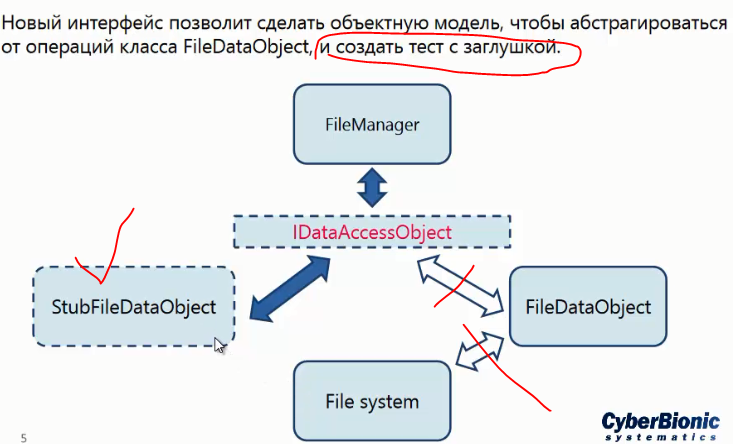
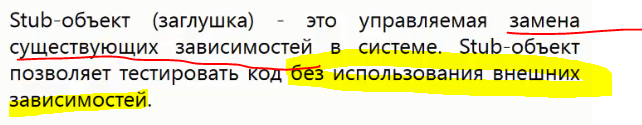


Вместо реальной файловой системы будем использовать класс заглушку

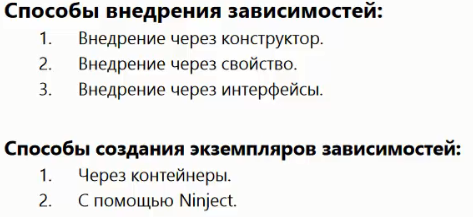






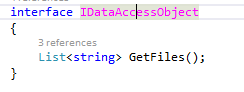


Внедрение зависимости



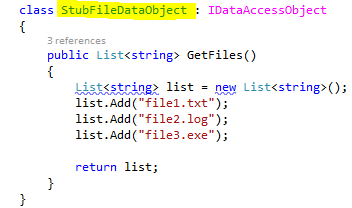
Внедрение зависимости через конструктор



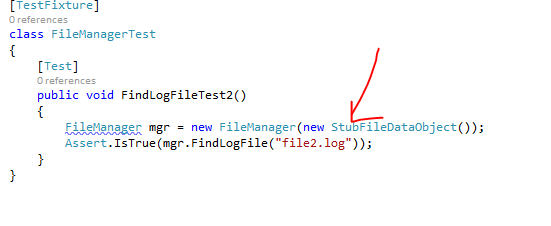


Оба класса настоящий и подставной реализуют интерфейс и в зависимости тест это либо нет мы работаем с классом FileManager

Класс для замены (подставной)

