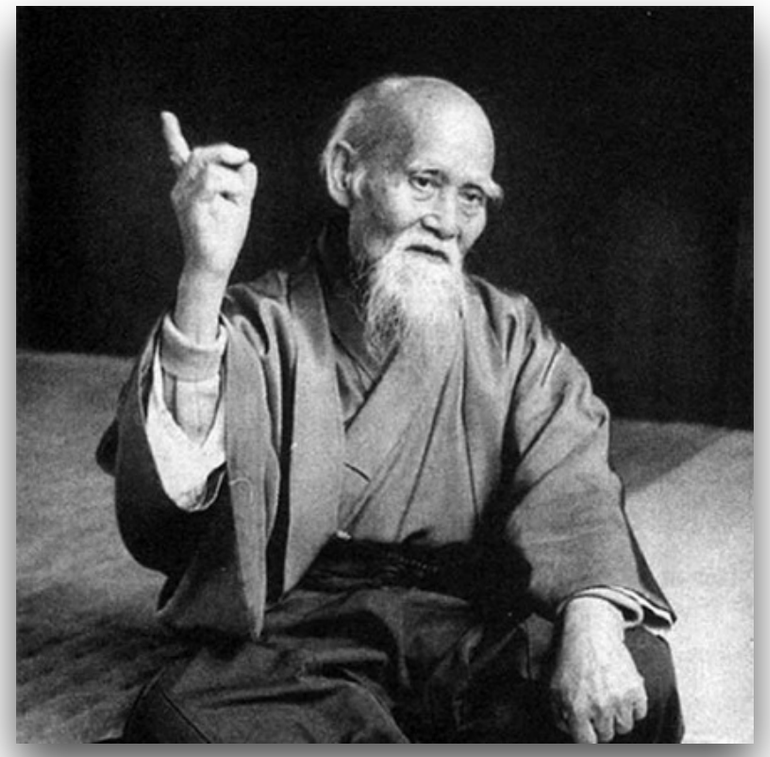
Лекция 6

Архитектура iOS приложений

Какой код хороший?



Код должен быть…

1. Рабочим и отказоустойчивым
2. Читаемым
3. Поддерживаемым
4. Тестируемым

Принципы

DRY Don’t Repeat Yourself

Если вдруг вы ловите себя на том, что этот код вы уже писали/встречали раньше, остановитесь, подумайте, и не повторяйте себя

Принципы

KISS

Keep it simple, stupid

Принципы

YAGNI

You ain’t gonna need it?

Принципы

SOLID

Single Responsibility Principle

Принцип единой ответственности: у каждого класса должна быть только одна причина его изменять.

