Анализ структуры кода и разработка методов рефакторинга архитектуры приложения в ООП

Афонина Алиса
Научный руководитель: Шкредов Сергей

Санкт-Петербург, 2012

Рефакторинг – это процесс изменения программной системы, при котором не меняется внешнее поведение кода, но при этом улучшается его внутренняя структура

ReSharper

- Дополнение для IDE Microsoft Visual Studio
- Повышает продуктивность работы за счет проведения статического анализа кода, предоставления дополнительных возможностей как-то:
  - автодополнения
  - рефакторинги
  - подсветка синтаксиса
  - навигация
  - ...

Цели

- Реализовать несколько методов автоматического рефакторинга архитектуры приложения
- Научиться анализировать возможность применения некоторых рефакторингов архитектуры

Задача

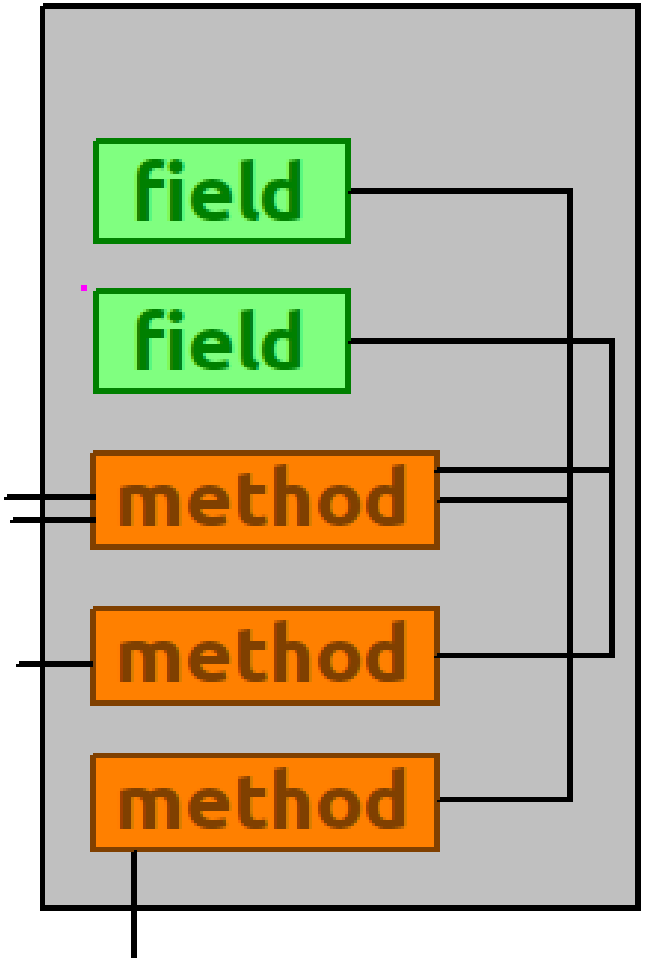
Реализовать рефакторинги

- “Выделение класса” (Extract Class)
- “Включение класса” (Inline Class)

