ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

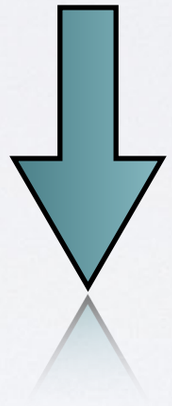
ПРИЕМЫ РЕФАКТОРИНГА

- Составление методов.
- Перемещение функций между объектами.
- Организация данных.
- Упрощение условных выражений.
- Упрощение вызовов методов.
- Решение задач обобщения.

РЕШЕНИЕ ЗАДАЧ ОБОБЩЕНИЯ

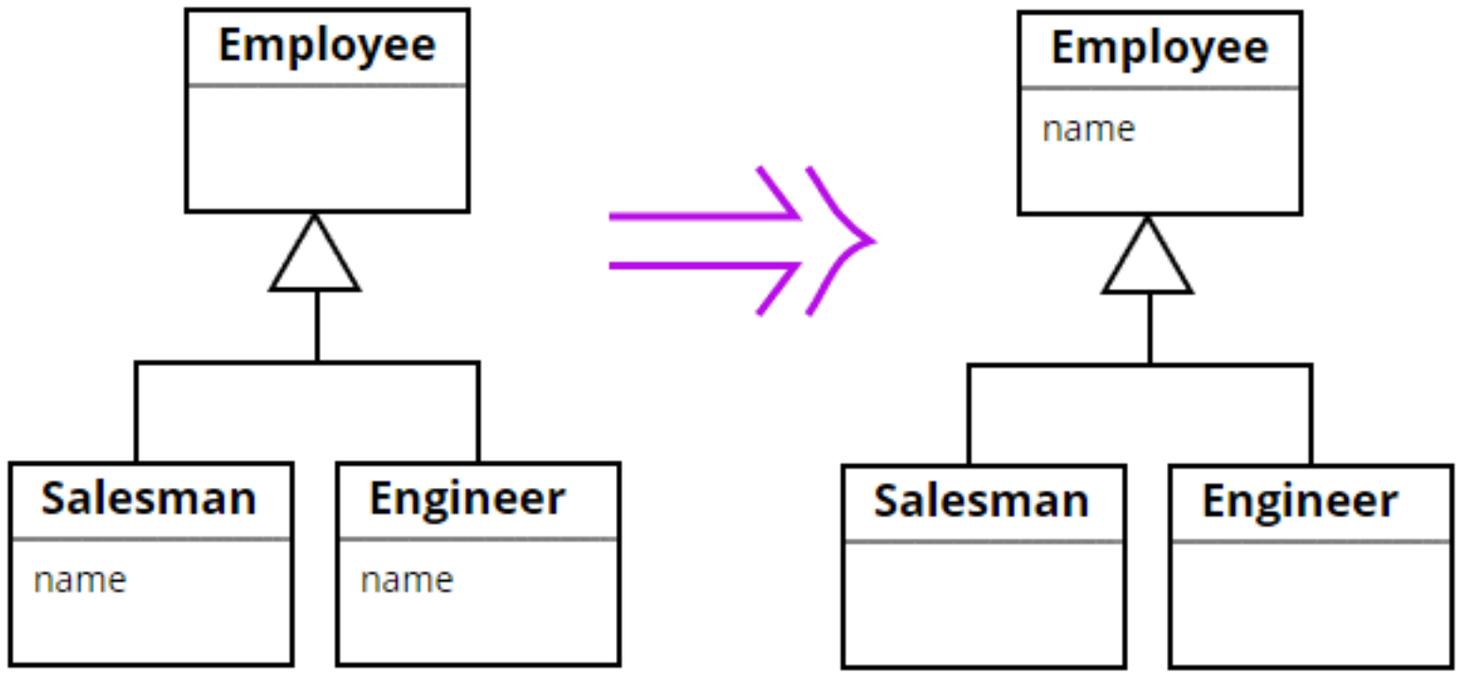
ПОДЪЕМ ПОЛЯ (PULL UP FIELD)

Два класса имеют одно и тоже поле.



Переместите поле в базовый класс, убрав его из подклассов.

ПОДЪЕМ ПОЛЯ (PULL UP FIELD)



ПРИЧИНЫ РЕФАКТОРИНГА

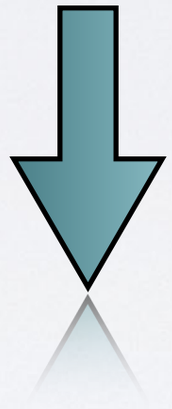
Подклассы развивались независимо друг от друга. Это привело к созданию одинаковых (или очень похожих) полей и методов.

