# OOP-MVC-Design Pattern

# Lập trình hướng đối tượng (OOP)

- OOP là một kỹ thuật lập trình cho phép dev tạo ra các đối tượng trong code, trừu tượng hóa các đối tượng trong thực tế.
- OOP có 4 tính chất cơ bản:
  - Tính trừu tượng
  - Tính kế thừa
  - Tính đa hình
  - Tính đóng gói

# Tính kế thừa

- Cho phép một đối tượng có thể có sẵn các đặc tính mà đối tượng khác đã có thông qua kế thừa. Điều này cho phép các đối tượng chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại

```
class Vehicle {
    func getName() {
        print("Vehicle")
    }
}

class Bikecycle:Vehicle {
}

var bike:Bikecycle = Bikecycle()
bike.getName()
```

# Tính đa hình

- Thể hiện thông qua việc gửi các **thông điệp** (*message*). Việc gửi các thông điệp này có thể so sánh như việc gọi các hàm bên trong của một đối tượng. Các phương thức dùng trả lời cho một thông điệp sẽ tùy theo đối tượng mà thông điệp đó được gửi tới sẽ có phản ứng khác nhau.

```
class Vehicle {
    func getName() {
        print("Vehicle")
    }
}

class Bikecycle:Vehicle {
    override func getName() {
        print("Bikecycle")
    }
}

var listVehicle:[Vehicle] = [Vehicle(), Bikecycle()]
var vehicle:Vehicle = listVehicle[1]
vehicle.getName()
```

# Tính đóng gói

- Chỉ có các phương thức nội tại của đối tượng cho phép thay đổi trạng thái của nó. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã. Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.

```
class Vehicle {
    private var material = "material"
    func getName() {
        print("Vehicle")
    }
}


var vehicle:Vehicle = Vehicle()
print(vehicle.material)
```

# Tính trừu tượng

- Dữ liệu trừu tượng: là việc bỏ qua hay không chú ý đến một số khía cạnh của thông tin, chỉ tập trung vào những cốt lõi cần thiết.
- Lớp trừu tượng: một đối tượng ban đầu có thể có một số đặc điểm chung cho nhiều đối tượng khác như là sự mở rộng của nó nhưng bản thân đối tượng ban đầu này có thể không có các biện pháp thi hành.

- Ví dụ: Một cái xe máy khi thành một Class sẽ chỉ cần một vài thuộc tính cần thiết trở thành property.

# Protocol

- Protocol là thành phần trừu tượng cho phép bạn khai báo danh sách các phương thức và thuộc tính nhưng không cài đặt các phương thức này (Interface).
- Vì sao cần Protocol: Vì IOS không hỗ trợ đa kế thừa ngoài ra còn dùng Protocol để "uỷ quyền" chức năng.

```
protocol: <Tên protocol>  {
//  Khai báo các property và method.
}
```

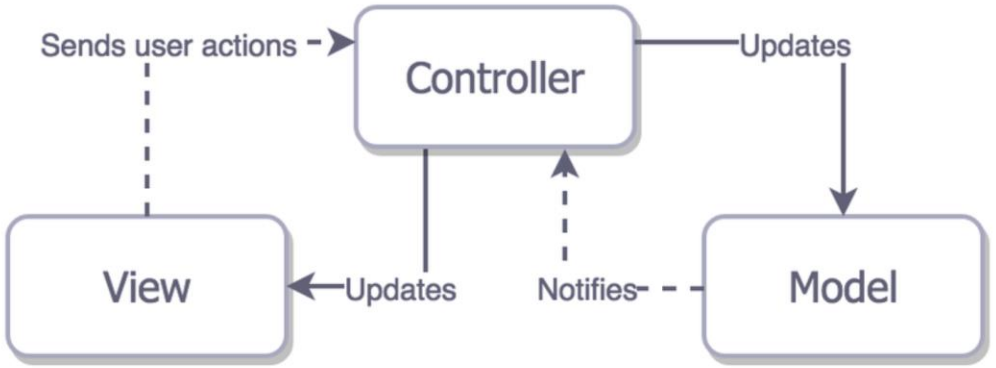
```
protocol CustomViewProtocol {
    func getInfo()
}
```

