MVC

Model-View-Controller (MVC) is a fundamental architectural pattern in iOS development..

What is MVC?

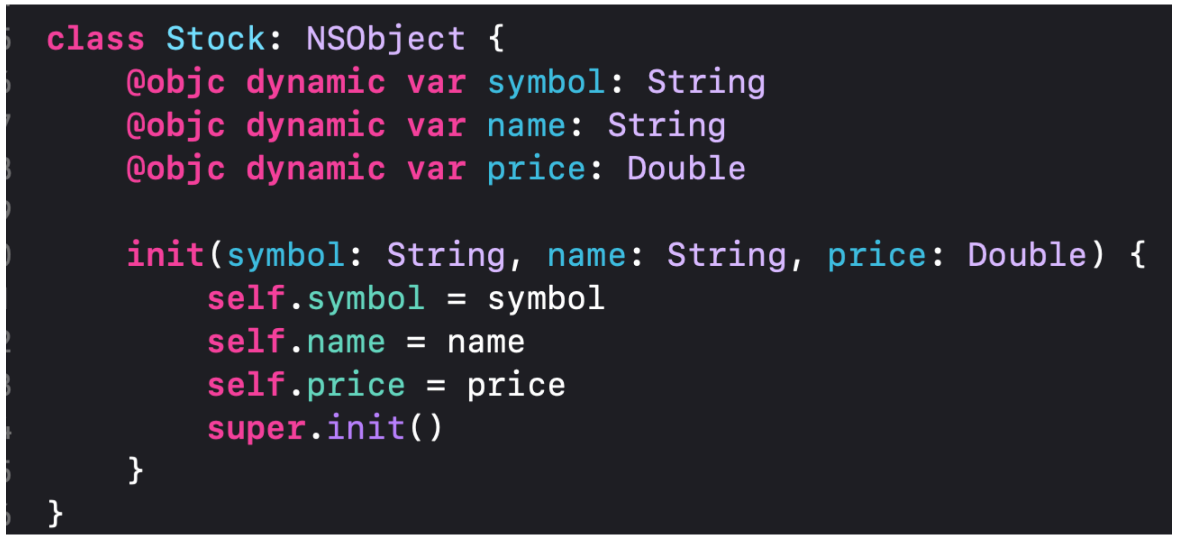
- MVC stands for Model-View-Controller
- Architectural pattern used in iOS development
- Separates application logic into three interconnected components

Components of MVC?

1. **Model**: Manages data and business logic
2. **View**: Displays information to the user
3. **Controller**: Coordinates between Model and View

