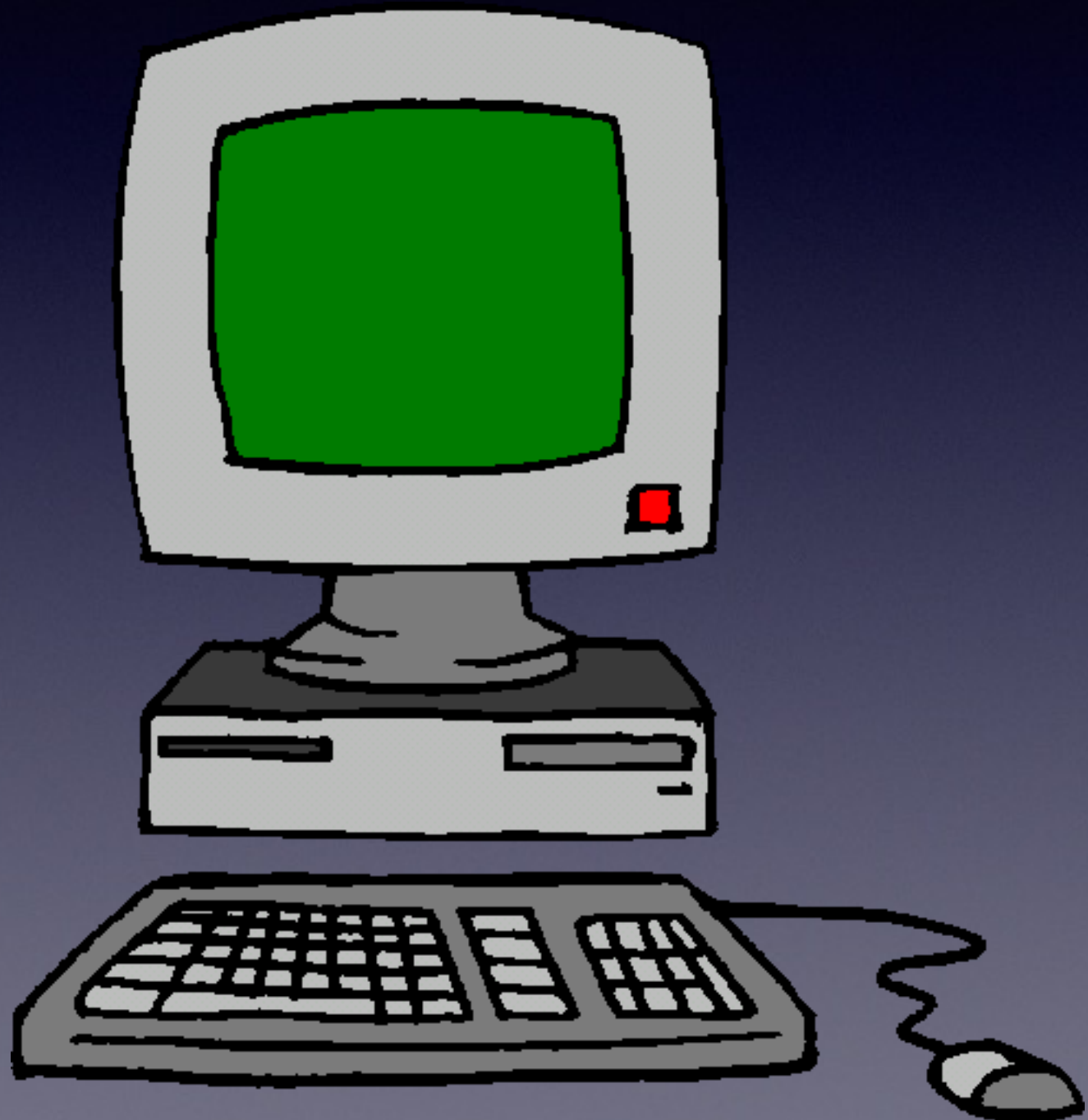# Swift Fundamentals IV Classes Intro

CS112 Unit 5
Max Luttrell, Fall 2016

objects?

