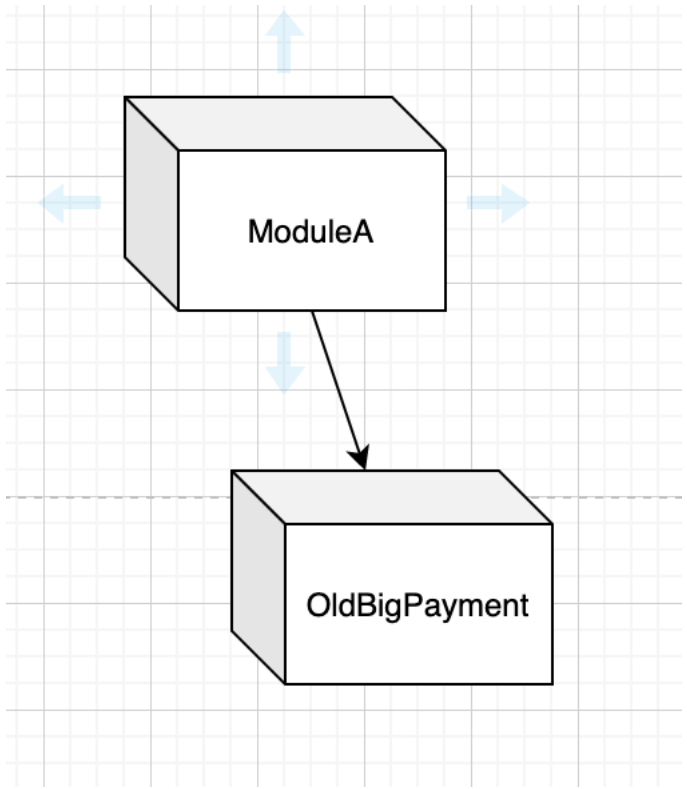
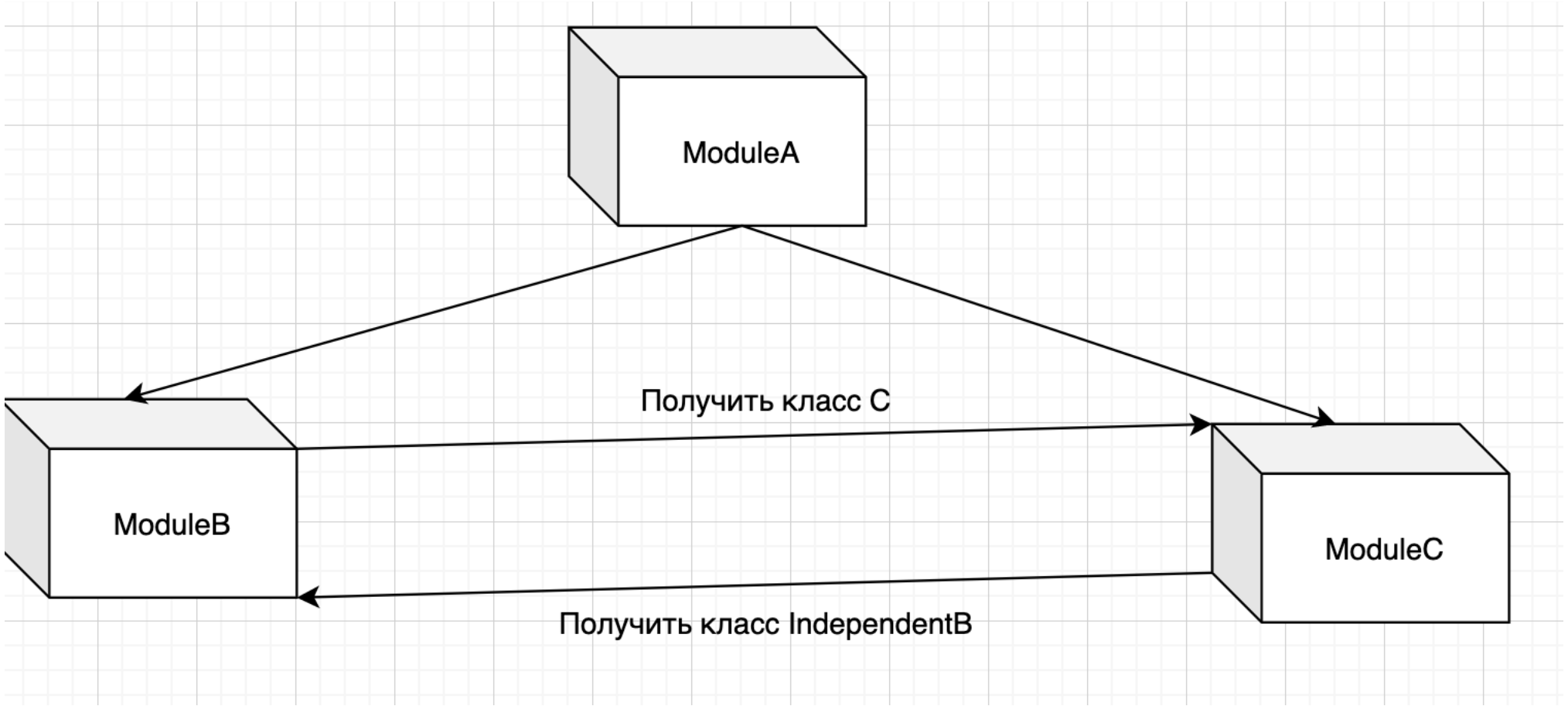
Зачем и как использовать DI