Язык: C#
Интеграция с проектом ReSharper

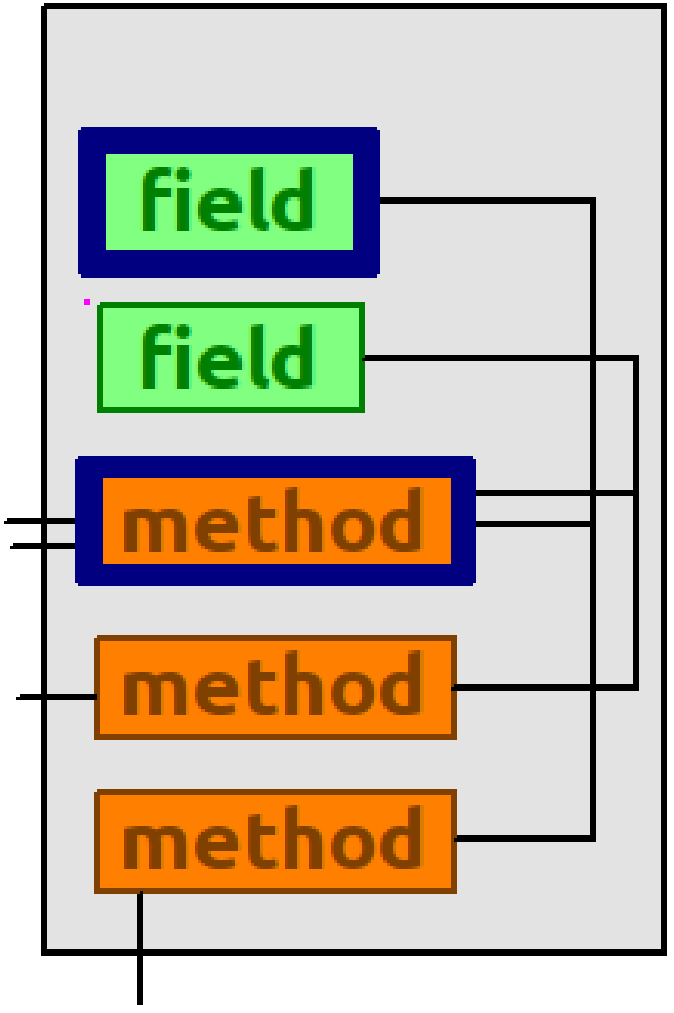
Extract Class Refactoring

Разбиение одного класса на два или более



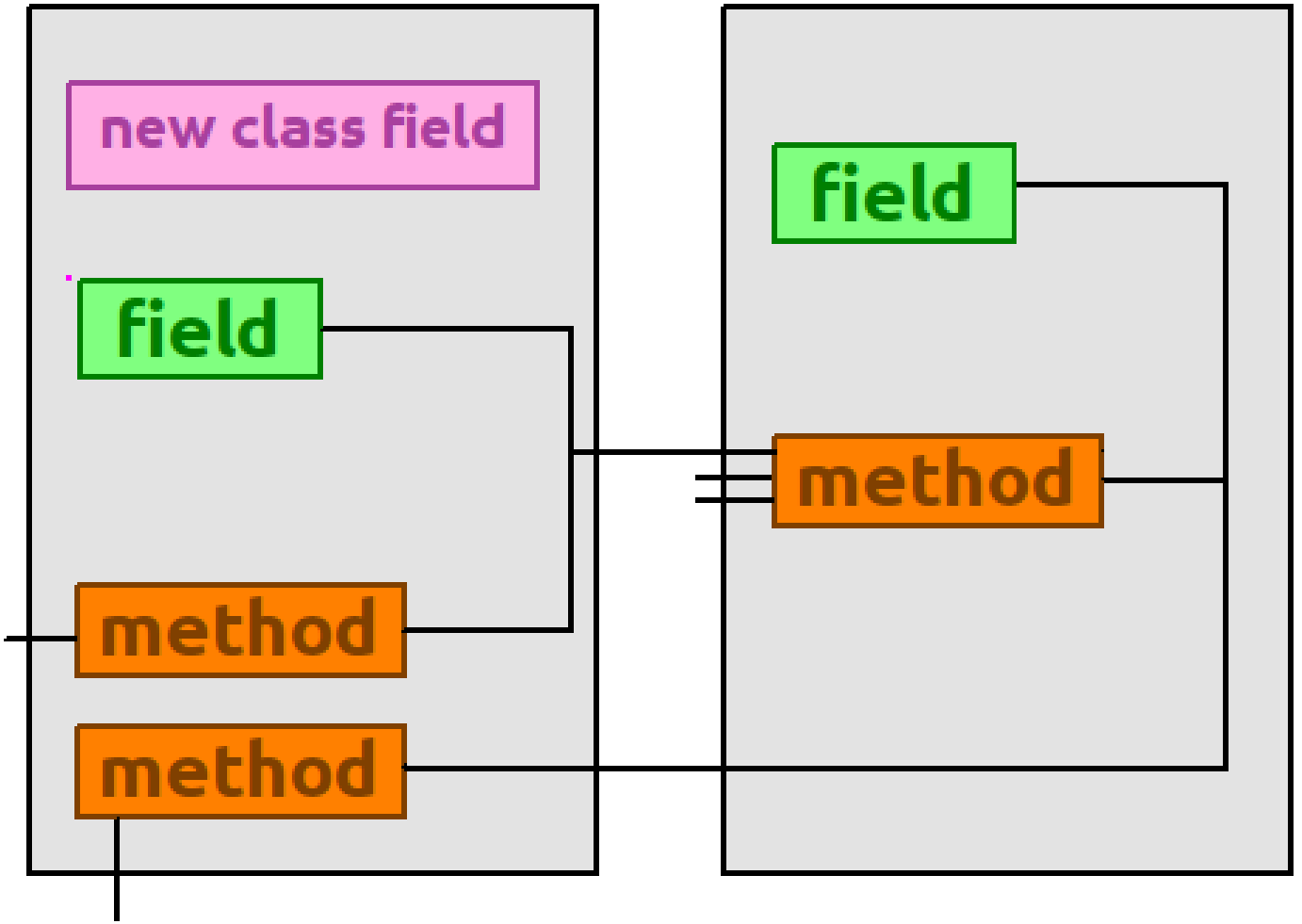
Extract Class Refactoring

Разбиение одного класса на два или более



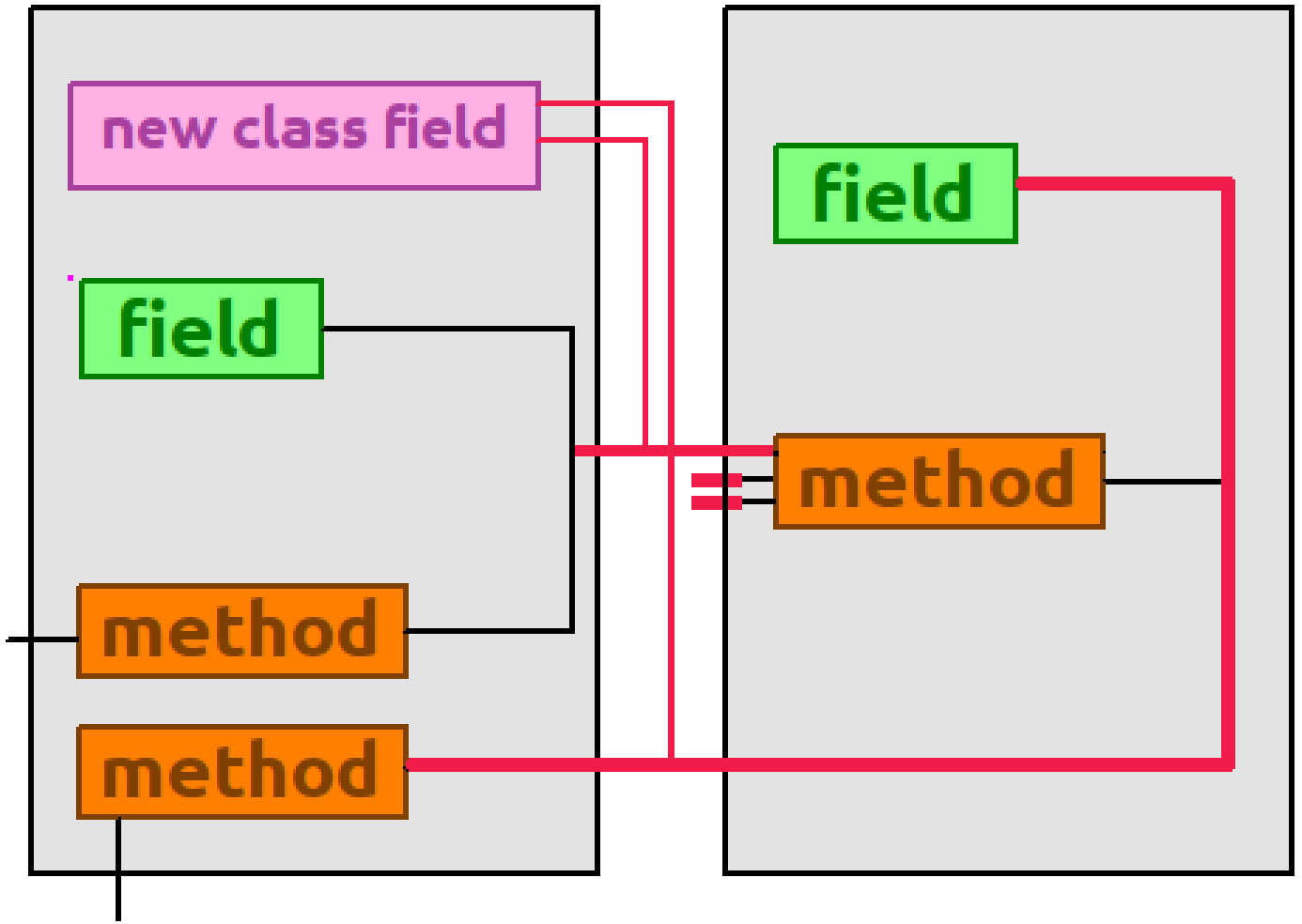
Extract Class Refactoring

Разбиение одного класса на два или более



Extract Class Refactoring

Разбиение одного класса на два или более



Предпосылки к применению

- Сложный класс
- Один класс – несколько функций
- Наличие циклических зависимостей

Аналоги

- C#

  Для ReSharper существует несколько аналогов (CodeRush (Refactor!Pro), Visual Assist X, JustCode)

  Ни в одном из них данного рефакторинга не реализовано

- Java
  - Eclipse
    - Вынесение только данных, нет вынесения зависимых функций
  - Intellij IDEA
    - Наибольшая функциональность
    - Некорректная обработка некоторых случаев
    - Не изменяет числа методов

Требования к реализации

- Возможность выбора данных
- Определение членов класса, зависящих от выбранных данных
- Возможность делегирования методов вместо их прямого использования
- Разрешение конфликтов

Конфликты

- После переноса элемент становится недоступен
  - Создание элементов доступа
  - Изменение доступа
- Новому классу необходим экземпляр старого
  - Добавление связи с исходным классом