- A system can be thought of as a group of objects that interact with each other

classes and objects

- Swift allows us to bundle functions and data together by defining a **class**

- Once we have defined a class, we can create **objects** of the class. This is known as instantiating an object of the class

- A note on terminology — inside of a class:

  - functions are often called **methods** or member functions

  - data are often called **properties** or member data

class vs. object
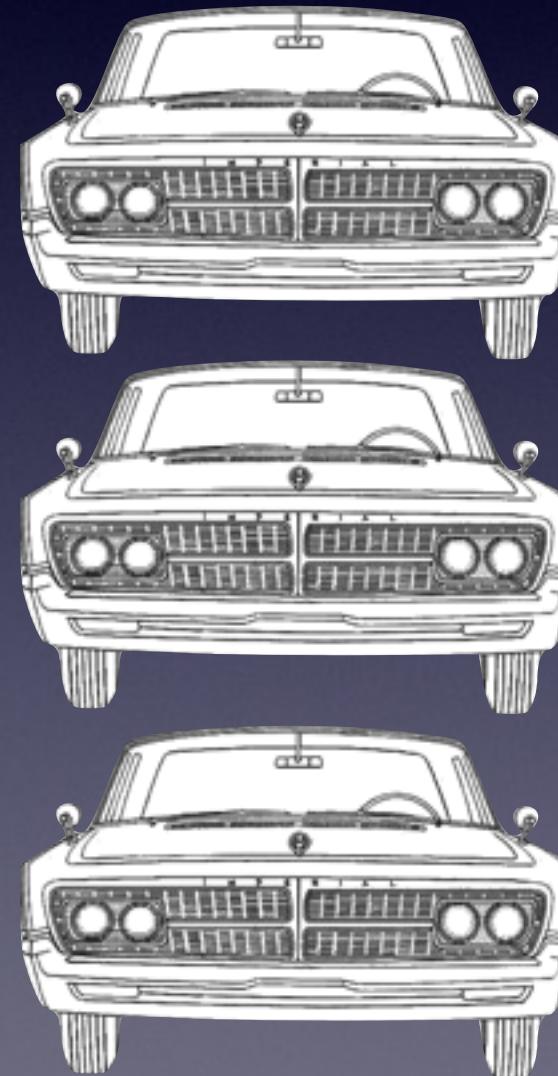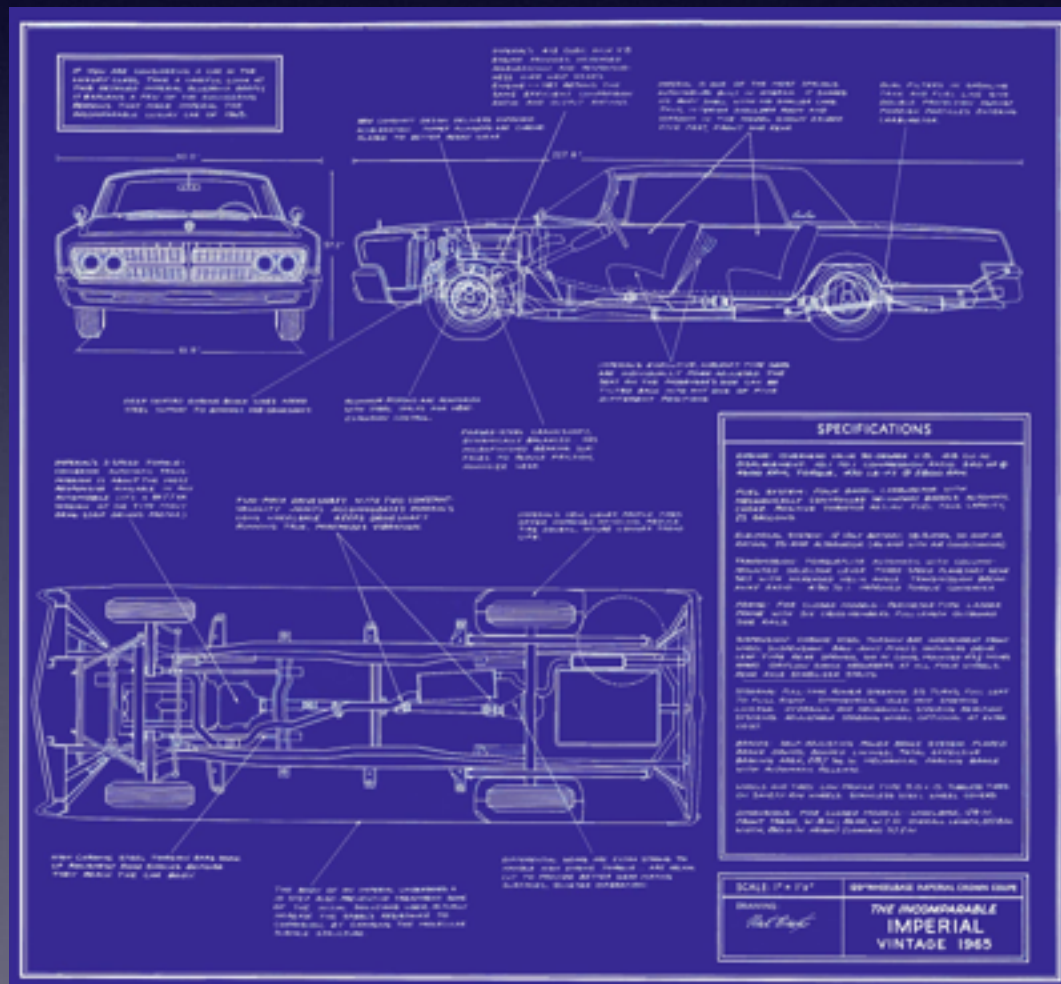
abstraction, instantiation

