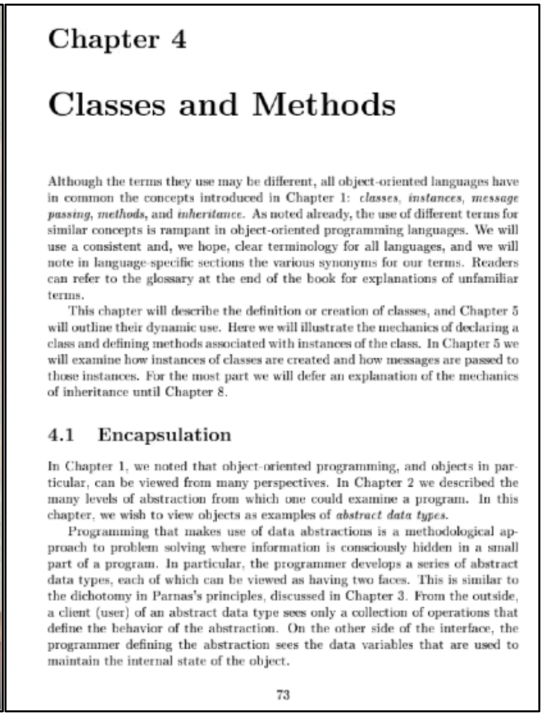
*Object-Oriented Programming*, 3rd ed. (2002)

Timothy Budd

Addison-Wesley

- General OOP principles
- Hardcopy in Science Library on reserve


AN INTRODUCTION TO
Object-Oriented Programming
third edition
OBJECTS ARE CLOSER THAN THEY APPEAR
TIMOTHY BUDD



Chapter 4

Classes and Methods

Although the terms they use may be different, all object-oriented languages have in common the concepts introduced in Chapter 1: *classes*, *instances*, *message passing*, *methods*, and *inheritance*. As noted already, the use of different terms for similar concepts is rampant in object-oriented programming languages. We will use a consistent and, we hope, clear terminology for all languages, and we will note in language-specific sections the various synonyms for our terms. Readers can refer to the glossary at the end of the book for explanations of unfamiliar terms.

This chapter will describe the definition or creation of classes, and Chapter 5 will outline their dynamic use. Here we will illustrate the mechanics of declaring a class and defining methods associated with instances of the class. In Chapter 5 we will examine how instances of classes are created and how messages are passed to those instances. For the most part we will defer an explanation of the mechanics of inheritance until Chapter 8.

4.1 Encapsulation

In Chapter 1, we noted that object-oriented programming, and objects in particular, can be viewed from many perspectives. In Chapter 2 we described the many levels of abstraction from which one could examine a program. In this chapter, we wish to view objects as examples of *abstract data types*.

Programming that makes use of data abstractions is a methodological approach to problem solving where information is consciously hidden in a small part of a program. In particular, the programmer develops a series of abstract data types, each of which can be viewed as having two faces. This is similar to the dichotomy in Parnas's principles, discussed in Chapter 3. From the outside, a client (user) of an abstract data type sees only a collection of operations that define the behavior of the abstraction. On the other side of the interface, the programmer defining the abstraction sees the data variables that are used to maintain the internal state of the object.