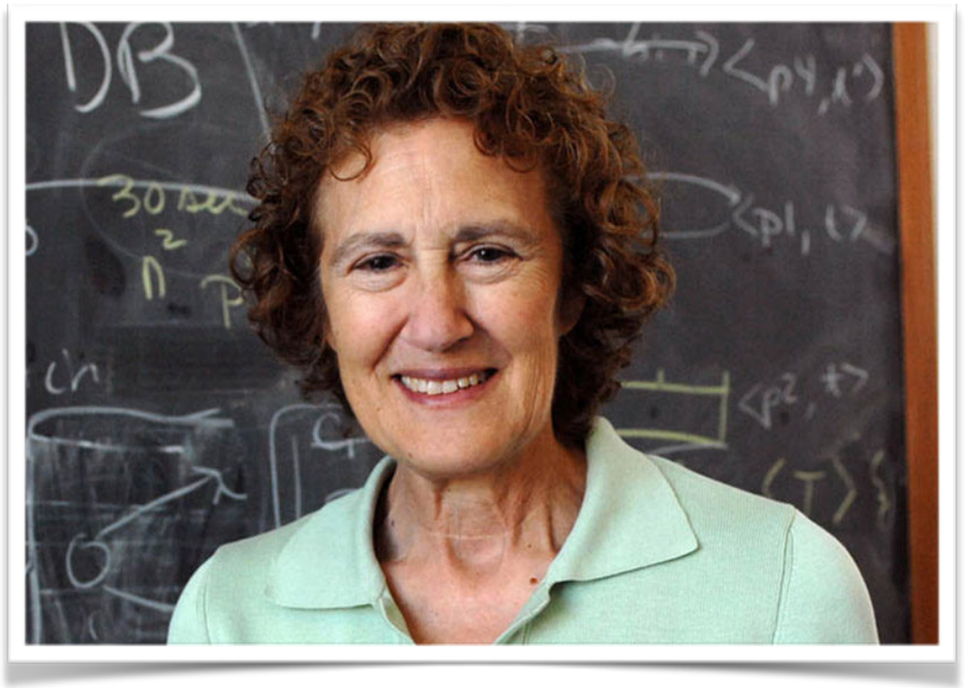
Open/Closed Principle

Принцип открытости/закрытости: Классы, модули, функции должны быть открыты для расширения, но закрыты для изменения

Open/Closed Principle

```
class SizePrinter {
    func printSize(of shape: ISizeable) {
        print(shape.size)
    }
}


protocol ISizeable {
    var size: CGSize { get }
}

struct Circle: ISizeable {
    let radius: CGFloat
}


struct Rectangle: ISizeable {
    let width: CGFloat
    let height: CGFloat
}
```

Liskov Substitution Principle



Принцип подстановки Лисков:

Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом.

Наследуйтесь правильно!

Нарушение принципа Лисков

```
class Bird {
    var wings: Any
    var beak: Any
    func run(to point: CGPoint)
    func layDownAnEgg()
    func fly(to point: CGPoint)
}

class Duck: Bird {
    func fly(to point: CGPoint) {
        self.startFlyAnimation()
        self.position = point
    }
    //....
}

class Hen: Bird {
    func fly(to point: CGPoint) {
        // Курица не умеет летать, поэтому реализация пустая
    }
    //....
}
```

Interface Segregation Principle

Принцип разделения интерфейсов:
Много отдельных интерфейсов лучше, чем один универсальный.

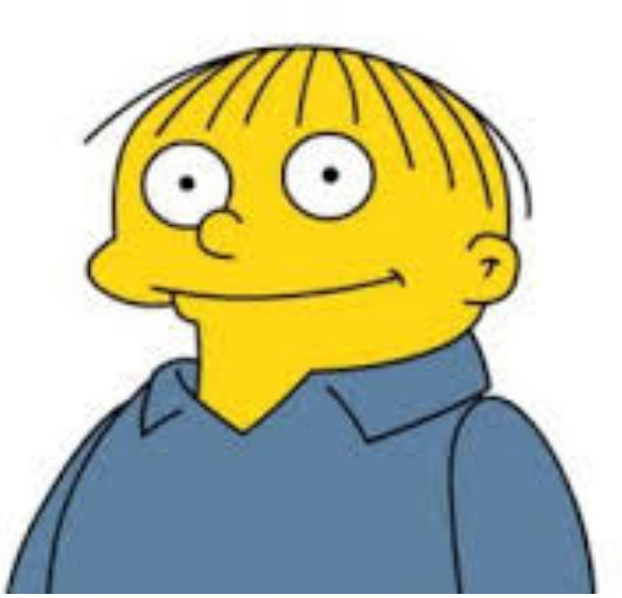
Нарушение принципа разделения интерфейсов

```swift
protocol ICardManager {
    func createCard() -> Card?
    func readCard() -> Card?
    func update(card: Card)
    func delete(card: Card)
}

class UnauthorizedUser: ICardManager {
    // Неавторизованный пользователь может просматривать карты
    func readCard() -> Card? {
        let card = ... // логика чтения карты
        return card
    }

    // Больше он ничего не может делать
    func createCard() -> Card? { return nil }
    func update(card: Card) { }
    func delete(card: Card) { }
}

class Admin: ICardManager {
    // Админ может все. Реализованы все методы.
}
```

Нарушение принципа разделения интерфейсов

```
struct Card { }

protocol ICardCreator {
    func createCard() -> Card
}

protocol ICardReader {
    func readCard() -> Card
}

protocol ICardUpdater {
    func update(card: Card)
}

protocol ICardTrasher {
    func delete(card: Card)
}

class UnauthorizedUser: ICardReader {
    // Неавторизованный пользователь может просматривать карты
    func readCard() -> Card? {
        let card = ... // логика чтения карты
        return card
    }
}

class Admin: ICardCreator, ICardReader, ICardUpdater, ICardTrasher {
    // Админ может все. Реализованы все методы.
}
```