# Protocol

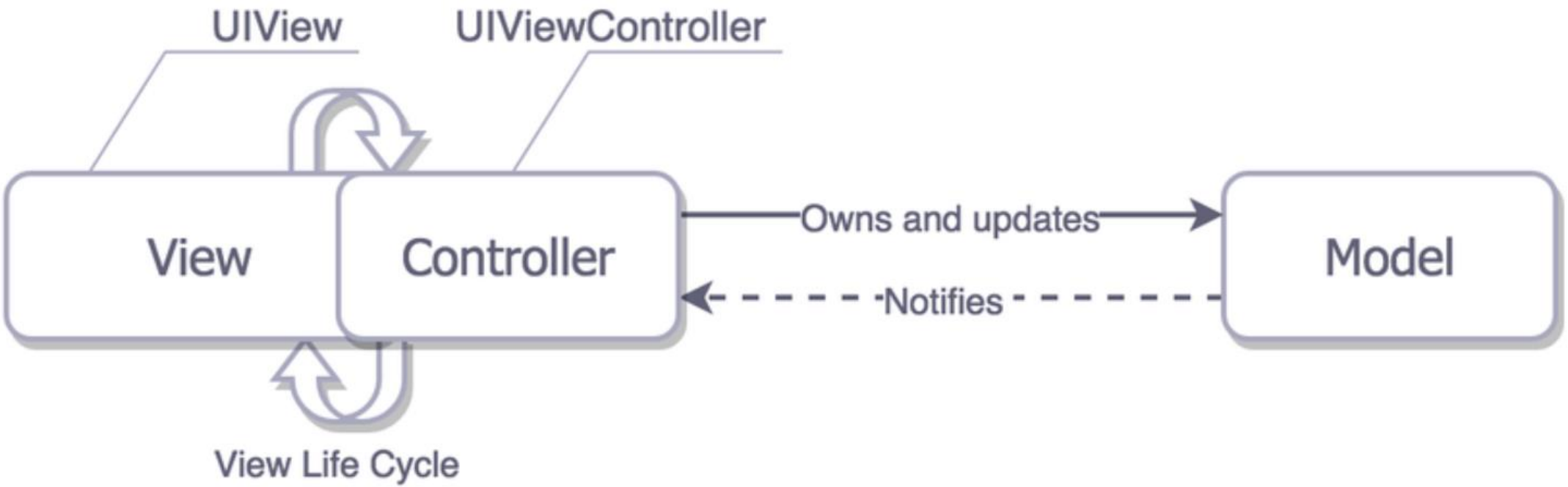
```
protocol CustomViewProtocol {
    func getInfo()
}

class CustomView1: UIView, CustomViewProtocol {
    func getInfo() {

    }
}

class CustomView2: UIView, CustomViewProtocol {
    func getInfo() {

    }
}

var listCustomView:[CustomViewProtocol] = [CustomView1(), CustomView2()]
```

# MVC

- **MVC** là viết tắt của 3 từ Model, View và Controller:
  - **Models**: Là đối tượng có vai trò định nghĩa, tổ chức lưu trữ và thao tác trực tiếp với dữ liệu
  - **Views**: Là đối tượng chịu trách nhiệm về hiển thị giao diện hay nói cách khác đó là tầng mà người dùng quan sát và tương tác trực tiếp.
  - **Controller**: Là bộ điều kiển trung gian có vai trò quản lý, điều phối tất cả công việc. Nó truy cập dữ liệu từ Model và quyết định hiển thị nó trên View. Khi có một sự kiện được truyền xuống từ View, Controller quyết định cách thức xử lý: có thể là tương tác với dữ liệu, thay đổi giao diện trên View



# MVC Trong IOS



# Design Pattern

- Design Pattern sẽ cung cấp cho bạn các "mẫu thiết kế", giải pháp để giải quyết các vấn đề chung, thường gặp trong lập trình. Các vấn đề mà bạn gặp phải có thể bạn sẽ tự nghĩ ra cách giải quyết nhưng có thể nó chưa phải là tối ưu. Design Pattern giúp bạn giải quyết vấn đề một cách tối ưu nhất, cung cấp cho bạn các giải pháp trong lập trình OOP.

- Ví dụ : Bạn muốn tạo một đối tượng để cả project dùng chung, chỉ khởi tạo một lần.