By Core Batyas

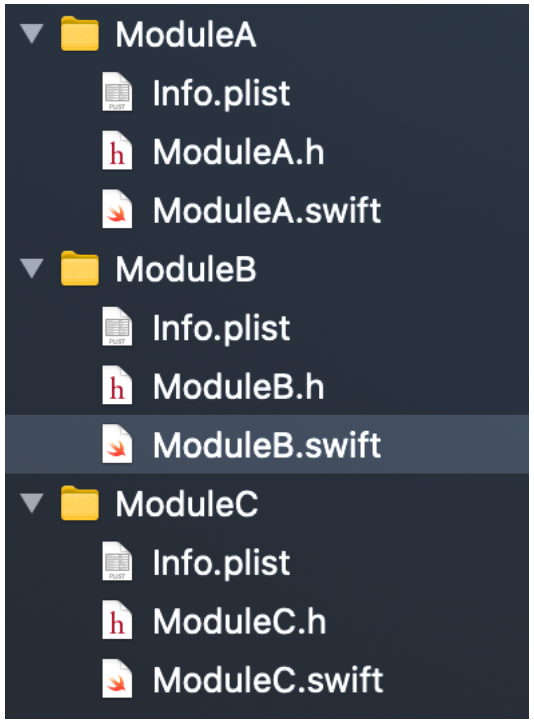
Стартовая картина до начала рефакторинга



Середина рефакторинга, появляется ошибка циклических зависисмостей



Пример циклической зависимости



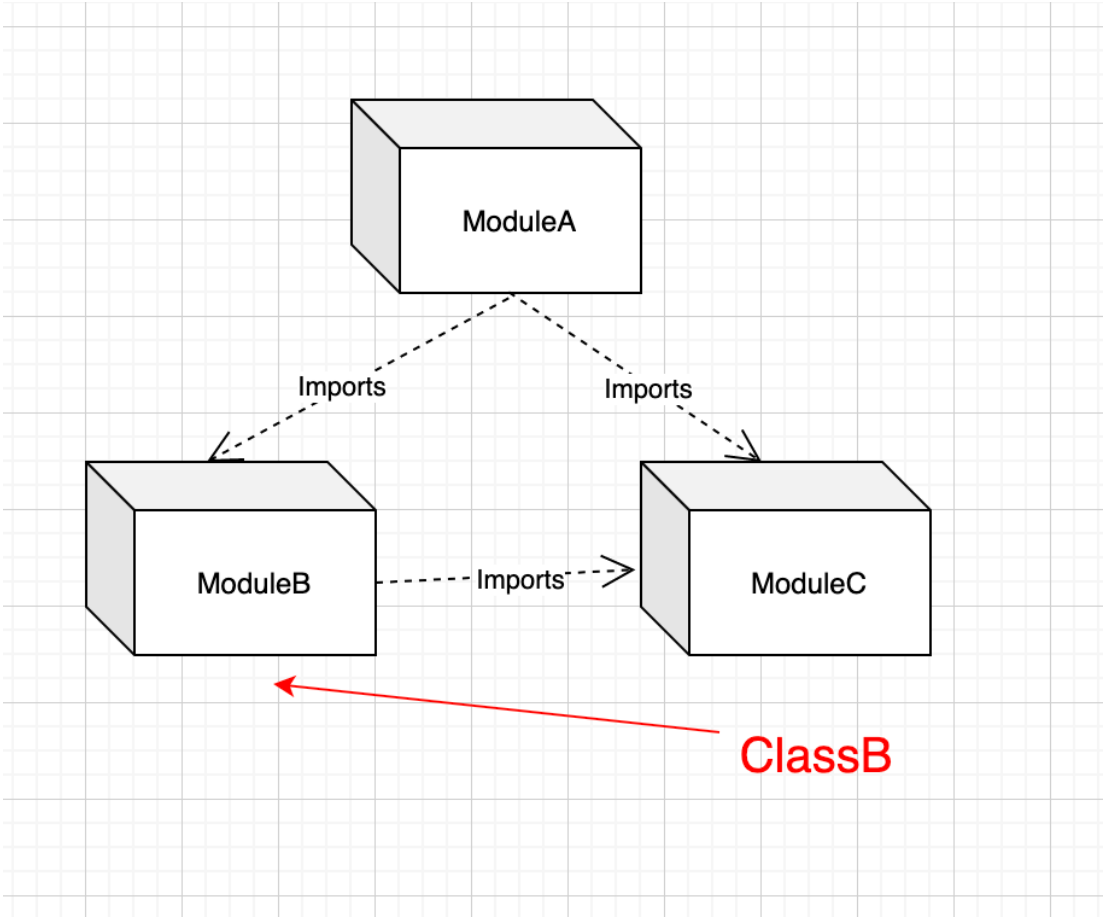

```swift
// ModuleB.swift
// Copyright © PJSC Bank Otkritie. All rights reserved.

import Foundation
import ModuleC

/// Class B
open class BClass {
    func doSomethingWith(classBobject: CClass) {}
}
```

```swift
// ModuleC.swift
// Copyright © PJSC Bank Otkritie. All rights reserved.

import Foundation
import ModuleB

/// Class C
open class CClass {
    func doSomethingWith(classBobject: BClass) {}
}
```

DI

- Одна из задач DI – разрулить циклические зависимости

Что имеем



ModuleA – добавим пустой компонент

```swift
import Foundation
import NeedleFoundation

/// Class A
open class AClass {
    public init(componentA: ComponentA) {
        self.componentA = componentA
    }

    var componentA: ComponentA
}

/// Зависимости компонента
public protocol DependencyA: Dependency {}

/// Компонент
public final class ComponentA: Component<DependencyA> {}
```

