

# DEPENDENCY *INJECTION*

<https://github.com/kontur-courses/di>



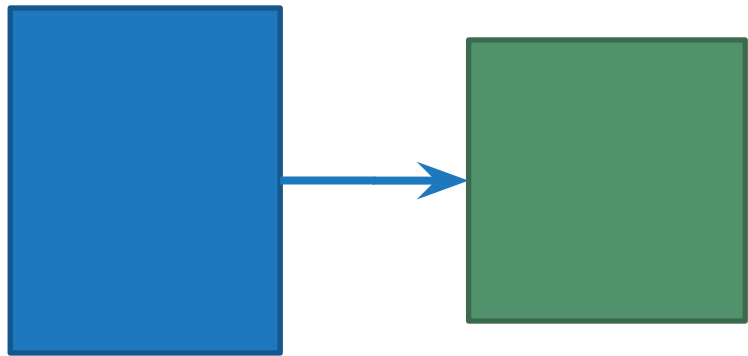
# НЕЯВНОЕ УПРАВЛЕНИЕ ЗАВИСИМОСТЯМИ

---

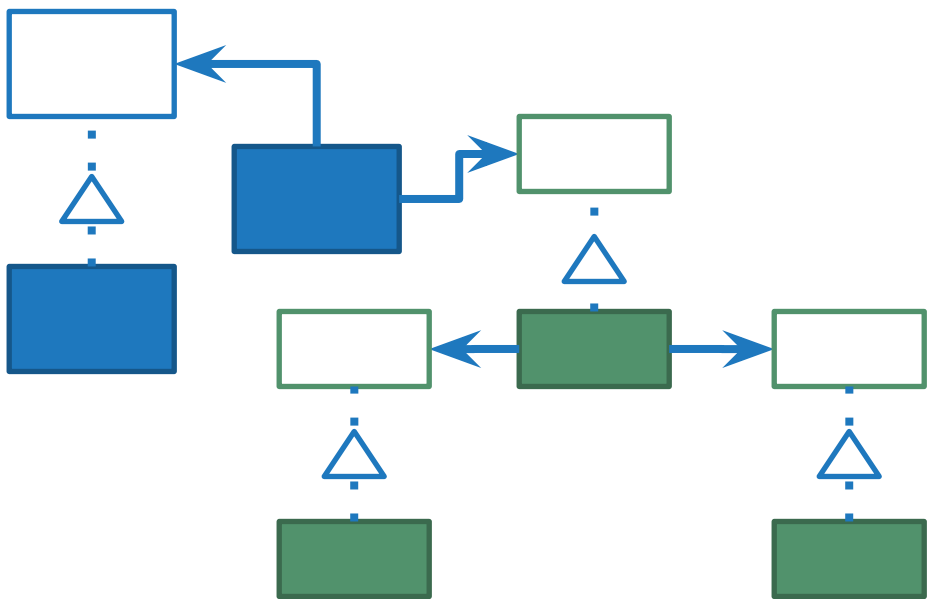
```
public class Chessboard
{
    public Chessboard()
    {
        this.cells = LoadBoard(
            new StreamReader("input.txt"),
            new BoardParser());
    }
}
```



Неявные  
связи




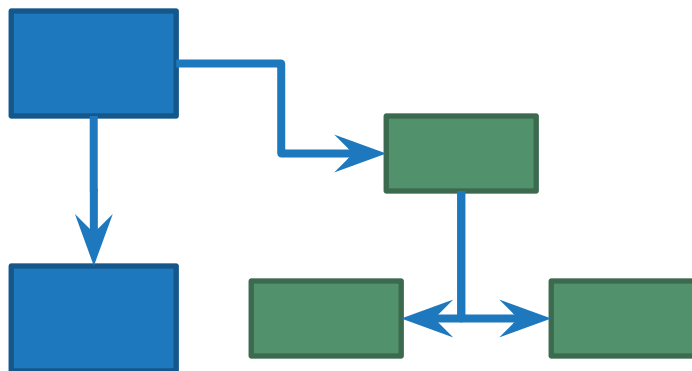
SRP



использование



реализация  
интерфейса  . . .



DIP

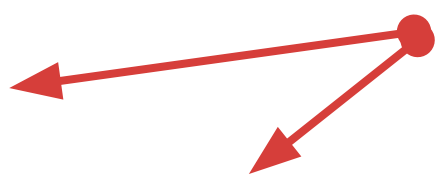


# ЯВНОЕ (ИНЪЕКЦИЯ ЗАВИСИМОСТЕЙ)

---

```
public class Chessboard
{
    public Chessboard(
        StreamReader input,
        IBoardParser boardParser)
    {
        this.cells = LoadBoard(input, boardParser);
    }
}
```

Нужные  
значения  
передадут извне



# ЯВНОЕ УПРАВЛЕНИЕ ЗАВИСИМОСТЯМИ

---

- ❑ Не вызывать статические методы
- ❑ Не вызывать конструкторы
- ✓ Ссылки на объекты передавать в конструктор
  - Кто их передает в конструктор?
  - Кто-то «свыше»!
    - А ему?
  - Тоже кто-то «свыше»!
    - А ему?