73

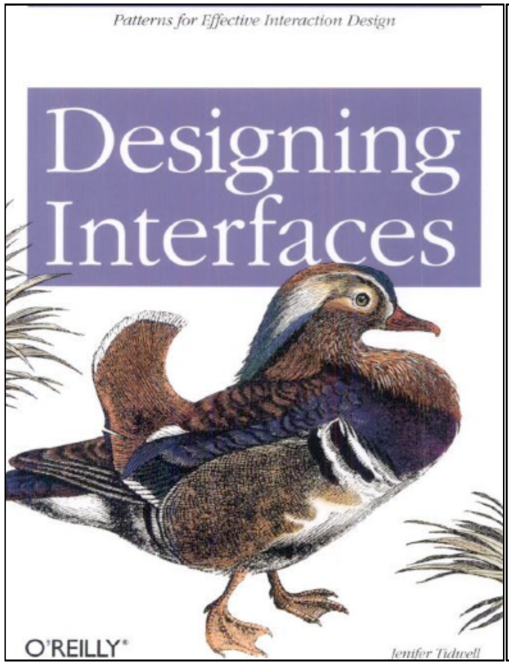
# Reading

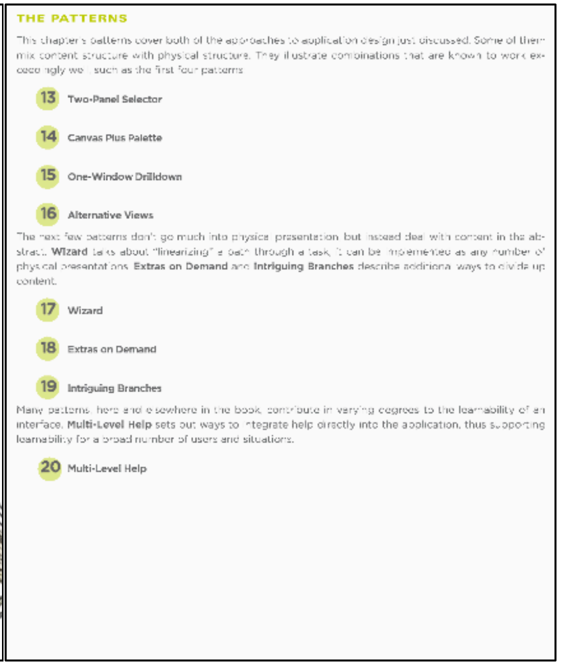
*Designing Interfaces* (2002)

Jenifer Tidwell

O'Reilly Media Inc.

- User Interface principles and design patterns
- Electronic version from the Science Library




THE PATTERNS

This chapter's patterns cover both of the approaches to application design just discussed. Some of them mix content structure with physical structure. They illustrate combinations that are known to work exceedingly well, such as the first four patterns.

13 **Two-Panel Selector**

14 **Canvas Plus Palette**

15 **One-Window Drilldown**

16 **Alternative Views**

The next few patterns don't go much into physical presentation but instead deal with content in the abstract. **Wizard** talks about "linearizing" a path through a task; it can be implemented as any number of physical presentations. **Extras on Demand** and **Intriguing Branches** describe additional ways to divide up content.

17 **Wizard**

18 **Extras on Demand**

19 **Intriguing Branches**

Many patterns, here and elsewhere in the book, contribute in varying degrees to the learnability of an interface. **Multi-Level Help** sets out ways to integrate help directly into the application, thus supporting learnability for a broad number of users and situations.

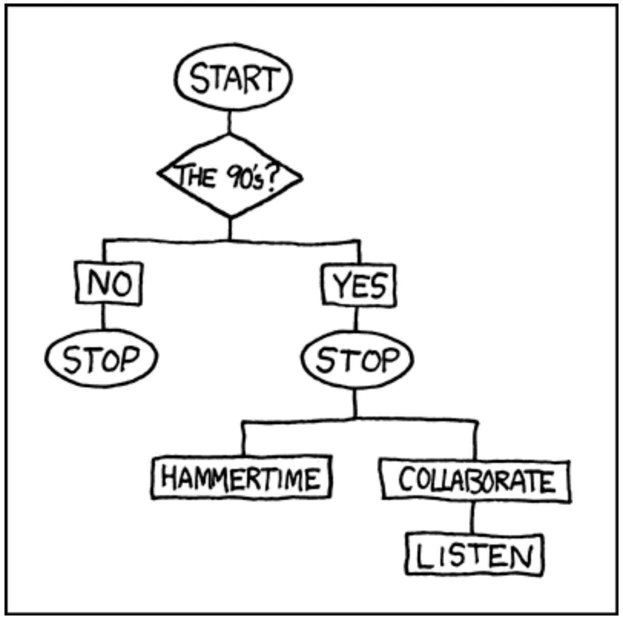
20 **Multi-Level Help**

# Reading

- The Swift Programming Language (2017), Apple Inc.
- C. Eidhof, A. Velocity (2016), Advanced Swift, Objc.io.
- A. Hillegass, A. Preble, N. Chandler (2015), *Cocoa Programming for Mac OS X* ***(5th ed)***, Big Nerd Ranch Guides.
- Timothy Budd (2002), *Object-Oriented Programming* *(3rd ed)*, Addison-Wesley.
- Jenifer Tidwell (2006), Designing Interfaces, O'Reilly Media, Inc.