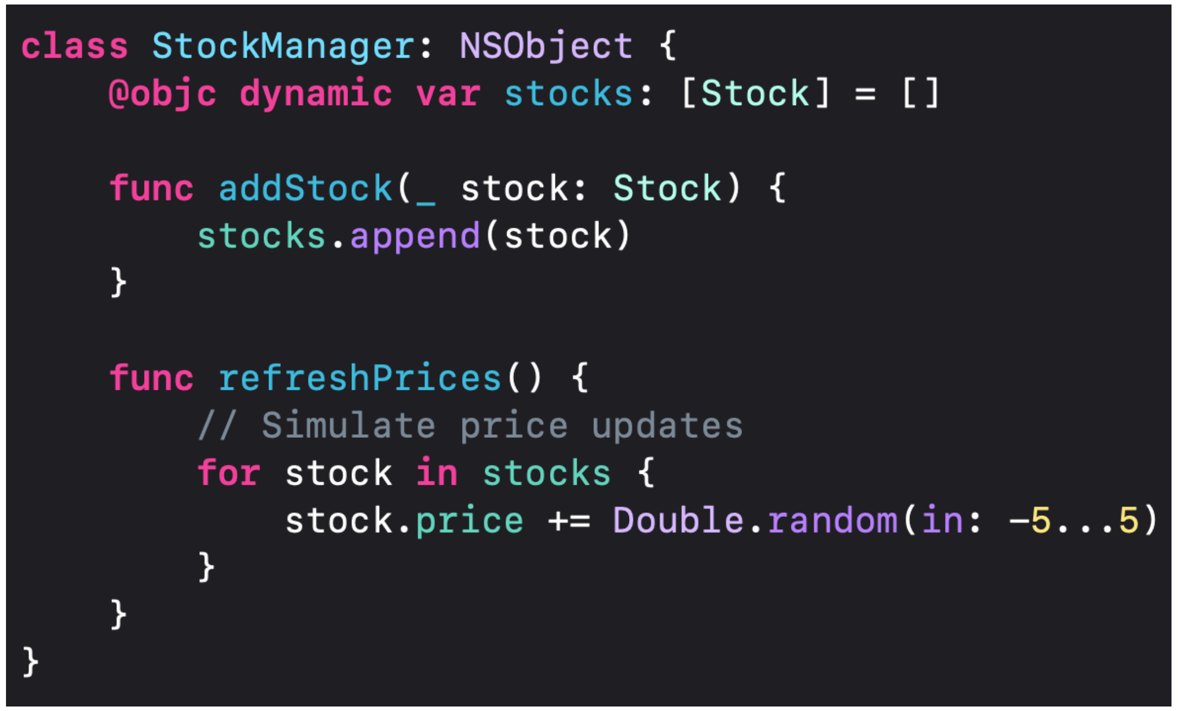
Model

- Represents data and business logic
- Independent of the user interface
- Notifies observers about changes
- Uses Key-Value Observing (KVO) for change notifications



```swift
class Stock: NSObject {
    @objc dynamic var symbol: String
    @objc dynamic var name: String
    @objc dynamic var price: Double

    init(symbol: String, name: String, price: Double) {
        self.symbol = symbol
        self.name = name
        self.price = price
        super.init()
    }
}
```



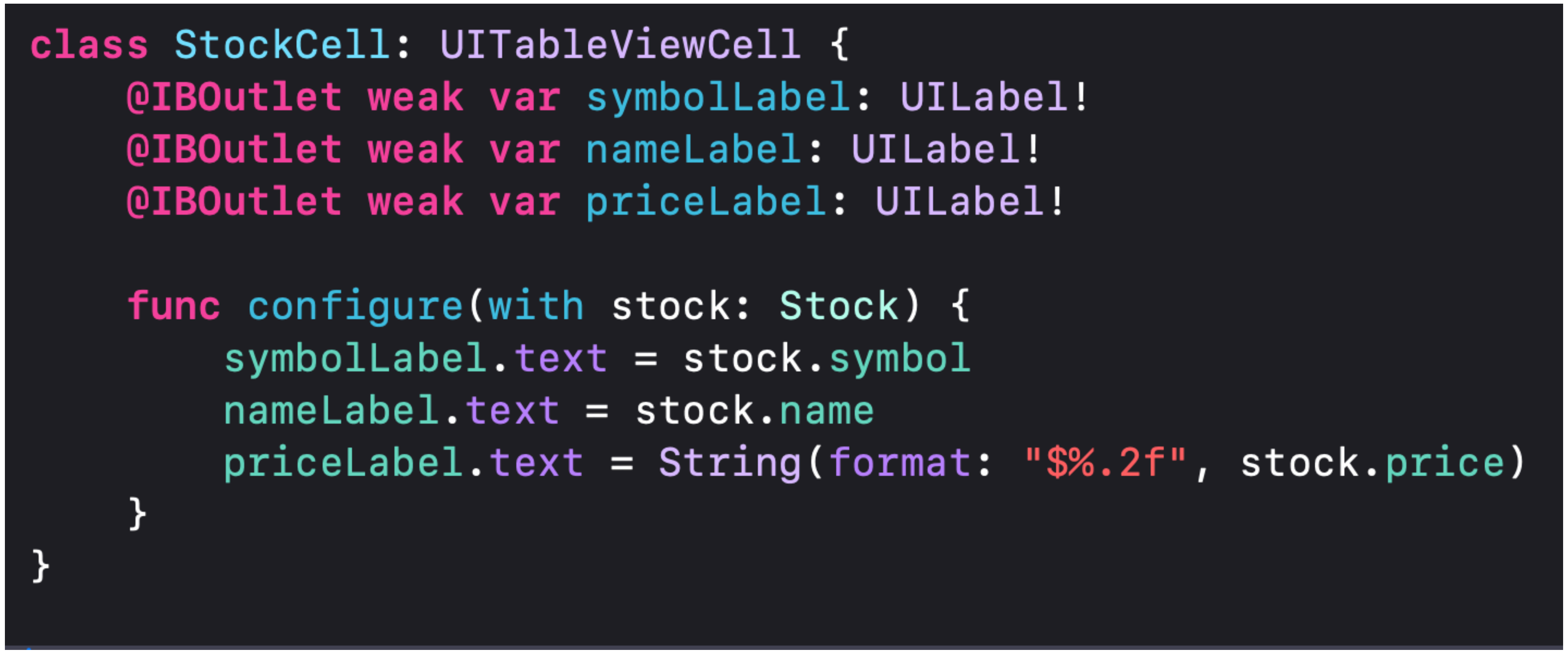
```swift
class StockManager: NSObject {
    @objc dynamic var stocks: [Stock] = []

    func addStock(_ stock: Stock) {
        stocks.append(stock)
    }

    func refreshPrices() {
        // Simulate price updates
        for stock in stocks {
            stock.price += Double.random(in: -5...5)
        }
    }
}
```

View

- Presents data to the user
- Passive; doesn't contain business logic
- Updates when the model changes