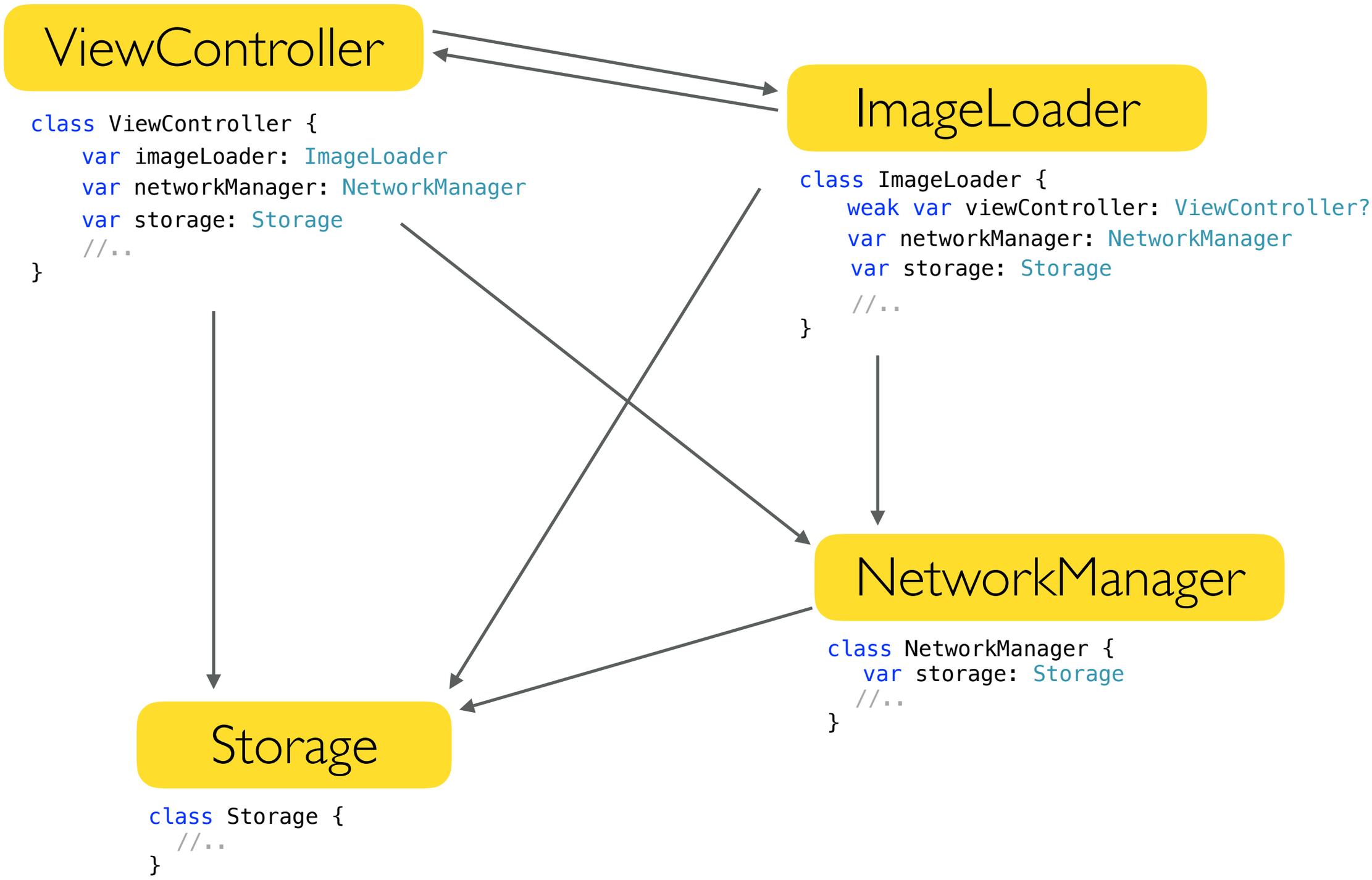
STUPID код.