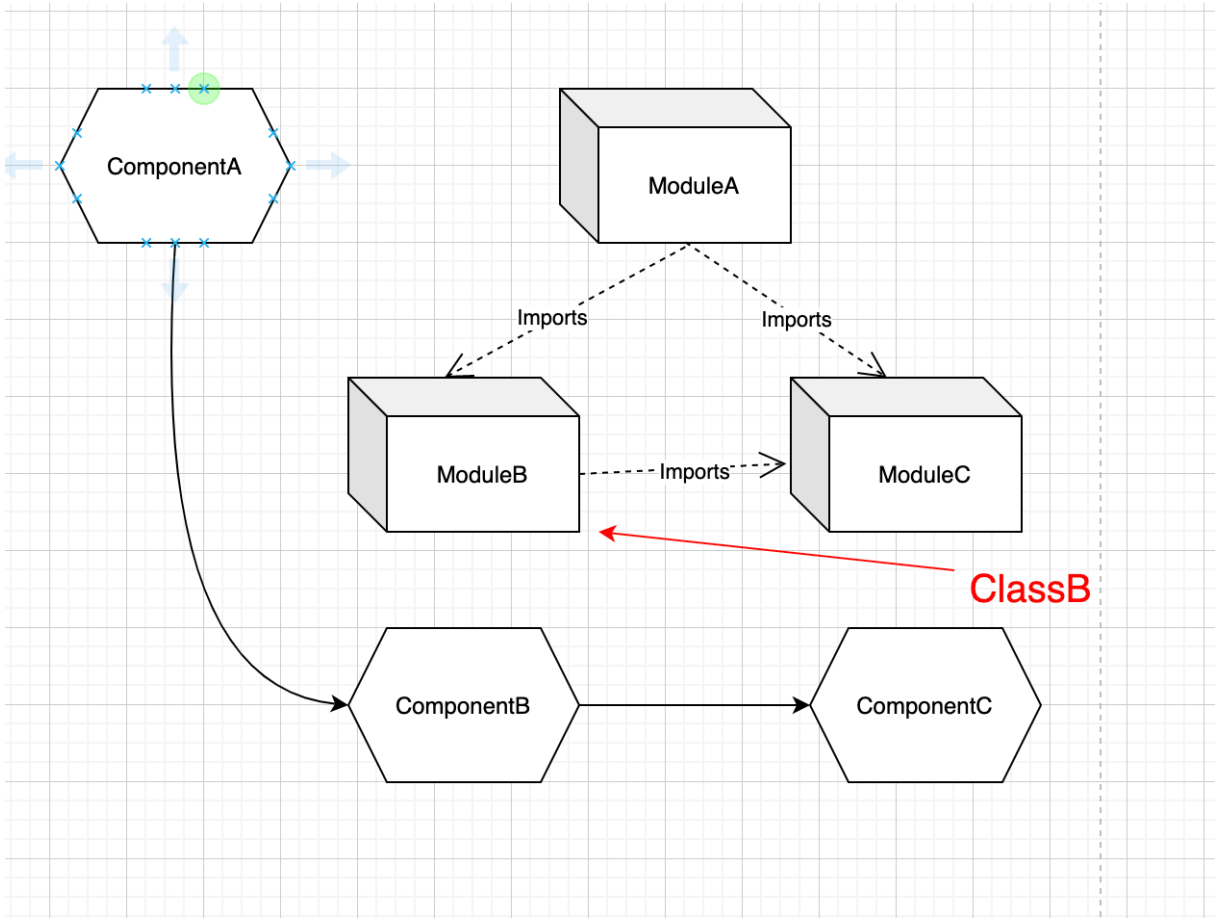
ModuleB – добавим пустой компонент

```swift
import Foundation
import ModuleC
import NeedleFoundation

/// Class B
open class BClass {
    public init(componentB: ComponentB) {
        self.componentB = componentB
    }

    var componentB: ComponentB

    func createCClass() -> CClass{
        return CClass(componentC: componentB.componentC)
    }
}

/// Зависимости компонента
public protocol DependencyB: Dependency {}

/// Компонент
public final class ComponentB: Component<DependencyB> {
    var componentC: ComponentC { return ComponentC(parent: self) }
}
```

ModuleC - добавим пустой компонент

```
/// Class C
open class CClass {
    public init(componentC: ComponentC) {
        self.componentC = componentC
    }

    var componentC: ComponentC

    func doSomethingWithBClass() {
    }
}

/// Зависимости компонента
public protocol DependencyC: Dependency {}

/// Компонент
public final class ComponentC: Component<DependencyC> {}
```

