GENERAL ASSEMBLY

MOBILE DEVELOPMENT

08 — OBJECT-ORIENTED PROGRAMMING

LEARNING OBJECTIVES

- Define object-oriented programming.
- Identify and apply object oriented principles:
  - inheritance,
  - polymorphism,
  - encapsulation.
- *Differentiate between classes and structs.*
- *Create protocols and apply them to classes, structs, and types.*

WHAT IS OOP?

WHAT IS OBJECT-ORIENTED PROGRAMMING?

Wikipedia says:

- **Object-oriented programming (OOP)** *is a programming paradigm based on the concept of "objects", which are data structures that*
  - *contain data, in the form of fields, often known as attributes; and*
  - *code, in the form of procedures, often known as methods.*

WHAT IS OBJECT-ORIENTED PROGRAMMING?

Wikipedia says also:

- *In OO programming, computer programs are designed by making them out of objects that interact with one another.*

WHAT IS OBJECT-ORIENTED PROGRAMMING?

Some of the characteristics of OOP we've already discussed:

- Classes and instances
- Inheritance
- Polymorphism
- Composition (instances holding instances of other classes)

WHAT IS OBJECT-ORIENTED PROGRAMMING?

Some of the characteristics of OOP we've already discussed:

- Classes and instances
- Inheritance
- Polymorphism
- Composition (instances holding instances of other classes)

There's one more important one:

- Encapsulation

OOP CONCEPTS : ENCAPSULATION

- In practice, classes can be tricky. They are very flexible, almost too flexible. We need some guiding principles for how to use them effectively.
- One rule is to have each class represent *one thing*, and each method, property, a line do one thing, etc.
- Encapsulation is a rule that isolates code according to that one-rule principle.
  - It ensures that there is no other code or data related to the class appearing outside the class.
  - It makes us limit the scope of various methods and properties so other objects don't see the whole picture.
  - It separates "implementation" from "interface."

HOW DO WE USE CLASSES TO DESCRIBE APP DATA?

- Creating a "data model" is the process of writing classes that mimic the data we want to store, manipulate, and represent in an app.
- Like the exercise we did before, where a Pizza class could potentially have a class Topping that represents pizza toppings, we can design the data in familiar terms and build an app around them.

MVC PATTERN

- There are some "design patterns" that we find useful in building apps. One is MVC, or Model-View-Controller pattern.
- The View is exactly what it sounds like. It's very much a UIView (or a subclass) that draws something to the screen and handles interactivity.
- A Controller (or for us, View Controller) is a class that manages the relationship between data, Views, and interaction.
- A Model is a class that just represents data and the things we can do to that data. e.g. Add a Topping to a Pizza.

SINGLETON PATTERN

- Sometimes we need to have only one instance of a class.
- One way to ensure that all parts of our app work with the same data is to attach that common data to single instance, and only allow the app to instantiate one instance of the class.
- This is called a *Singleton*.

EXERCISE: MAKE A WEATHER APP!

POWERING APPS WITH CODE

EXERCISE

KEY OBJECTIVE(S)

Make a weather app with a well-defined object model.

TIMING

25 min	1. Code with partner
5 min	2. Debrief

DELIVERABLE

A working weather app that enables you to pick among 3 locations and see the weather there (fake data).

EXERCISE: SIMULATE A GAME

SIMULATE A GAME

- Make three classes, ‘Player’, ‘GoodPlayer’ and ‘BadPlayer’.
  - Player has an ‘attack’ method, which returns an Int, the amount of damage the attack does.
  - GoodPlayers and BadPlayers, when they attack, attack with different magnitudes (between GoodPlayer and BadPlayer, or [BONUS] different from attack to attack).
  - Players also have a health property (an Int that defaults to 100) and an ‘isAlive’ method (returns a Boolean). A Player is alive if their health is above 0.
- Create a ‘Match’ class, which takes two players during initialization.
  - It has a ‘playGame()’ method, which pits each player against each other, alternating attacks until one of the players is no longer alive. At the end of the match, return the winner.
  - Hint: This means you’ll need a way to have the attack of one player affect the health of the other.
- Pit one GoodPlayer against a BadPlayer.
- Bonus: Give players names, print those out before they match.