Pizza Manager using delegates (P*)

Learning Objectives

- Understand delegation protocol patterns
- Understand how delegation helps polymorphism
- Understand how delegation is more than just a callback mechanism, and how it can extend an object's functionality

Instructions

We're going to keep working on that pizza place app. We know how to make pizza, but when should we be making pizza? We're going to use the delegate pattern to allow the Kitchen to behave differently under different circumstances. Let's make a protocol so we can get some management up in here.

Open up the PizzaRestaurant project and create a new protocol named `KitchenDelegate`. Add three methods to it:

1. `(BOOL)kitchen:(Kitchen *)kitchen shouldMakePizzaOfSize:(PizzaSize)size andToppings:(NSArray *)toppings;`
2. `(BOOL)kitchenShouldUpgradeOrder:(Kitchen *)kitchen;`
3. `(void)kitchenDidMakePizza:(Pizza *)pizza;` (make this method optional)

Create a `delegate` property on the `Kitchen` class. Use the `KitchenDelegate` methods in the `makePizza...` method. If there is a delegate set, call the first

two methods before creating a pizza. If the delegate returns `NO` (false) for the `shouldMakePizza` method, don't make the pizza at all. If the delegate returns `YES` (true) for the `shouldUpgrade` method, then make a large pizza, no matter what was actually ordered. Once the pizza has been made, send a `kitchenDidMakePizza` message to the delegate. Remember, `kitchenDidMakePizza` is optional; so, you'll have to check in code to see if, whoever the delegate is, actually implements this method. If the `delegate` does respond, send the message, otherwise don't. (HINT: use `respondsToSelector:`. Look it up in the Documentation).

Now create a new Manager class and have it conform to our `KitchenDelegate` protocol. This will be for a class of managers that hate anchovies so much, they refuse to make pizzas that have anchovies. So the implementation of `shouldMakePizza` should return `NO` if any of the toppings are anchovies. This manager does not upgrade orders, and doesn't do anything after the kitchen makes a pizza.

Create a second Manager class that also conforms to the `KitchenDelegate` protocol. This manager is more cheery. They don't stop any pizzas from being made, they always upgrade orders, and they "say" a nice thing (just implement this as an `NSLog` statement) when a pizza is made.

Now in the main part of the program, prior to entering the while loop, initialize a manager of each class. Add code to the input checking that will allow the

user to switch from one manager to the other, and to no manager at all. Try ordering some pizzas with each of those three options.

Stretch Gloal

Add logic to `main.m` that makes sure that you only ever create an instance of each manager IF they are needed, and if you do need them, make sure the next time you need them you don't just create a fresh one, but you reuse the one you already made. This pattern is called "Lazy Initialization" and is widely used in iOS development.

Resources

https://developer.apple.com/library/ios/documentation/general/conceptual/DevPedia-CocoaCore/Delegation.html