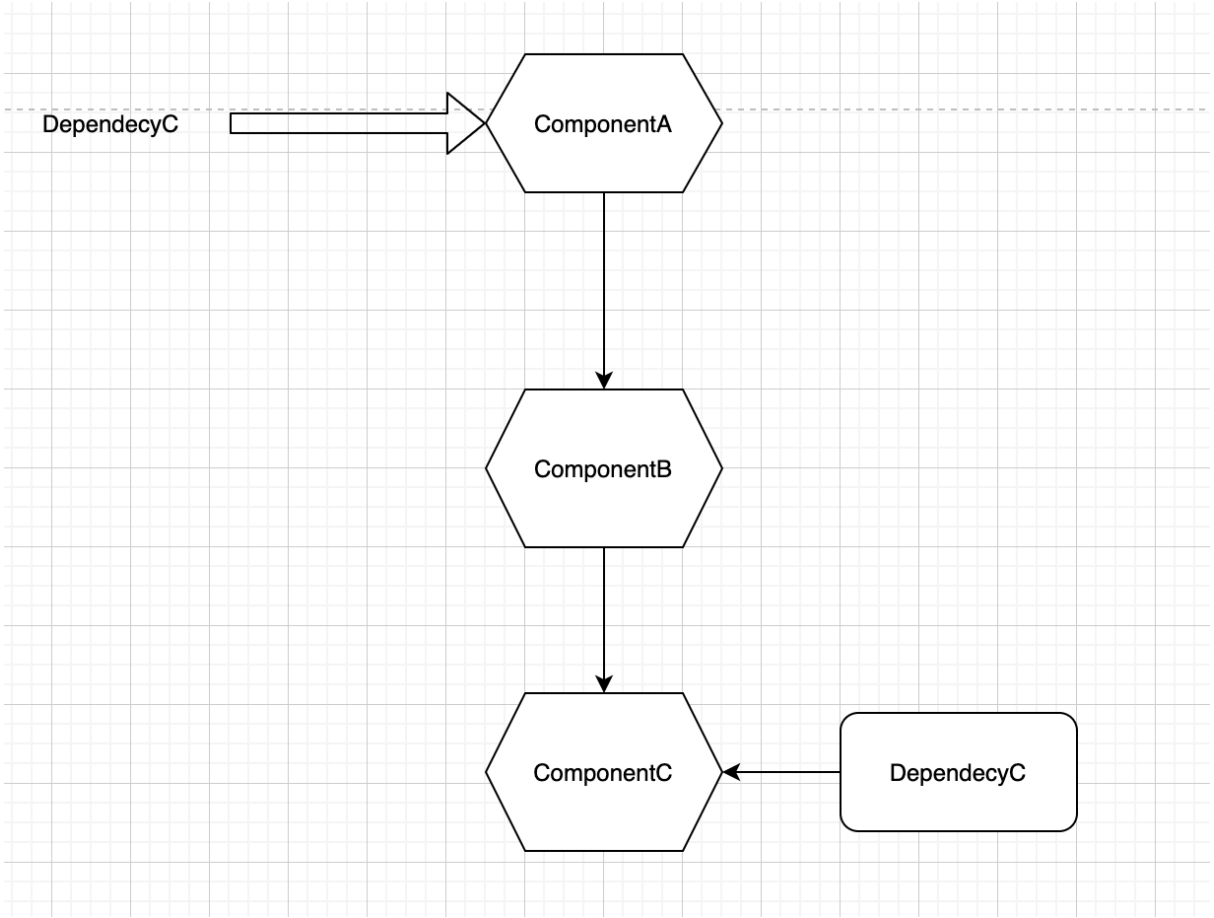
Пустые компоненты DI



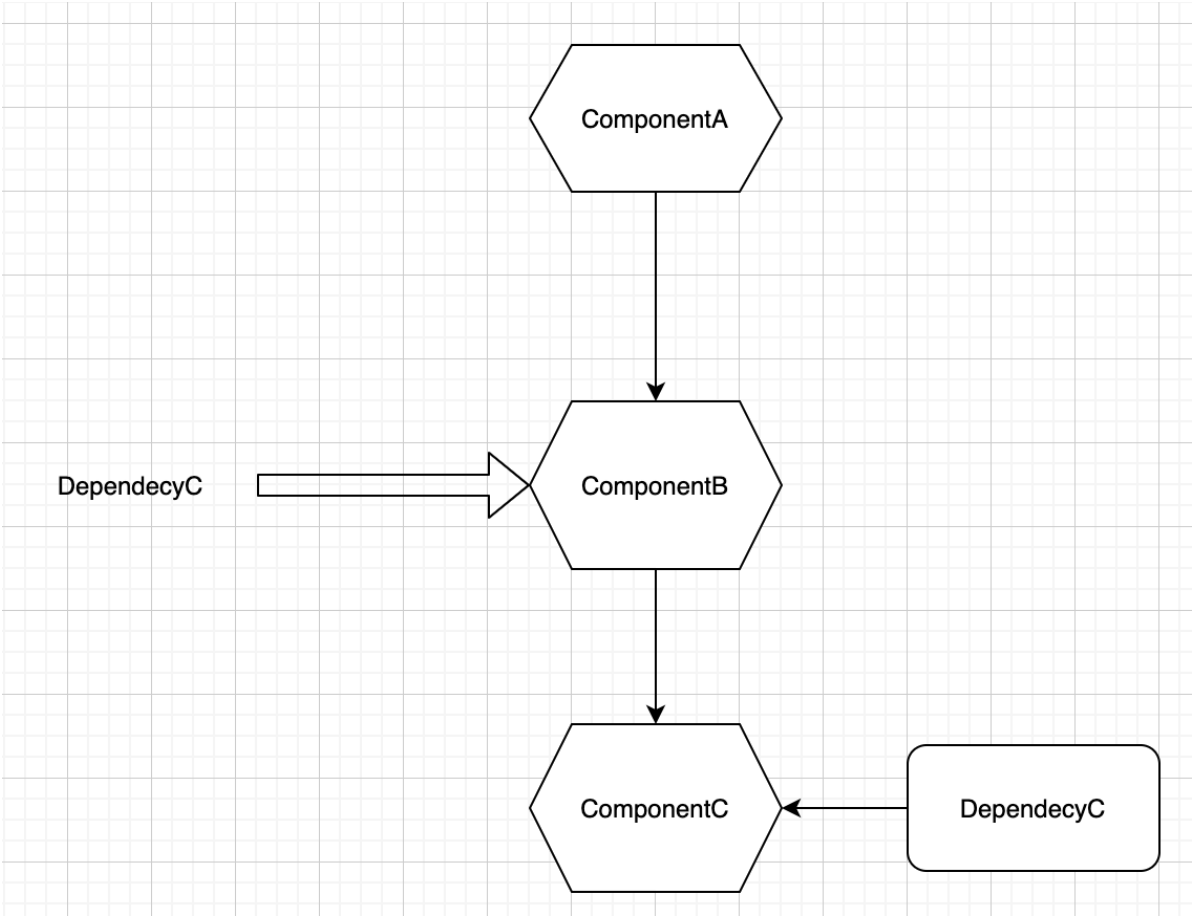
Задача

- Получить Класс из модуля B через DI

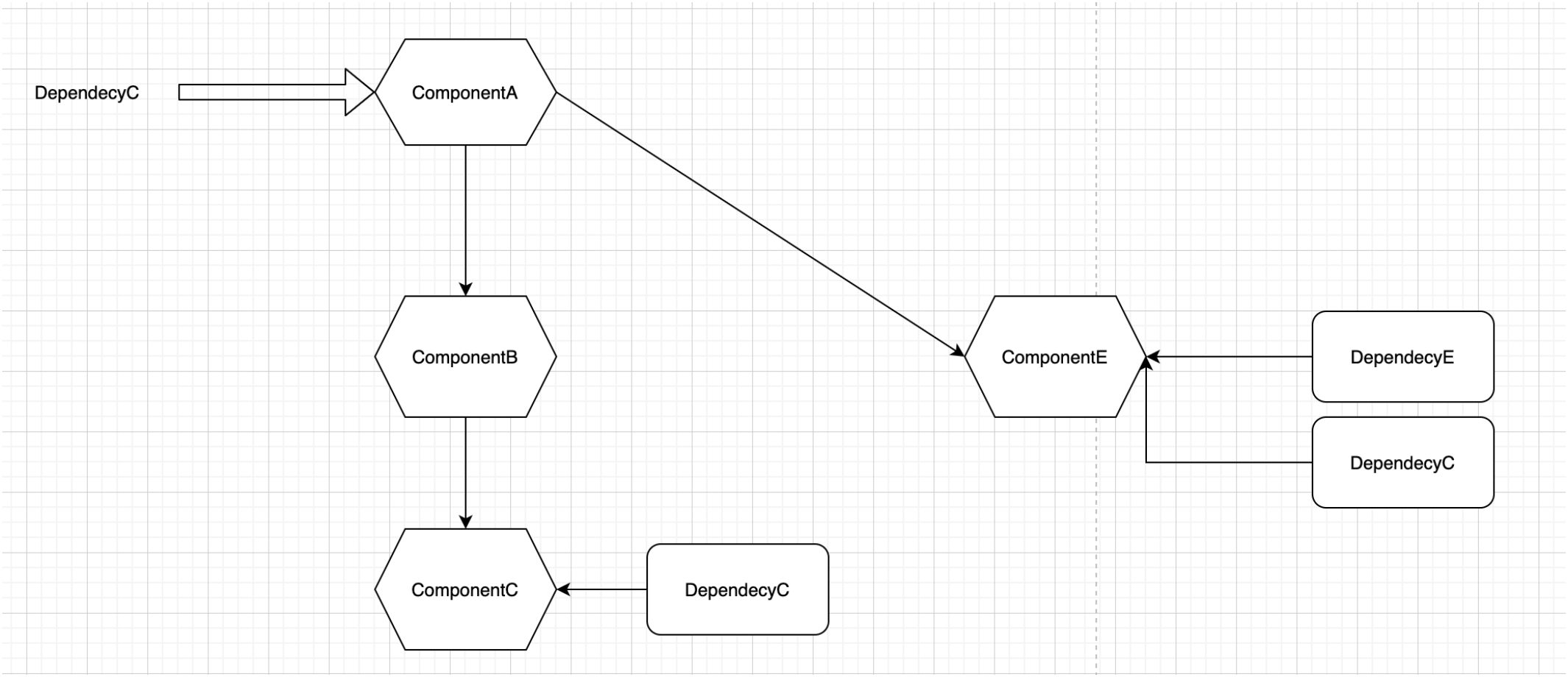