# What is OOP?

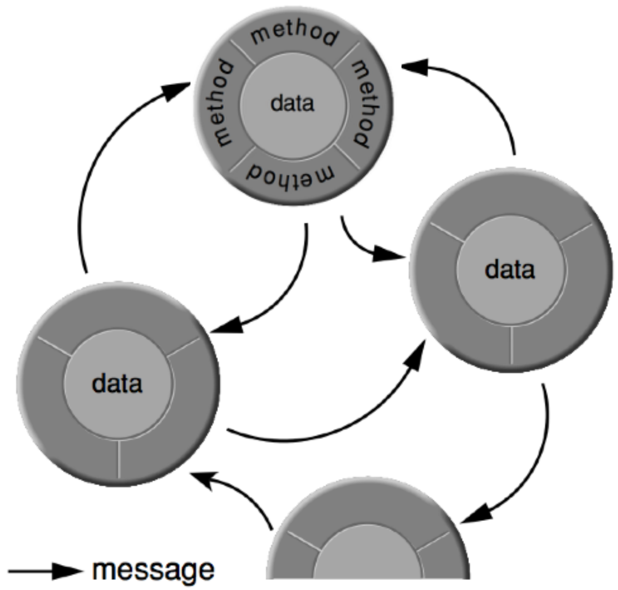
Procedural



xkcd.com

1. Functions act on data.
2. A program organises function calls to manipulate data.

Object-Oriented

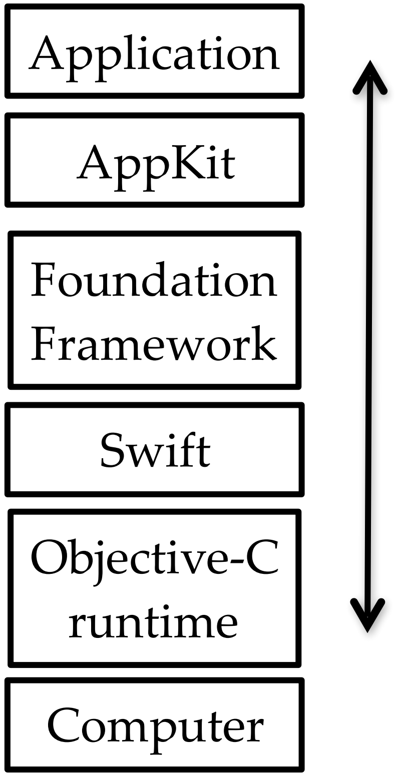


1. *Objects* contain *encapsulated* data and associated *methods*.
2. A program describes how objects interact via *messages*.

## What is OOP?

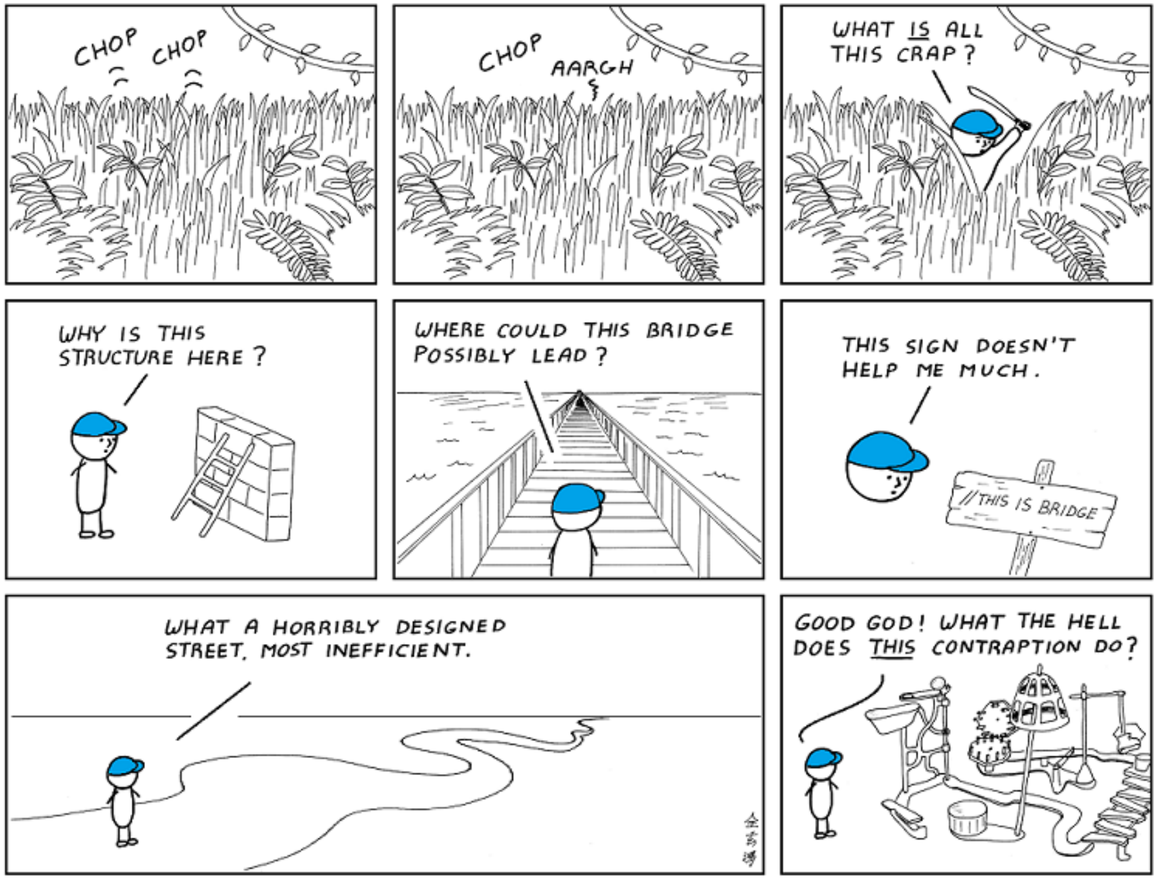
vs.