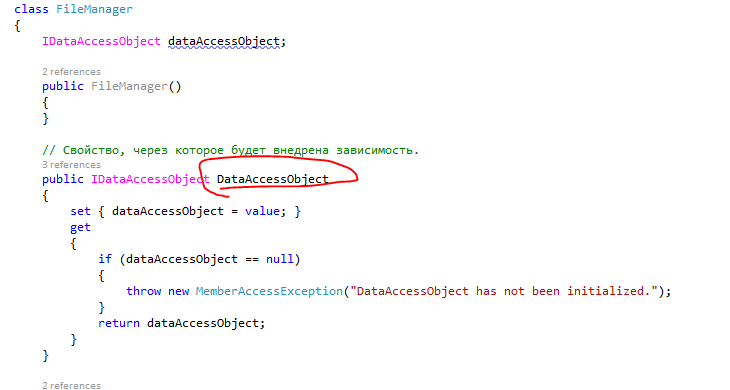


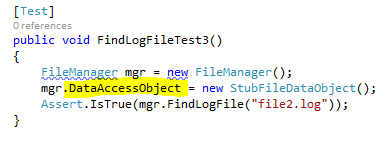
Вставляем для теста подставной класс



Внедрение зависимости через свойство

Аналогично как с конструктором только экземпляр класса присваивается свойству





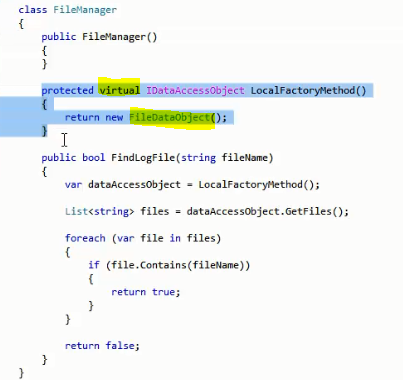
Внедрение зависимости через интерфейс

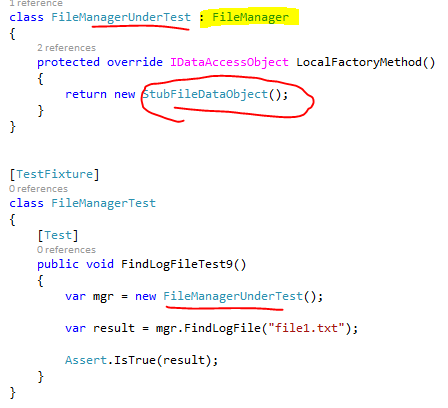
Все как и в остальных примерах только передается ПОДСТАВНОЙ класс в метод параметром



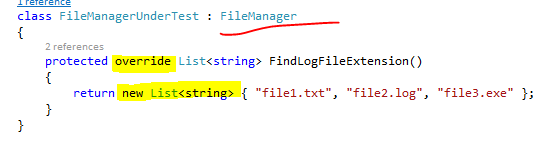


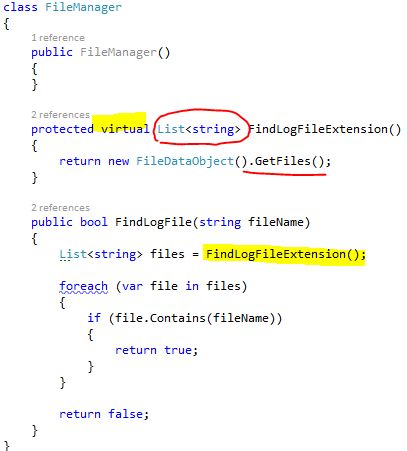
Локальный фабричный метод, виртуальный и если мы захотим создать подставной класс то унаследовавшись переопределим этот метод





Вызывая на экземпляре метод, возвращающий коллекцию



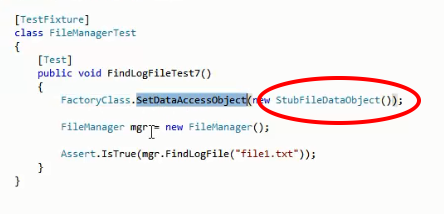


Еще один способ

Инициализируем подставной класс статической фабрикой, если объект пуст, то возвращаем настоящий экземплер, а если для теста то в методе SetDataAcessObject



Подставляем подставной класс в тесте

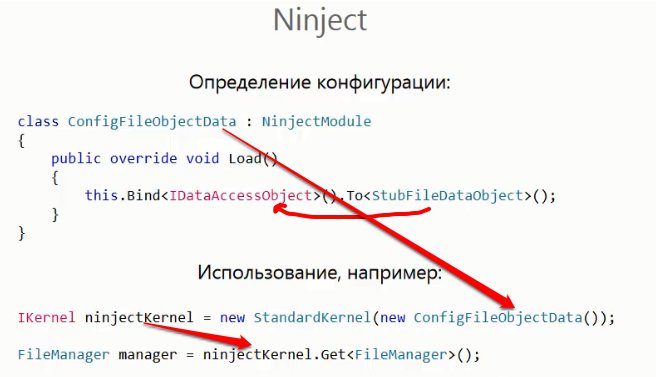




Везде где будет требоваться объект IDataAcessObject будет передаваться StubFileDataObject