В каком месте инжектить зависимость в DI дерево



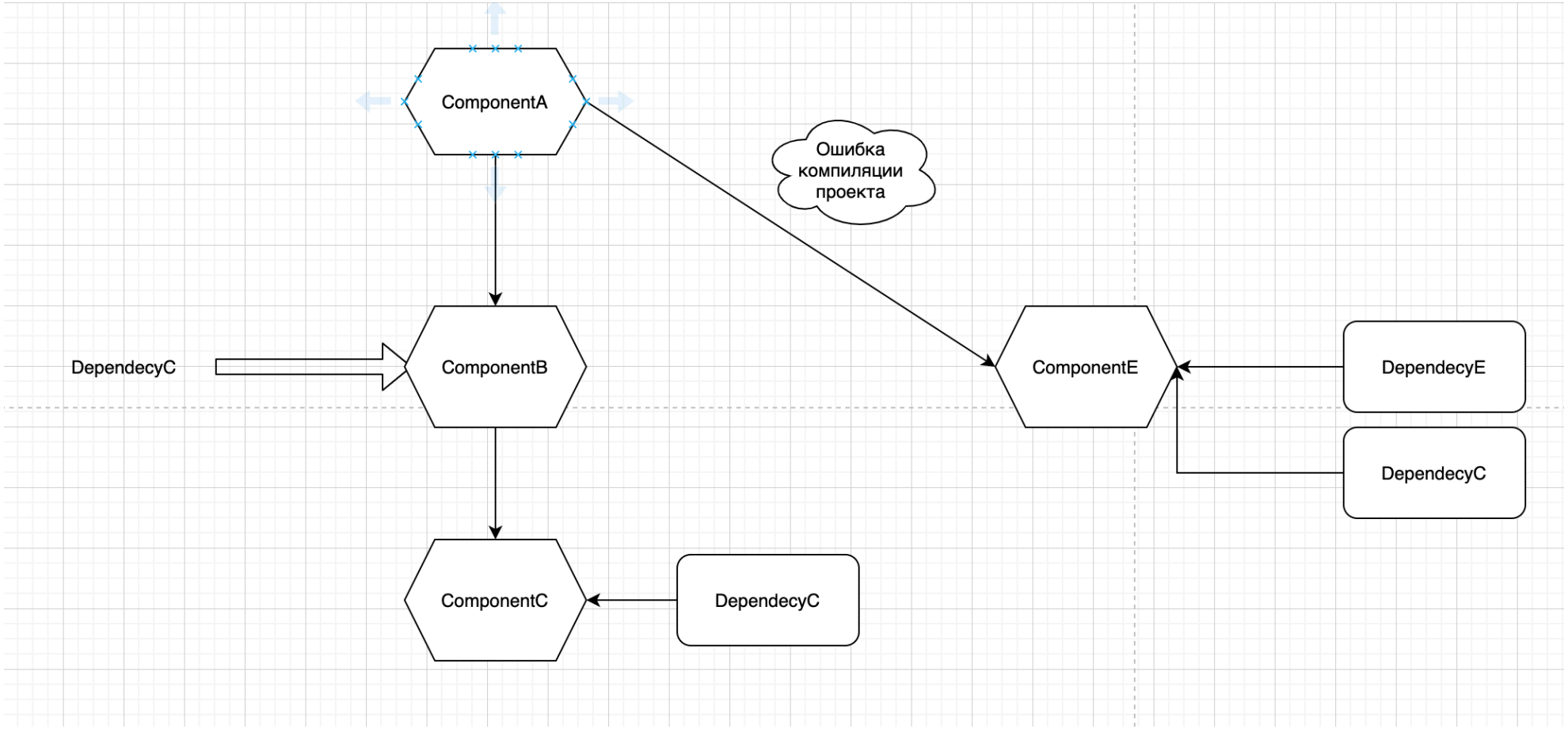
Чем ближе к родительский узел к исходному компоненту тем лучше, чтобы не захламлять компоненты верхнего уровня