Что делает код «тупым»?

1. Singleton
2. Tight Coupling – жесткая связность.
3. Untestability – нетестируемость.
4. Premature Optimization – преждевременная оптимизация
5. Indescriptive Naming – плохие имена.
6. Duplication – повторение кода.

Связность


ViewController
class ViewController {
var imageLoader: ImageLoader
var networkManager: NetworkManager
var storage: Storage
//..
}
ImageLoader
class ImageLoader {
weak var viewController: ViewController?
var networkManager: NetworkManager
var storage: Storage
//..
}
NetworkManager
class NetworkManager {
var storage: Storage
//..
}
Storage
class Storage {
//..
}

ViewController

```
class ViewController {
    var imageLoader: ImageLoader
    var networkManager: NetworkManagerInterface
    var storage: StorageInterface
    //..
}
```

ImageLoader

```
class ImageLoader {
    weak var viewController: ViewController?
    var networkManager: NetworkManagerInterface
    var storage: StorageInterface
    //..
}
```

NetworkManagerInterface

StorageInterface

NetworkManager

```
class NetworkManager {
   var storage: StorageInterface
}
```

Storage

```
class Storage: StorageInterface {

}
```

CoreDataStorage

```
class CoreDataStorage {

}
```

Вывод № раз

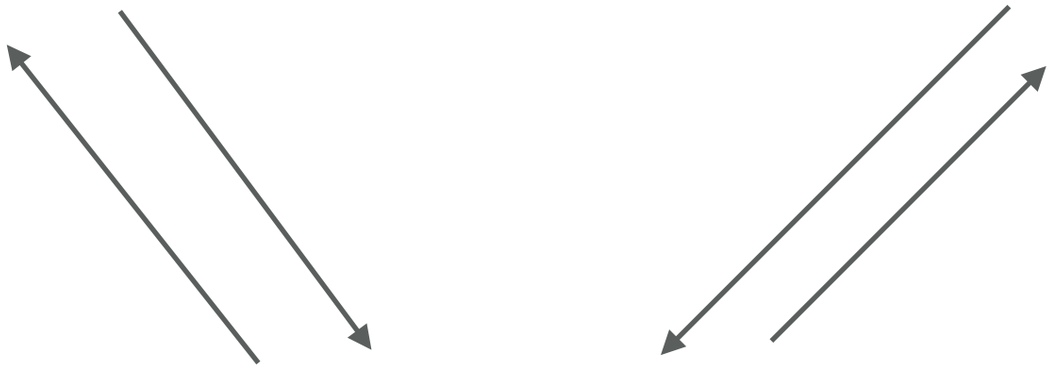
Использовать протоколы ✓

ViewController

```
class ViewController {
    var imageLoader: ImageLoader
    var networkManager: NetworkManagerInterface
    var storage: StorageInterface
    //..
}
```

PinguinViewController

```
class PinguinViewController {
    var imageLoader: ImageLoader
    //..
}
```



ImageLoader

```
class ImageLoader {
    weak var viewController: ViewController?
    var networkManager: NetworkManagerInterface
    var storage: StorageInterface
    //..
}
```

ImageLoaderDelegate

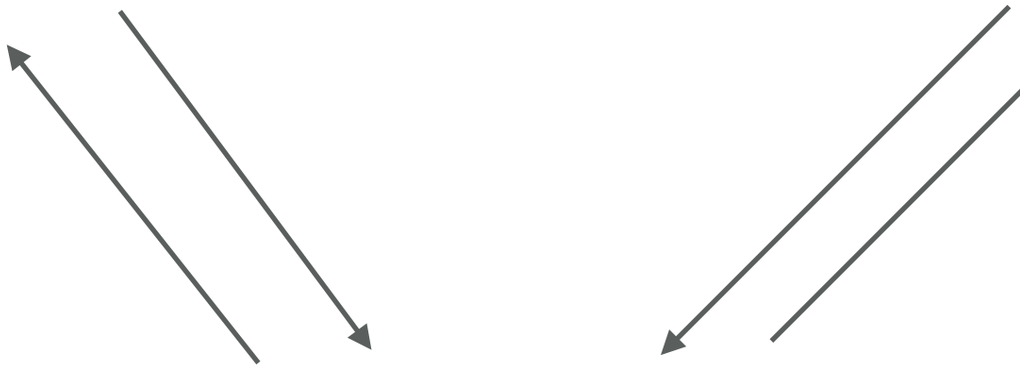
ViewController

```
class ViewController {
    var imageLoader: ImageLoader
    var networkManager: NetworkManagerInterface
    var storage: StorageInterface
    //..
}
```