Иногда разрешить невозможно

Реализация

Source class

```
class A {
  private int a,b,c,d,e,f,g;
  A () {}
  public void foo() {}
. . .
  private void bar() {}
}
```

Реализация

Source class

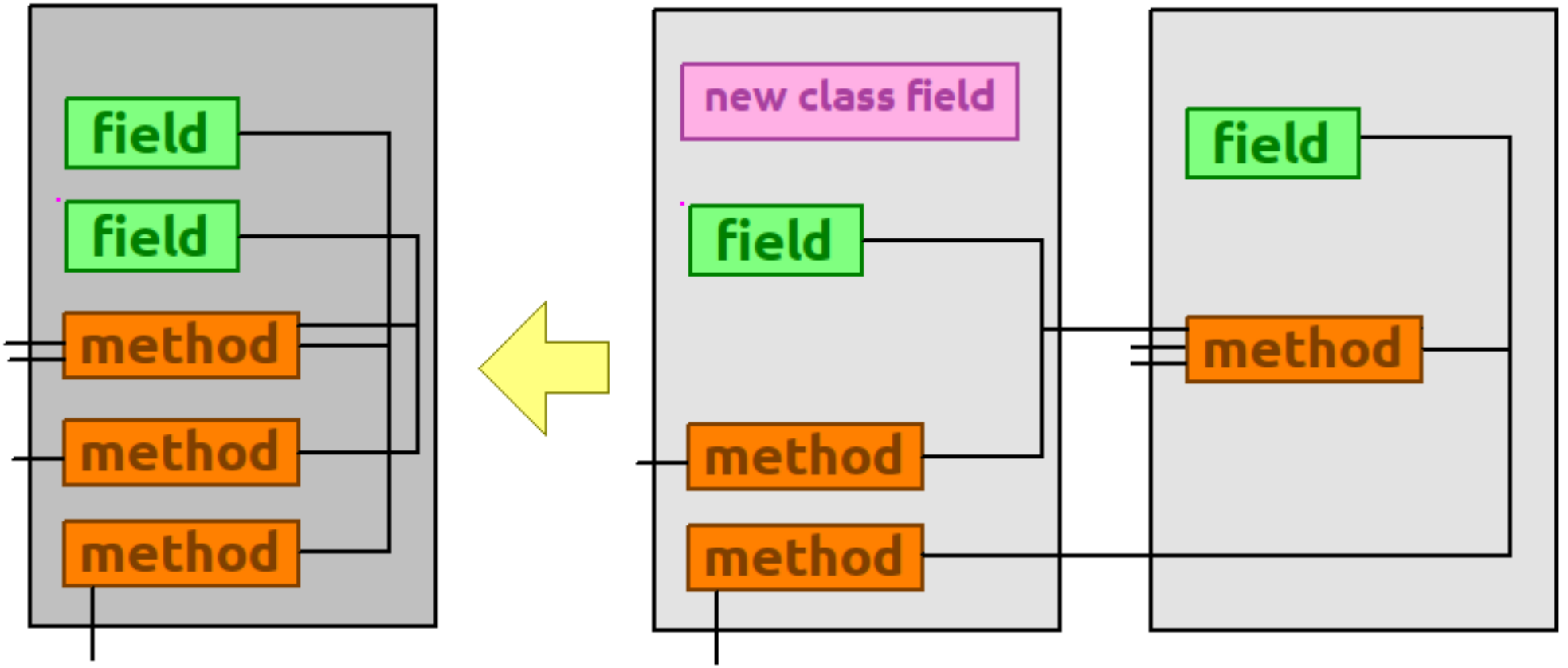
```
class A {
  private int a,c,e;
  private B myB;
  A () {myB = new B();...}
  public void foo() {}
...
}
```

Target Class

```
class B {
  private int b,d,f,g;
  B () {...}
...
  private void bar() {}
}
```

Inline Class Refactoring

- Объединение сущностей



Inline Class Refactoring

- Обратный к Extract Class Refactoring
- Предпосылки к использованию
  - Удаление маленького класса
  - Удаление класса-посредника
- Аналогов не найдено

Требования к реализации

- Выбор поля в классе
- Внесение в класс всех членов типа выбранного поля
- Разрешение конфликтов

Реализация

- Возможность запуска на поле класса
- Невозможность применения рефакторинга
  - класс включен в некоторую иерархию
  - класс создается где-либо, кроме конструкторов того класса, в который предполагается его внести
- Обработка делегирования, конструкторов и других особых ситуаций

Результаты

- Разработаны, протестированы и внедрены в ReSharper компоненты для выполнения рефакторингов Extract Class и Inline Class
  - Учитывают особенности языка
  - недостатки конкурентов