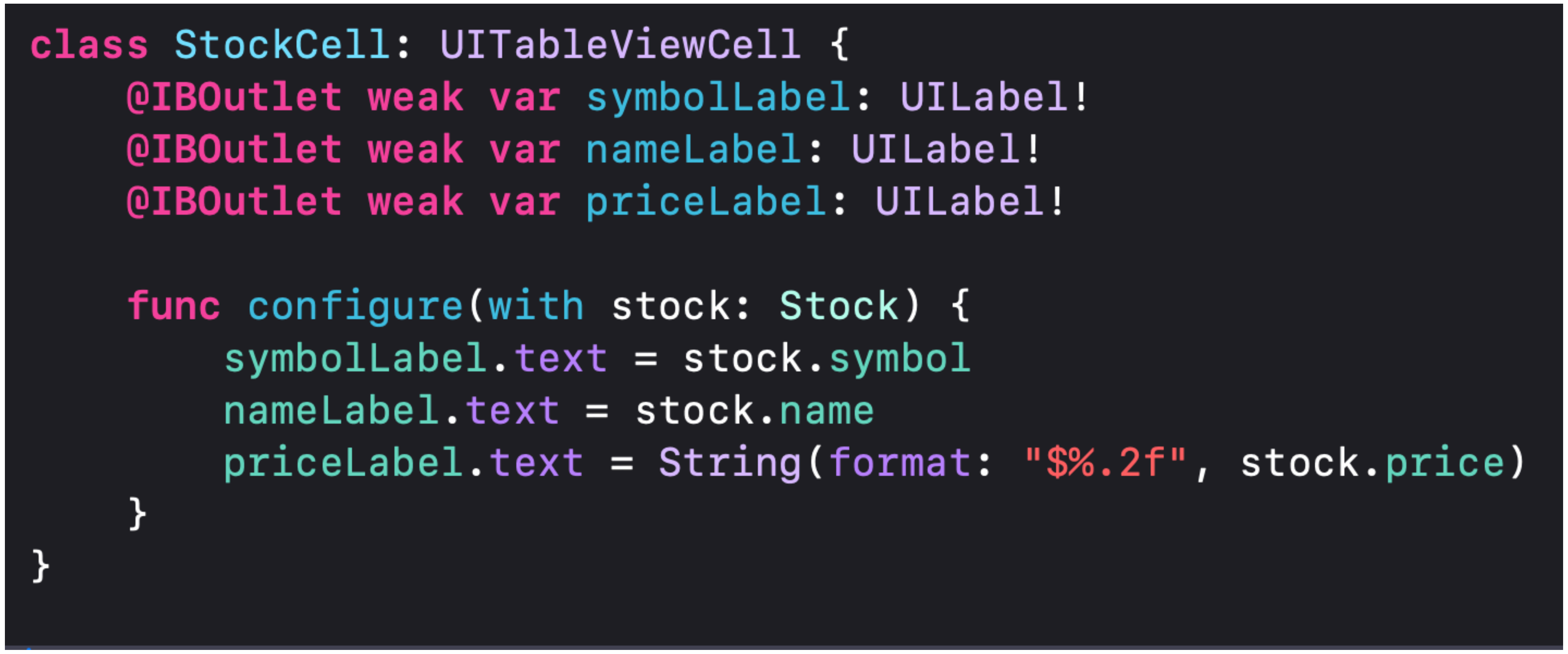


```swift
class StockCell: UITableViewCell {
    @IBOutlet weak var symbolLabel: UILabel!
    @IBOutlet weak var nameLabel: UILabel!
    @IBOutlet weak var priceLabel: UILabel!

    func configure(with stock: Stock) {
        symbolLabel.text = stock.symbol
        nameLabel.text = stock.name
        priceLabel.text = String(format: "$%.2f", stock.price)
    }
}
```

View

- Presents data to the user
- Passive; doesn't contain business logic
- Updates when the model changes



```swift
class StockCell: UITableViewCell {
    @IBOutlet weak var symbolLabel: UILabel!
    @IBOutlet weak var nameLabel: UILabel!
    @IBOutlet weak var priceLabel: UILabel!

    func configure(with stock: Stock) {
        symbolLabel.text = stock.symbol
        nameLabel.text = stock.name
        priceLabel.text = String(format: "$%.2f", stock.price)
    }
}
```