ImageLoaderDelegate

PinguinViewController

```
class PinguinViewController {
    var imageLoader: ImageLoader
    //..
}
```

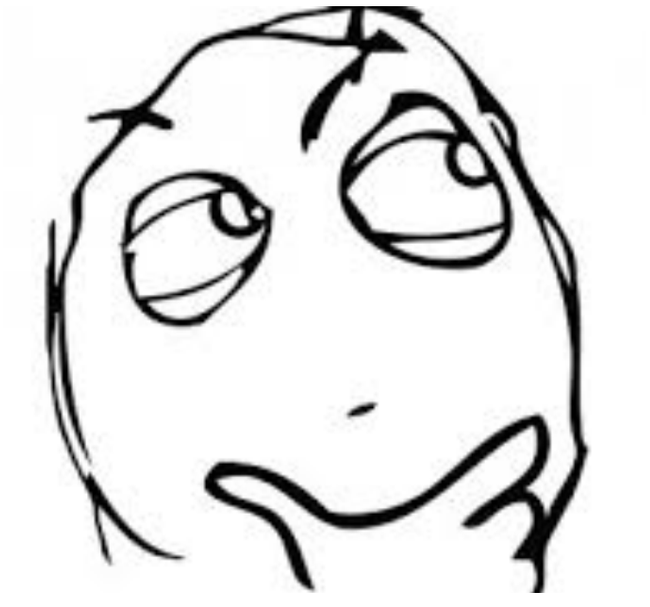


ImageLoader

```
class ImageLoader {
    weak var delegate: ImageLoaderDelegate?
    var networkManager: NetworkManagerInterface
    var storage: StorageInterface
    //..
}
```

Вывод № 2

Однонаправленные связи



✓

SOA – Сервисно-ориентированная архитектура

Presentation Layer

VC 1

VC 2

Delegates&
Datasources

Animators

Service Layer

S1Protocol

Service 1

S2Protocol

Service 2

Core Layer

NMProtocol

Network
Manager

DBProtocol

Database
Manager

ParserProtoc

Parser

OperationProto

Operation

Dependency inversion principle

Принцип инверсии зависимостей:
Модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те и другие должны зависеть от абстракций.

SOLID

А как же...

MVC, MVP, MVVM, VIPER, CleanSwift и т.д. ?