# Why OOP?



| Speed | Code | Development | Environment | User Interface |
|---|---|---|---|---|
| Slow | Re-usable | Team, Fast | Runtime Decisions | Complex, Graphical |
| Fast | Specific | Individual, Slow | Compile-time Decisions | Simple, Text-based |

# Reusability


CHOP CHOP
CHOP AARGH
WHAT IS ALL THIS CRAP?
WHY IS THIS STRUCTURE HERE?
WHERE COULD THIS BRIDGE POSSIBLY LEAD?
THIS SIGN DOESN'T HELP ME MUCH.
//THIS IS BRIDGE
WHAT A HORRIBLY DESIGNED STREET. MOST INEFFICIENT.
GOOD GOD! WHAT THE HELL DOES THIS CONTRAPTION DO?

**I hate reading other people's code.**

abstrusegoose.com

## Why Swift?

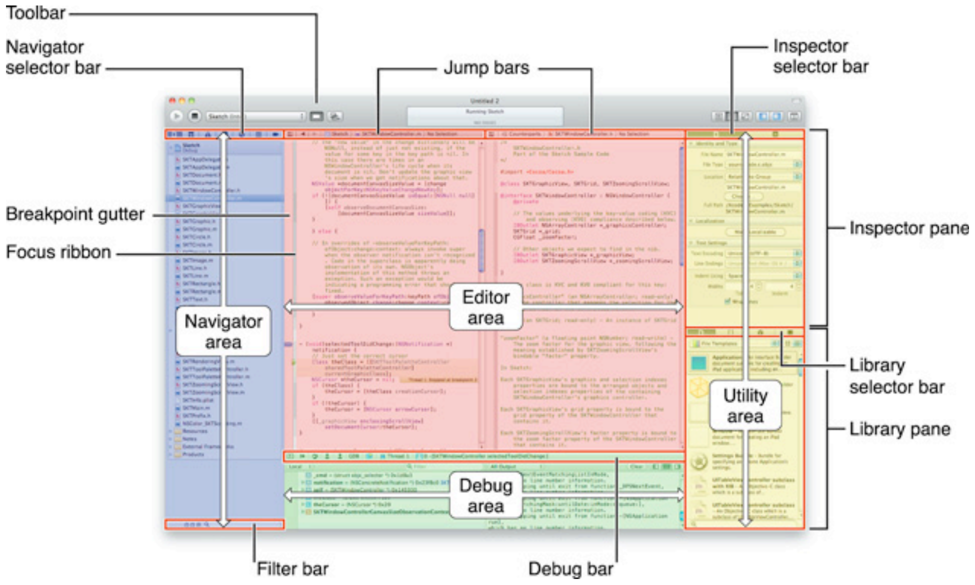
- Modern
  - Result of research on programming languages
  - Multi-paradigm – takes best features from many languages (in COSC346 we focus on the Object-Oriented aspect)
- Safe
  - Compiler forces you to do things right
  - Tries to detect errors at compile time, not run-time
- Concise
  - Easier and faster to develop software
  - Easier to create development tools
- Cocoa environment – good example of natural progression from OOP to User Interfaces

