Protocol Oriented Programming in Swift

Protocols

Protocols are blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality. It's an interface. A powerful feature of Swift language.

```
protocol TableCellProtocol {
    func cellButtonTapped()
}
```

Protocol OP is a new approach for programming which you decorate your classes, structs or enums using protocols. Swift does not support multiple inheritance, therefore problem pops out when you want to add multiple abilities to your class. Protocol OP lets you to add abilities to a class or struct or enum with protocols which supports multiple implementations.

OOP should be enough?

Modelling abstractions using classes relies on inheritance. Subclasses will have the ability of their super classes. A subclass can override that ability, add specific one. OOP works perfect till you need another ability from another class. There is no multiple inheritance in Swift. But protocols serve as blueprints rather than parents. A protocol models abstraction by describing what the implementation types shall implement. This is the key difference between OOP and Protocol OP. And two other pros over OOP are here:

- Types can conform more than one protocol. This brings perfect flexibility.
- Protocols also can be extended and adopted by classes, structs and enums.

What? Extending Protocols? Inheritance?

Protocols are extendable. Lets see it in an example:
// I extend Bombable protocol from Fireable. With this new protocol, // I wanted to add maxDistance feature.

```
protocol Bombable: Fireable {
    var maxDistance: Double { get }
}

// Then I create brand new Weapon, Howitzer

struct Howitzer: Weapon, Bombable {
    let name = "Howitzer"
    let magazineSize = 1
    let maxDistance = 1000 // in km
}
```

// So this extensibility made my code base more flexible. Consider, // I do not want maxDistance in Fireable but Bombable.

Why implement Protocol Oriented Programming in Swift?

In several other languages which have interfaces like C++, C # or others which have protocols like objective-c, these interfaces and protocols don't implement anything

from their protocols, these protocols just work as a blueprint to specify the requirement that a class should adopt. Instead Swift Protocols have a rule changer feature: Protocol Extensions.

Other advantage of POP is objects can conform to multiple Protocols in Swift. In OOP objects can inherit from just one superclass.

POP and OOP are not mutually exclusive, they can compliment each other.

Difference between Delegate and Datasource

A **delegate** type object responds to actions that another object takes.
i.e the UITableViewDelegate protocol has methods such as didSelectRowAtIndexPath for performing actions upon a user selecting a particular row in a table and willDisplayCell which called before delegate use cell to draw row.

DataSource type object gives data to another object.
i.e UITableViewDataSource protocol has methods such as cellForRowAtIndexPath and numberOfSectionInTableView dictating what should be displayed in the table.