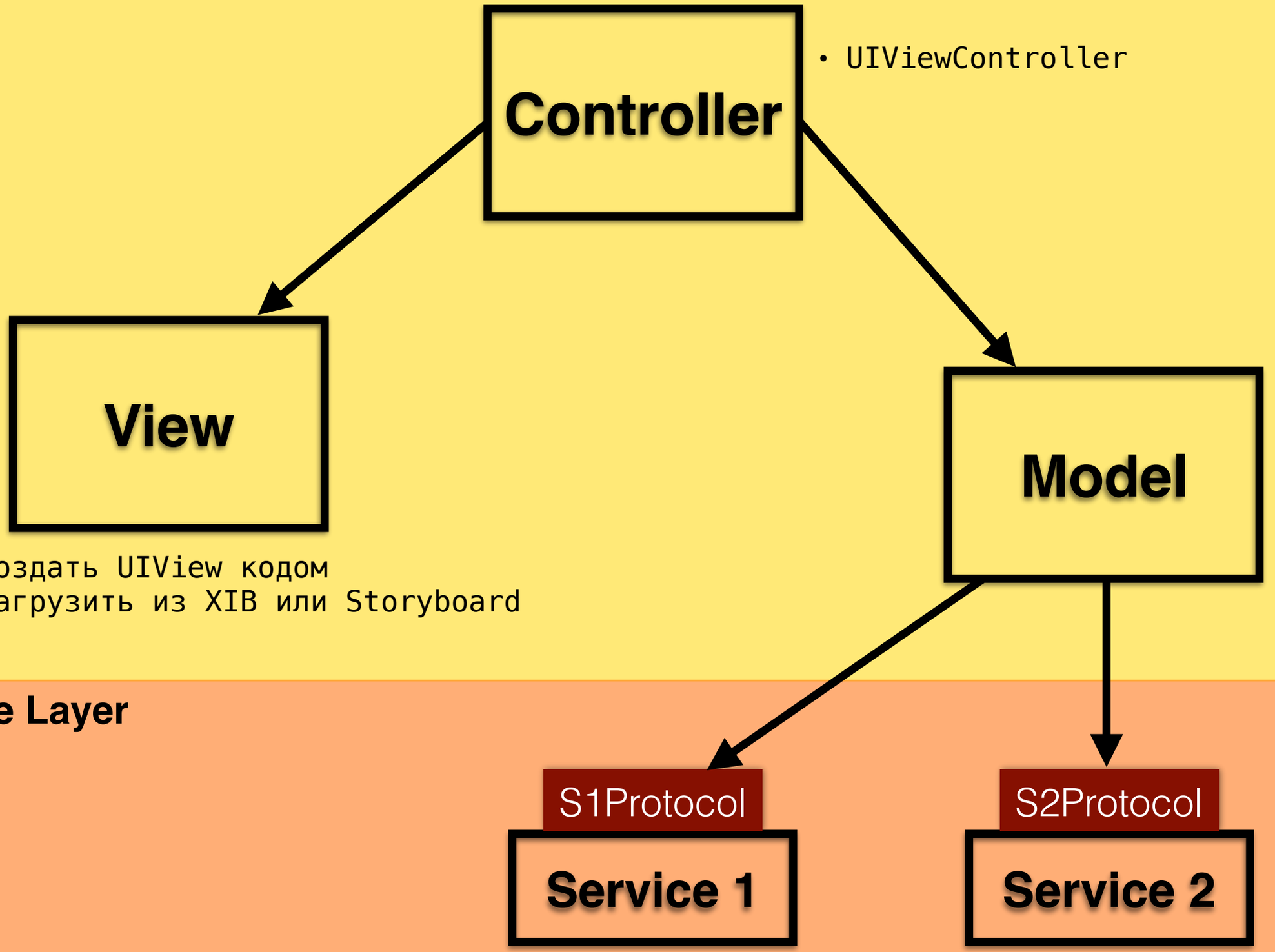
А как же...

MVC, MVP, MVVM, VIPER, CleanSwift и т.д. ?