## What is Cocoa?

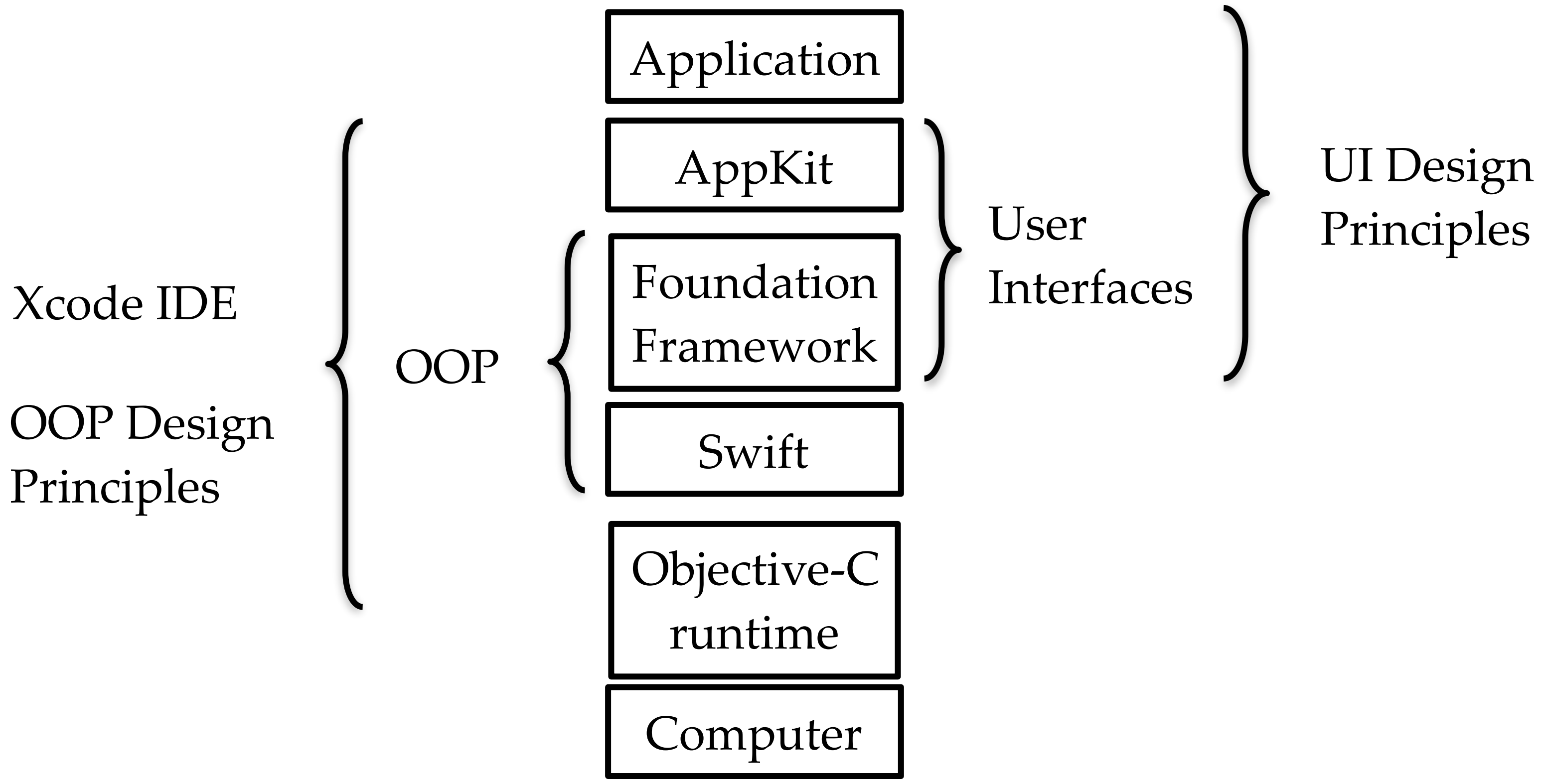
- Object-oriented framework for application development for OS X and iOS
  - In this course we will focus on OS X only
- "Its elegant and powerful design is ideally suited for the rapid development of software" – Cocoa Fundamentals Guide (2010, retired), Apple Inc.
- Huge number of classes and frameworks
  - Overwhelming for the first-time user
  - Powerful environment that abstracts away a lot of the details of application programming – you can concentrate on high level functionality