Замена данных классов



Пример через DI Контейнер

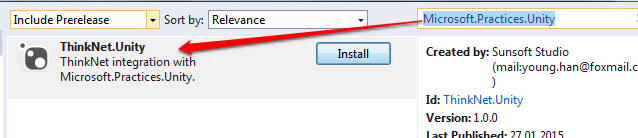
Свойство через которое будет внедрена зависимость помечается атрибутом [Dependency],

Только свойство

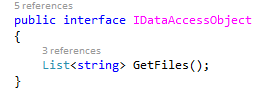








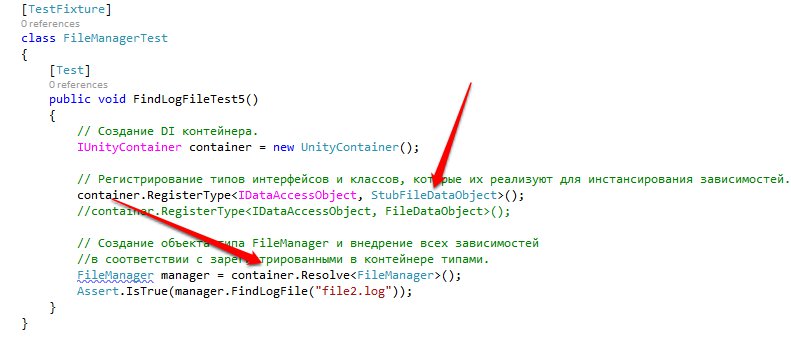
Реализация интерфейса



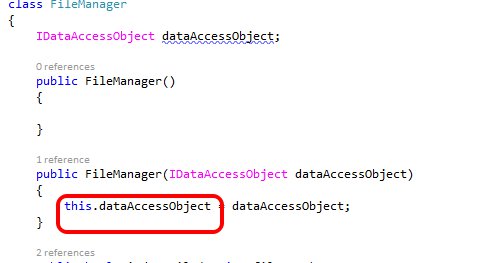
Подставной класс (контейнер)



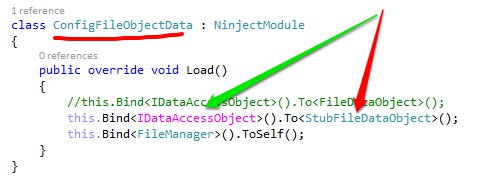
Тест



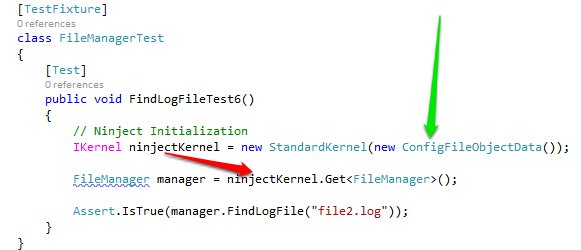
Пример тестирования через Ninject



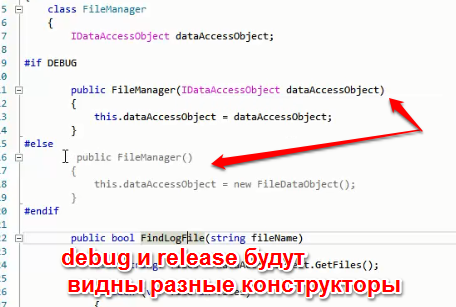
Создаем класс где указываем (красным) какой поставной объект использовать, вместо какого (зеленым)



Созданный класс подставной (зеленным)



Видимость закрытых членов класса



В тестах это тоже нужно указывать (но не обязательно)