Presentation Layer

MVC – Model, View, Controller

Presentation Layer



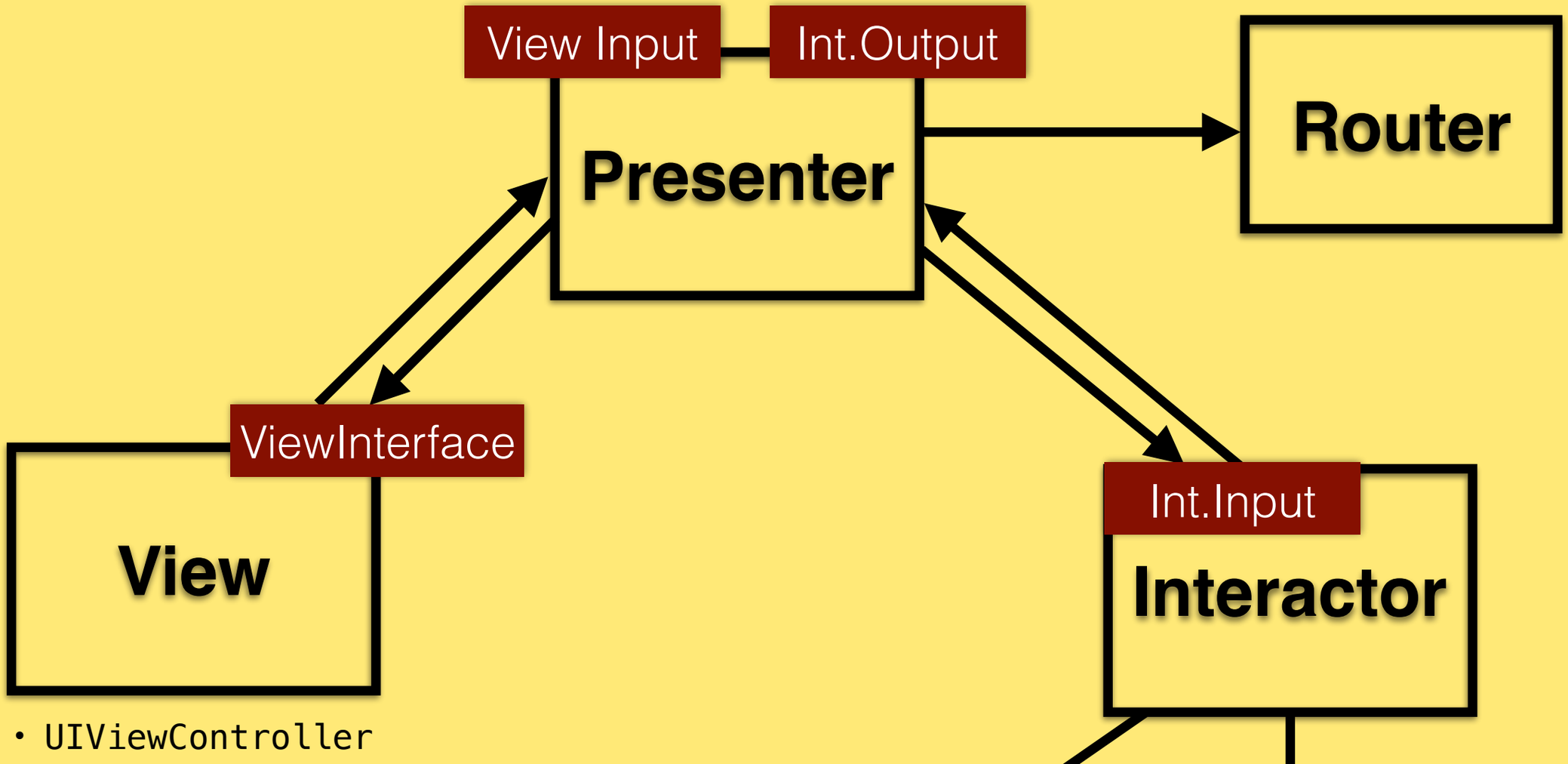
- `UIViewController`

- `Создать UIView кодом`
- `Загрузить из XIB или Storyboard`

Service Layer

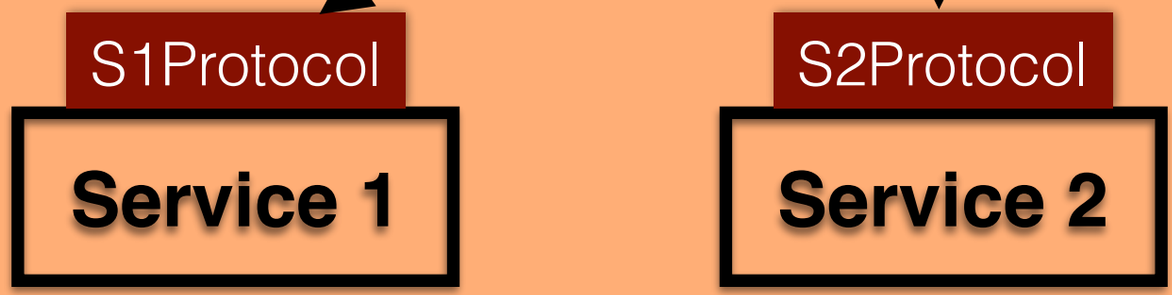
MVC – Model, View, Controller

Presentation Layer



- `UIViewController`

Service Layer



There is no Silver bullet

Спасибо за внимание!

Алексей Зверев