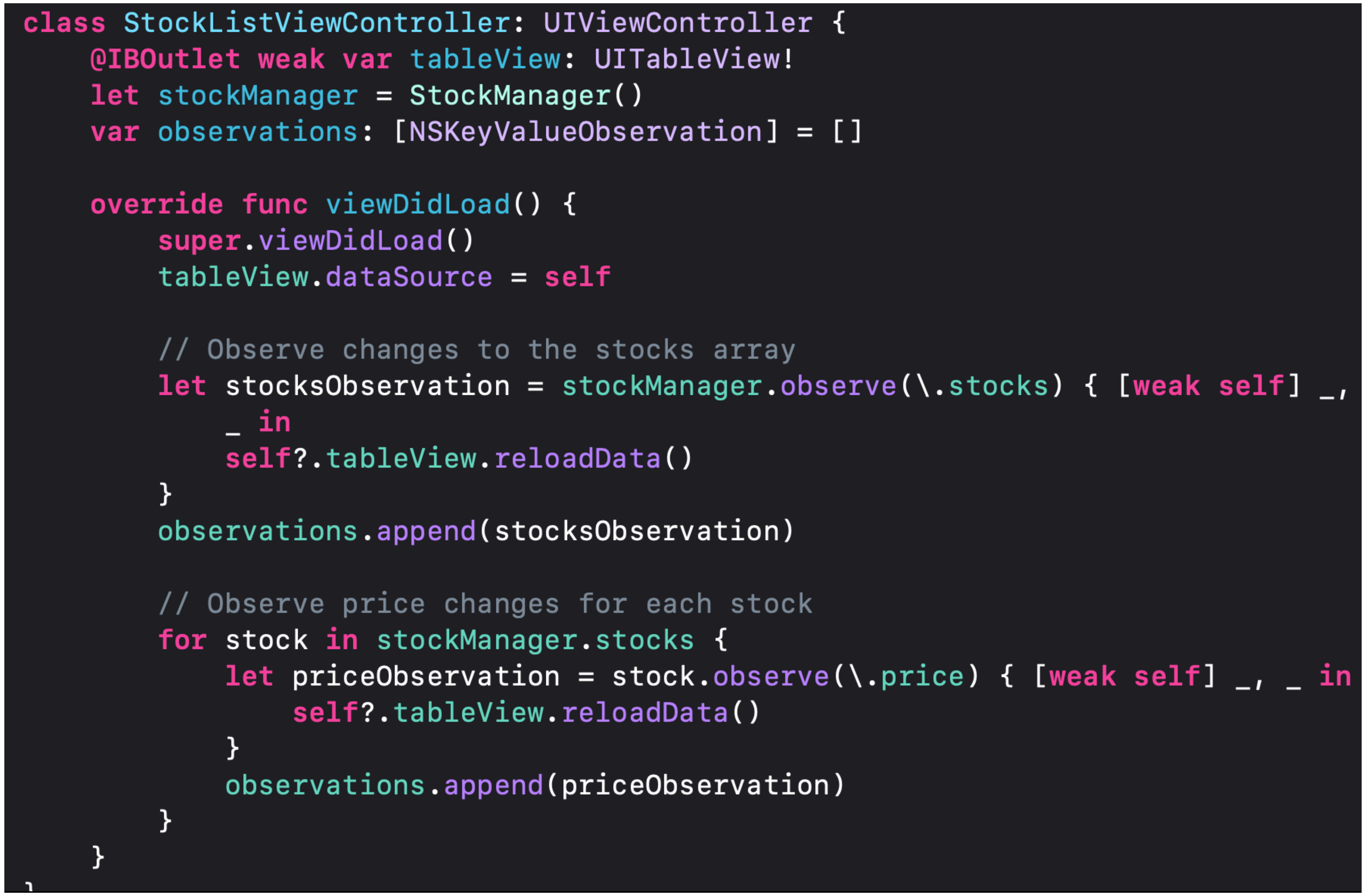
Controller

Mediates between Model and View

Handles user input and updates Model

Updates View when Model changes

Uses KVO to observe Model changes



```
class StockListViewController: UIViewController {
    @IBOutlet weak var tableView: UITableView!
    let stockManager = StockManager()
    var observations: [NSKeyValueObservation] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        tableView.dataSource = self

        // Observe changes to the stocks array
        let stocksObservation = stockManager.observe(\.stocks) { [weak self] _,
            _ in
            self?.tableView.reloadData()
        }
        observations.append(stocksObservation)

        // Observe price changes for each stock
        for stock in stockManager.stocks {
            let priceObservation = stock.observe(\.price) { [weak self] _, _ in
                self?.tableView.reloadData()
            }
            observations.append(priceObservation)
        }
    }
```

Benefits of MVC

- Separation of Concerns
- Improved Testability
- Enhanced Reusability
- Better Scalability
- Easier Maintenance
- Flexibility

MVC Flow in UIKit

- User interacts with the View
- View informs the Controller of the interaction
- Controller updates the Model if necessary
- Model notifies the Controller of changes (via KVO)
- Controller updates the View with new data

Common Pitfalls in UIKit MVC

Massive View Controller: Controllers taking on too much responsibility

Tight Coupling: Components becoming too interdependent

Lack of True Separation: Blurring lines between MVC components