Boxes (P*)

Learning objectives:

- Understand the bundling of data and methods together to form objects
- Understand the use of self in a method on an object
- Understand encapsulation

Introduction

Object Oriented Programming allows developers to think of different components as objects. Our minds are familiar with the concept of objects and thinking of everything as objects that have behaviour and state.

Take your laptop for example. It has a specific role to play and you, the user, can interact with it. The screen of the laptop can be altered by the user. So the laptop has a screen state. The user can't just update the state of the screen directly though. The laptop exposes interfaces for the user to update what's on the screen (usually via keyboard and trackpad). It has other states too, like the amount of free vs used memory, the disk space. Is it powered on or off? Perhaps it's in sleep mode.

Typing and clicking are actions that can be performed on the Laptop object in order to change it's internal state (screen, memory, disk, etc).

Objective-C uses "Class-based" object system, where the blueprint for Objects are defined via Classes. Using our previous laptop example, think of the Laptop class as a factory which can be used to produce instances of identical laptops that start off with the same state and behaviour.

Instructions

- Create a new Command Line Application. Choose Objective-C as the language.
- Create a class called Box that is a subclass of NSObject. We are making a blueprint to hold the properties and methods that will apply to multiple instances of boxes.
- Inside Box:
    - Add three properties (height, width, and length) and make each member a float.
    - Create an instance method that initializes a Box by taking in three floats as parameters. An instance method is a method that applies to a specific instance of a Box, it has a “-” sign at the start of the function name.
    - Create a instance method that will calculate the volume and return it as a float. You refer to the properties of an instance by prefixing “self.”

- Inside main.m:
    - Initialize a Box using three floats as inputs for height, width, and length
    - Calculate the volume of the box and check your answer by NSLogging the box’s volume
    - To find the volume of any cube you need to know the length, width and height. The formula to find the volume multiplies the length by the width by the height.
- Inside Box:
    - Add a method that takes in another box and returns how many times one box will fit inside the other. Be conscious about understanding which box has a greater volume and how that will affect the result.
    - **NOTE:** Just use the volumes to calculate how many times a box can fit into another box, don't bother with trying to come up with the logic of physically fitting boxes into each other.