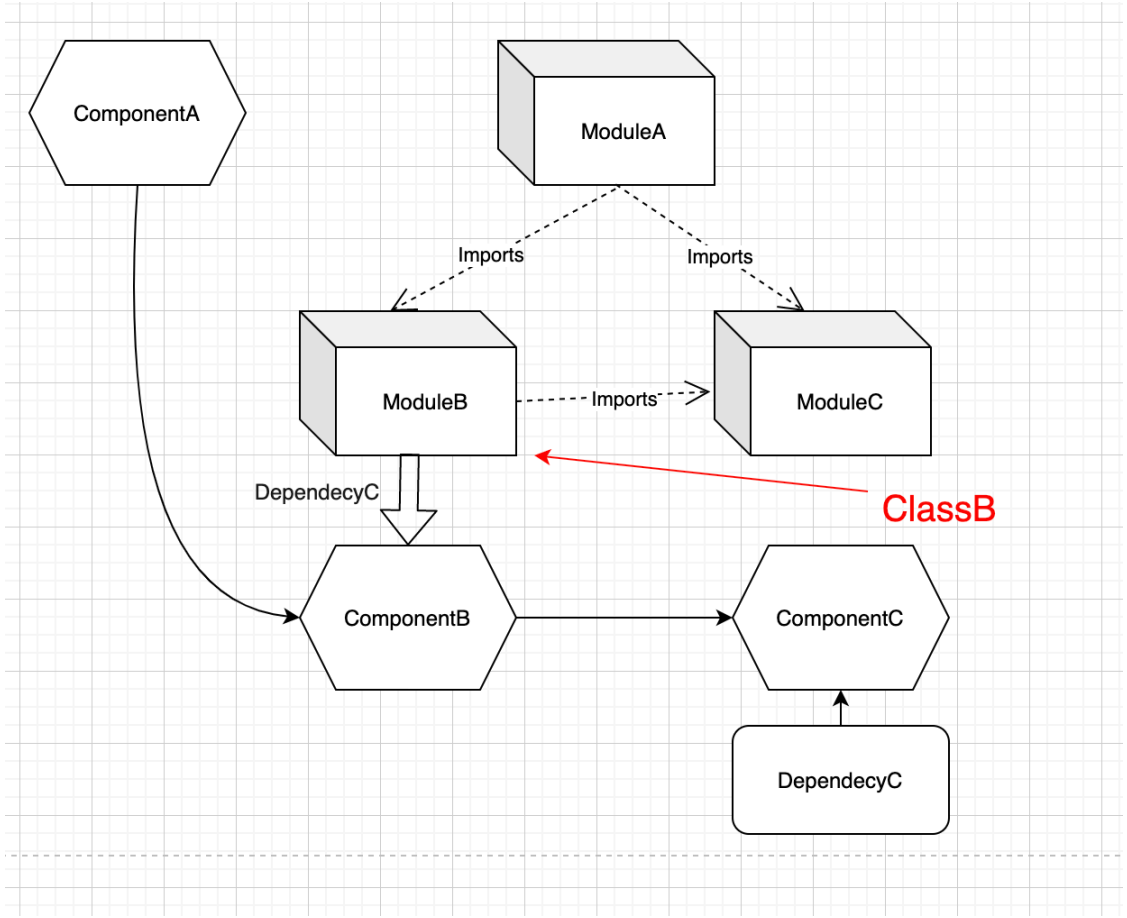
Те инжектим максимально низко, но учитываем граф