car

make
model
currentSpeed
fuelLevel

startEngine()
stopEngine()
accelerate()
brake()
refuel()

Abstraction: distill the essential data and methods into a class

honda
civic
35
12

startEngine()
stopEngine()
accelerate()
brake()
refuel()

vw
jetta
0
8

startEngine()
stopEngine()
accelerate()
brake()
refuel()

honda
civic
55
2

startEngine()
stopEngine()
accelerate()
brake()
refuel()

*Instantiatiation: create **objects** from class*

class definition

- Bundle your data and functions together by defining a **class**. Here, we define a class for a rectangle

```
class Rectangle {
    var length = 0
    var width = 0
    func incrementWidth() {
        width += 1
    }
    func incrementLength() {
        length += 1
    }
}
```

instantiation

- here, we instantiate a Rectangle object and use it with the "dot" notation: **.**

*instantiating a
rectangle object*

dot notation

```
class Rectangle {
    var length = 0
    var width = 0
    func incrementWidth() {
        width += 1
    }
    func incrementLength() {
        length += 1
    }
}
```

```
var bedroom = Rectangle()

bedroom.incrementWidth()
bedroom.incrementLength()
bedroom.incrementLength()
print("Length: \(bedroom.length), Width: \(bedroom.width)")
```

Sample Debug Output
Length: 2, Width: 1

more objects

- we can access an object's properties directly with "dot" notation

```
class Rectangle {
    var length = 0
    var width = 0
    func incrementWidth() {
        width += 1
    }
    func incrementLength() {
        length += 1
    }
}
```

```
var kitchen = Rectangle()
kitchen.length = 5
kitchen.width = 4
print("Length: \(kitchen.length), Width: \(kitchen.width)")
```

Sample Debug Output
Length: 5, Width: 4

more objects

```swift
class Rectangle {
    var length = 0
    var width = 0
    func incrementWidth() {
        width += 1
    }
    func incrementLength() {
        length += 1
    }
    func printInfo() {
        print("length: \(length), width: \(width)")
    }
}
```

```swift
var kitchen = Rectangle()
kitchen.length = 5
kitchen.width = 4
kitchen.printInfo()
```

Sample Debug Output
Length: 5, Width: 4

method returning a value

```
class Rectangle {
    var length = 0
    var width = 0
    func incrementWidth() {
        width += 1
    }
    func incrementLength() {
        length += 1
    }
    func printInfo() {
        print("length: \(length), width: \(width)")
    }
    func getArea() -> Int {
        return length*width
    }
}
```

```
var kitchen = Rectangle()
kitchen.length = 5
kitchen.width = 4
kitchen.printInfo()
print(kitchen.getArea())
```

Sample Debug Output
20

Exercise 5A

- In a playground, define the Rectangle class as discussed today (you can just copy from the slide)

- Enhance the Rectangle class with two new methods:

  - getPerimeter() -> Int -- returns the perimeter of the Rectangle, i.e. 2*width+2*length

  - isSquare() -> Bool - returns true if the rectangle is a square, i.e. length is equal to width, returns false if not

- Demonstrate your class works by instantiating two Rectangle objects. One should be square and one should not be. Then print the output of your two new methods to the debug area