# What is Xcode?

- Integrated Development Environment (IDE) for application development for OS X & iOS
  - It comes with iOS platform simulator
- Compiler and debugging tools
- Cocoa libraries and frameworks
  - Interface builder – GUI for creating GUIs
- Editor and tools for analysis

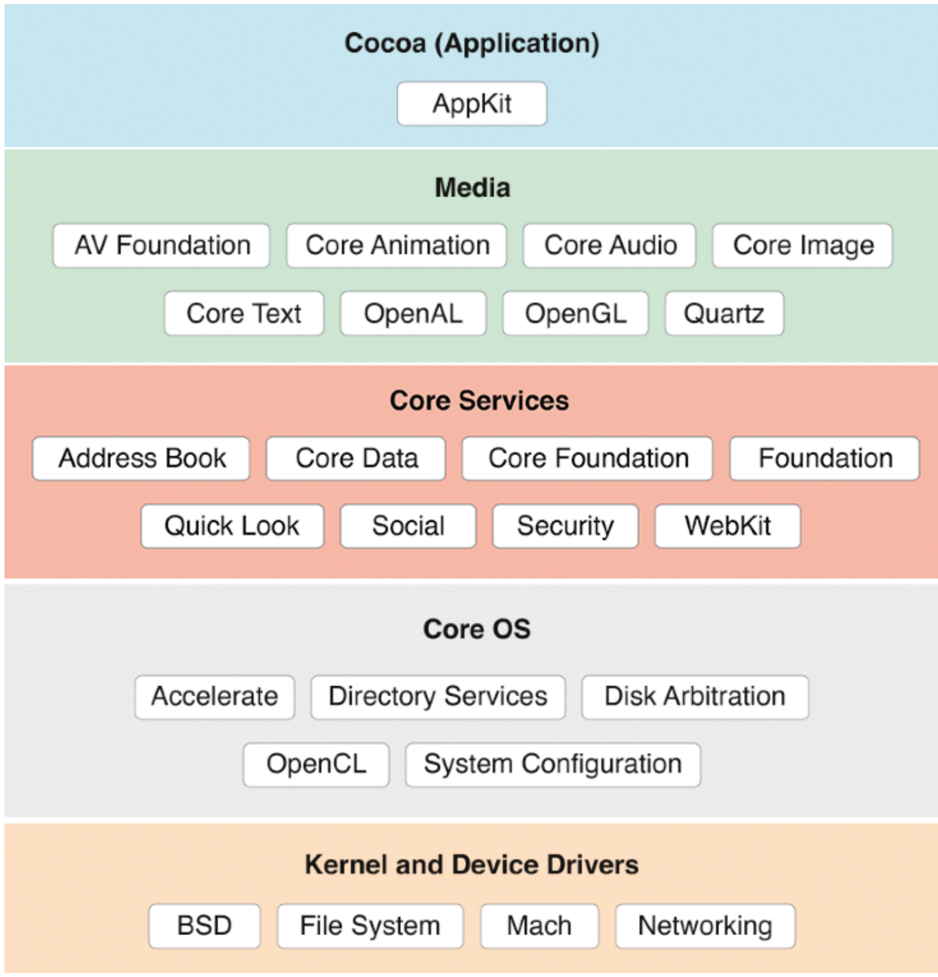


## Mac Platform



## Goals

- Object-Oriented Programming:
  - (a) Learn Swift language
  - (b) Understand OOP design principles
- User Interfaces:
  - (a) Learn Application Kit Framework
  - (b) Understand UI design principles


Cocoa (Application)
AppKit
Media
AV Foundation
Core Animation
Core Audio
Core Image
Core Text
OpenAL
OpenGL
Quartz
Core Services
Address Book
Core Data
Core Foundation
Foundation
Quick Look
Social
Security
WebKit
Core OS
Accelerate
Directory Services
Disk Arbitration
OpenCL
System Configuration
Kernel and Device Drivers
BSD
File System
Mach
Networking