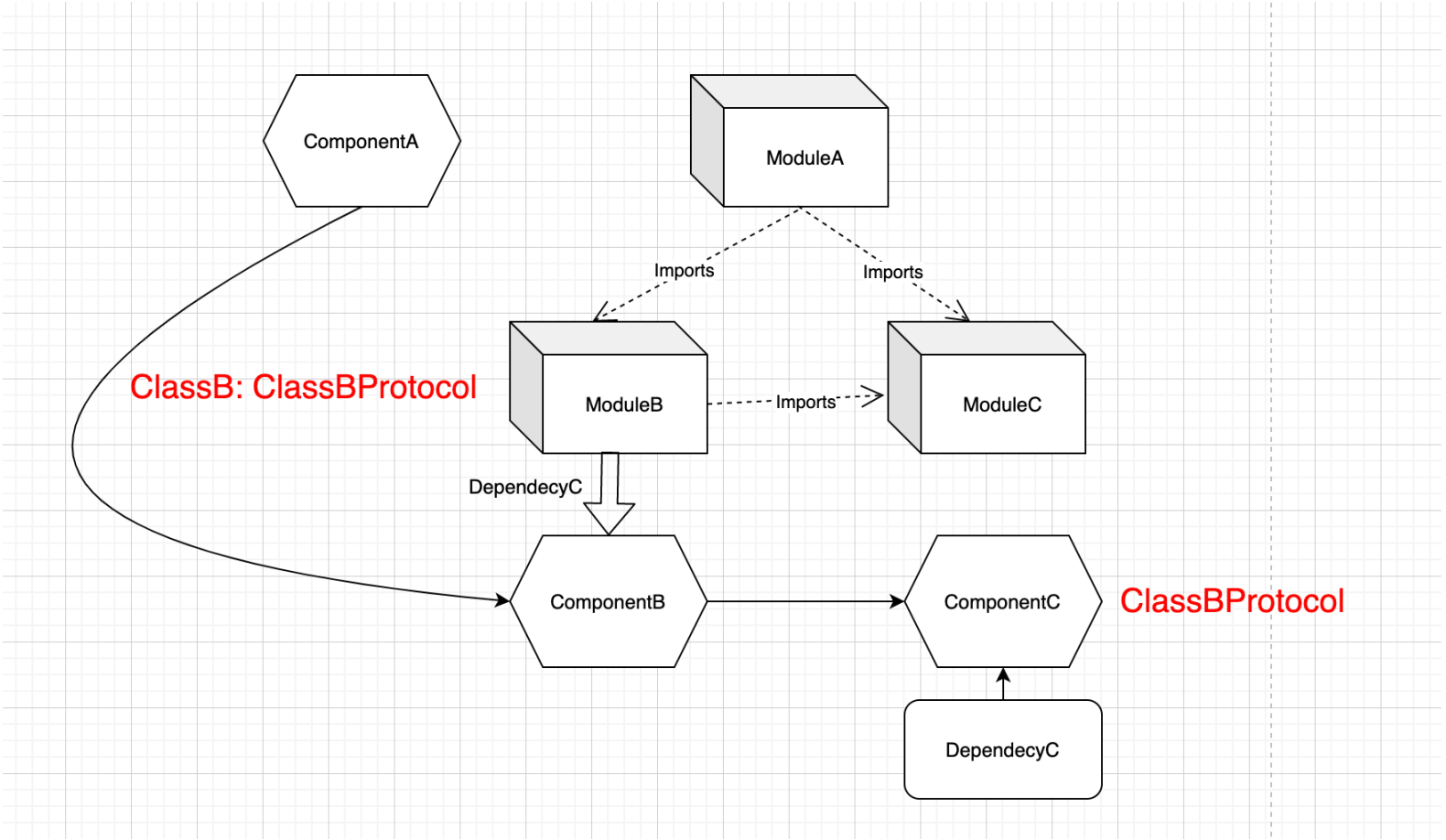
Если мы ошиблись, то ничего страшного. Есть компайл тайм проверки