# Design Pattern

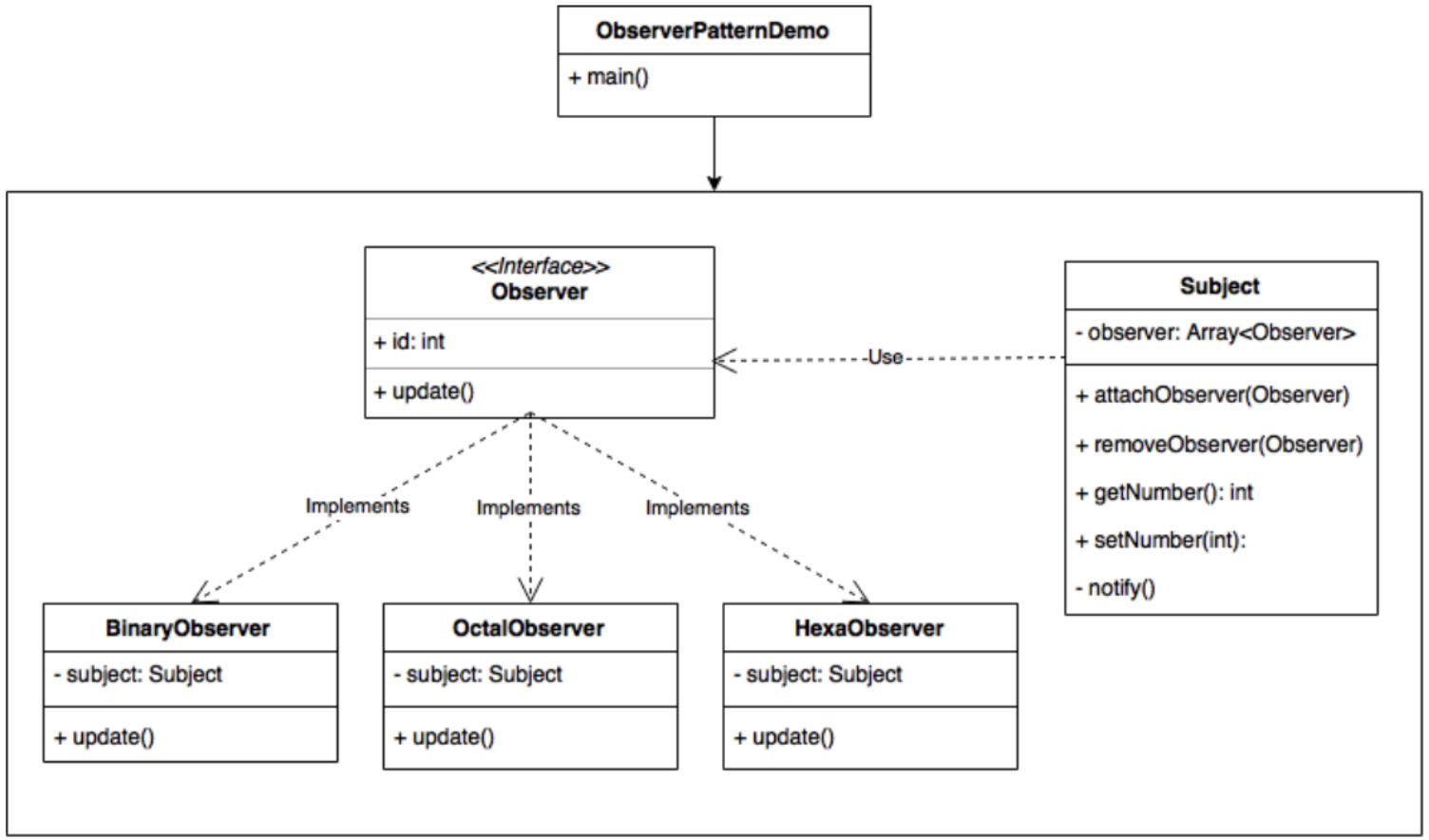
- Design Pattern được phân làm 3 loại cơ bản
  - Nhóm khởi tạo : Việc khởi tạo đối tượng
  - Nhóm hành vi : Hành vi của đối tượng
  - Nhóm cấu trúc : Quan hệ giữ các đối tượng
- Ngoài ra gần đây xuất hiện thêm vài loại khác.
- Trong IOS có 3 loại Design Pattern dùng nhiều nhất:
  - Singleton : Nhóm khởi tạo
  - Observer : Nhóm hành vi
  - Delegation : Nhóm khác

# Singleton

- Singleton tạo một đối tượng để cả project dùng chung, chỉ khởi tạo một lần.

```
class LocationManager{

    static let shared = LocationManager()
    var locationGranted: Bool?
    private init(){}

    func requestForLocation(){
        locationGranted = true
        print("Location granted")
    }
}

LocationManager.shared.requestForLocation()
```

# Observer


ObserverPatternDemo
+ main()
<<Interface>>
Observer
+ id: int
+ update()
Use
Subject
- observer: Array<Observer>
+ attachObserver(Observer)
+ removeObserver(Observer)
+ getNumber(): int
+ setNumber(int):
- notify()
Implements
Implements
Implements
BinaryObserver
- subject: Subject
+ update()
OctalObserver
- subject: Subject
+ update()
HexaObserver
- subject: Subject
+ update()

# Delegation

- Delegation dùng protocol để mở rộng một số tính năng cho đối tượng.