# ТОЧКА ВХОДА - МЕСТО СБОРА ЗАВИСИМОСТЕЙ

---

## Composition Root

- место, где модули соединяются вместе
- загончик для операторов `new`

## Чем ближе к точке входа – тем лучше

- `void Main()` для консольного приложения
- `HttpHandler` для веб-сервиса
- В библиотеках
  - либо не собирать – пусть собирает использующий
  - либо в классе-фасаде библиотеки

# КАКИЕ ЗАВИСИМОСТИ ДЕЛАТЬ ЯВНЫМИ?

---

Имена файлов, пути, порты, ...

Неудобные зависимости (файлы, консоль, ui, сеть, БД, ...)

Другие сервисы

Формат файла

*Алгоритм*

*если его может понадобиться менять*

*Реализация структуры данных*

*если её может понадобиться менять*

В ЧЕМ РАЗНИЦА МЕЖДУ  
DEPENDENCY *INJECTION*  
И  
DEPENDENCY *INVERSION* PRINCIPLE  
?



# DIP ЧЕРЕЗ SERVICE LOCATOR

---

```
public class Chessboard
{
    public Chessboard()
    {
        var reader = ServiceLocator.Instance.Get<StreamReader>();
        var parser = ServiceLocator.Instance.Get<IBoardParser>();
        this.cells = LoadBoard(reader, parser);
    }
}
```

# DIP ЧЕРЕЗ ИНЪЕКЦИЮ ЗАВИСИМОСТЕЙ

---

```
public class Chessboard
{
    public Chessboard(
        StreamReader input,
        IBoardParser boardParser)
    {
        this.cells = LoadBoard(input, boardParser);
    }
}
```

# DEPENDENCY INVERSION...

---

Обеспечивается разными способами

- Dependency Injection
- Service Locator

# ПОЧЕМУ SERVICE LOCATOR – АНТИПАТТЕРН?

---

- Скрывает реальные зависимости класса
  - Ухудшается читабельность
  - Увеличивается хрупкость
- Заражает весь код, в котором используется
  - Наркотик, с которого трудно слезть

---

DI CONTAINER

```
public class Robot : IRobot {  
    public Robot(IDistanceSensor distanceSensor) {  
        this.distanceSensor = distanceSensor;  
    }  
    ...  
}
```

```
public class OpticalSensor : IDistanceSensor {  
    ...  
}
```

*Как это работаем?*

```
IRobot robot = container.Get<IRobot>();
```

# А ЕСЛИ ЗАВИСИМОСТЕЙ МНОГО?

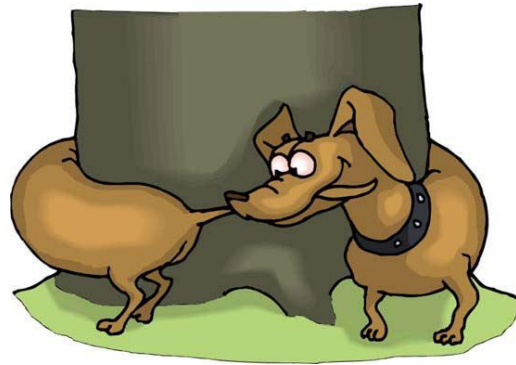
---

```
public HttpServer(IEnumerable<IHandler> handlers) { ... }  
public FileLoader(IFileFormat[] formats) { ... }  
public DocumentPage(IAction[] actions) { ... }
```

# А ЕСЛИ ЗАВИСИМОСТИ ЦИКЛИЧЕСКИЕ?

---

```
Controller(IView view)  
BasicView(HomeController controller): IView
```



```
Controller(Lazy<IView> lazyView)  
BasicView(HomeController controller) : IView  
var c = container.Get<Controller>();
```



# ФАБРИКА ВМЕСТО NEW

---

```
var createMyClass =  
    container.Get<Func<int, string, MyClass>>();
```

```
MyClass c = createMyClass(42, "!");
```

<https://github.com/ninject/Ninject.Extensions.Factory/wiki>

# ПРОЧИЕ ОСОБЕННОСТИ

---

- Время жизни (InSingletonScope, InThreadScope)
- Именованные зависимости (Named)
- Контекстно-зависимое инжектирование (WhenInjectedInto)
- Конвенции и авторегистрация (FromThisAssembly ...)
- Модульность конфигурации
- Одна реализация для нескольких интерфейсов (Bind<T1, T2>)
- Generics
- ...

# ЗАДАЧА **FRACTALPAINTER**

---

В программе **FractalPainter** странно реализован Dependency Inversion...

Необходимо произвести рефакторинг по списку в **README.md**

# РАЗБОР ЗАДАЧИ FRACTALPAINTER

---

- Можно внедрять контейнер постепенно
- Контейнер должен использоваться в одном месте, а не заражать код
- Контейнер подходит для разных ситуаций
  - Bind to Class, Bind to Constant, Bind to Method
  - Singleton Scope, Factory, Lazy
  - When
  - Conventions

# ОБРАТНАЯ СВЯЗЬ

---



Заполни форму обратной связи по ссылке

<http://bit.ly/kontur-courses-feedback>

ИЛИ

по ярлыку *feedback* в корне репозитория