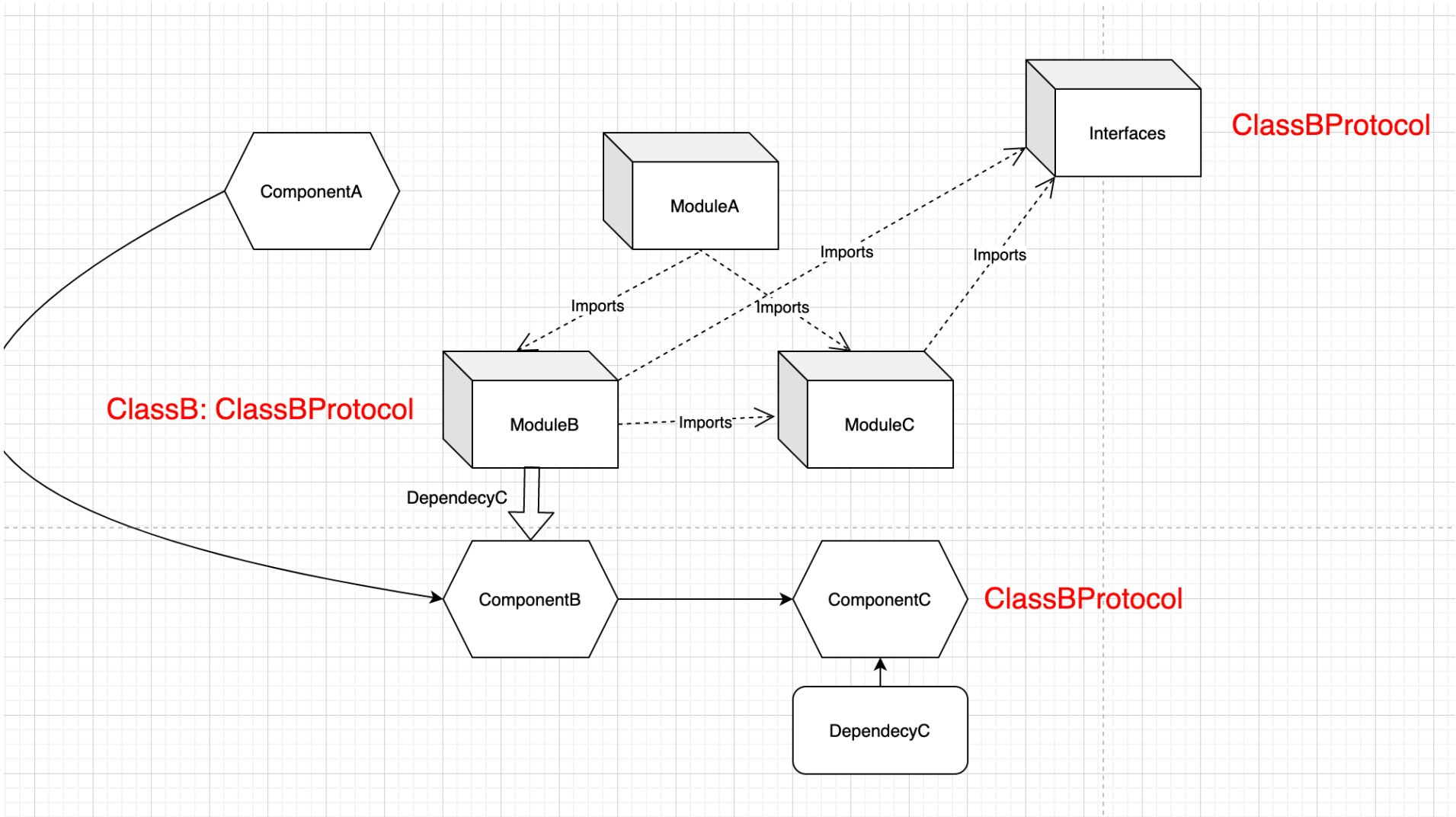
Добавляем зависимость



Добавляем протокол и уходим в абстракцию



Сохраняем протокол в отдельном модуле



Interfaces Module – покроем протоколом класс и сделаем его фабрику

```swift
/// bip
public protocol IndependentBClassProtocol {}

/// bip
public protocol IndependentBClassFactoryProtocol {
    func makeBClass(
        a: Int,
        b: Int
    ) -> IndependentBClassProtocol
}
```

Добавляем зависимость и используем ее

```
/// Class C
open class CClass {
    public init(componentC: ComponentC) {
        self.componentC = componentC
    }

    var componentC: ComponentC

    func doSomethingWithIndependentBClass() {
        let bClassFactory = componentC.dependency.getIndependentBClassFactory
        let classB: IndependentBClassProtocol = bClassFactory.makeBClass(a: 1, b: 2)
        print(classB)
    }
}

/// Зависимости компонента
public protocol DependencyC: Dependency {
    var getIndependentBClassFactory: IndependentBClassFactoryProtocol { get }
}

/// Компонент
public final class ComponentC: Component<DependencyC> {}
```

В компоненте модуля Б - резолвим зависисмости

```swift
//Case 1
/// Class B
open class BClass: BClassProtocol {
    public init(componentB: ComponentB) {
        self.componentB = componentB
    }

    var componentB: ComponentB

    func createCClass() -> CClass {
        CClass(componentC: componentB.componentC)
    }
}

/// IntependentBClass
open class IntependentBClass: IndependentBClassProtocol {
    func doSomething() {}
}

///
open class IntependentBClassFactory: IndependentBClassFactoryProtocol {
    public init() {}
    public func makeBClass(a: Int, b: Int) -> IndependentBClassProtocol { IntependentBClass() }
}

/// Зависимости компонента
public protocol DependencyB: Dependency {}

/// Компонент
public final class ComponentB: Component<DependencyB> {
    var componentC: ComponentC { ComponentC(parent: self) }

    var getIndependentBClassFactory: IndependentBClassFactoryProtocol { IntependentBClassFactory() }
}
```

Почему фактори – Если зависимость имеет сложный коструктор, то ее резолвинг будет сложным – в 2 строки

```swift
/// Хендлер для получения OfficeEnrollmentCoordinator
public typealias OfficeEnrollmentCoordinatorHandler = (MapObject, UINavigationController)
    -> OfficeEnrollmentCoordinatorProtocol

/// Зависимости компонента CardReplacementInfoViewController
public protocol CardReplacementInfoViewControllerDependency: Dependency {
    var mapObjectDetailCardViewController: MapObjectDetailCardViewControllerProtocol { get }
    var makeOfficeEnrollmentCoordinatorHandler: OfficeEnrollmentCoordinatorHandler { get }
}
```

```swift
var makeOfficeEnrollmentCoordinatorHandler: OfficeEnrollmentCoordinatorHandler { makeOfficeEnrollmentCoordinator }

func makeOfficeEnrollmentCoordinator(
    mapObject: MapObject,
    navigationController: UINavigationController
) -> OfficeEnrollmentCoordinatorProtocol {
    OfficeEnrollmentCoordinator(mapObject: mapObject, mainNavigationController: navigationController)
}
```

Почему фактори – наш пример

```swift
public protocol IndependentBClassFactoryProtocol {
    func makeBClass(
        a: Int,
        b: Int
    ) -> IndependentBClassProtocol
}
```

```swift
func doSomethingWithIndependentBClass() {
    let bClassFactory = componentC.dependency.getIndependentBClassFactory
    let classB: IndependentBClassProtocol = bClassFactory.makeBClass(a: 1, b: 2)
    print(classB)
}
```

```swift
/// Компонент
public final class ComponentB: Component<DependencyB> {
    var componentC: ComponentC { ComponentC(parent: self) }

    var getIndependentBClassFactory: IndependentBClassFactoryProtocol { IntependentBClassFactory() }
}
```

Помним

Переход к Factory нужен только тогда,
когда конструктор зависимости не пустой

Мы победили циклическую зависимость