ДОСТОИНСТВА

- **Убирает дублирование полей в подклассах.**
- **Облегчает дальнейший перенос дублирующих методов из подклассов в базовый класс, если они есть.**

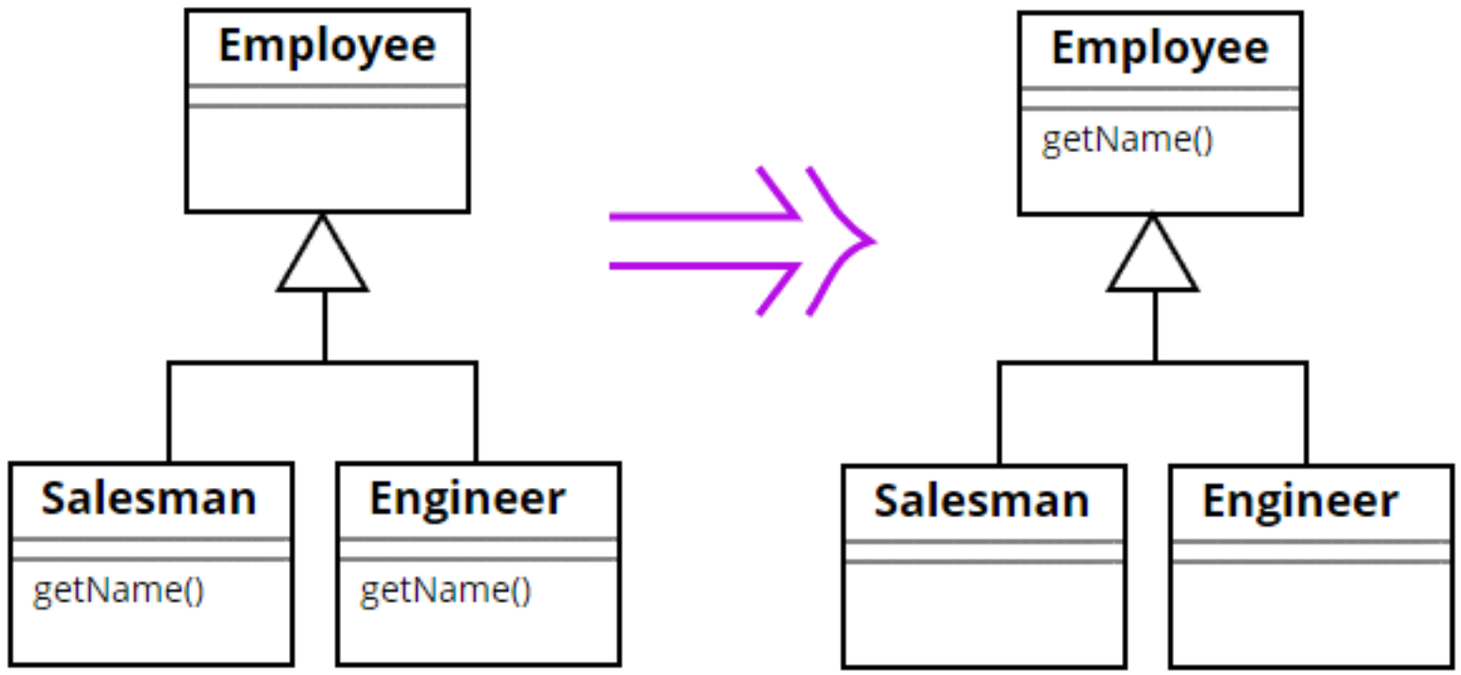
ПОДЪЕМ МЕТОДА (PULL UP METHOD)

Подклассы имеют методы, которые делают схожую работу.



Сделайте методы идентичными, а затем переместите их в базовый класс.

ПОДЪЕМ МЕТОДА (PULL UP METHOD)



ПРИЧИНЫ РЕФАКТОРИНГА

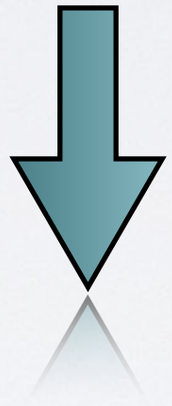
Подклассы развивались независимо друг от друга. Это привело к созданию одинаковых (или очень похожих) полей и методов.

ДОСТОИНСТВА

Убирает дублирование полей в подклассах. Проще модифицировать метод.

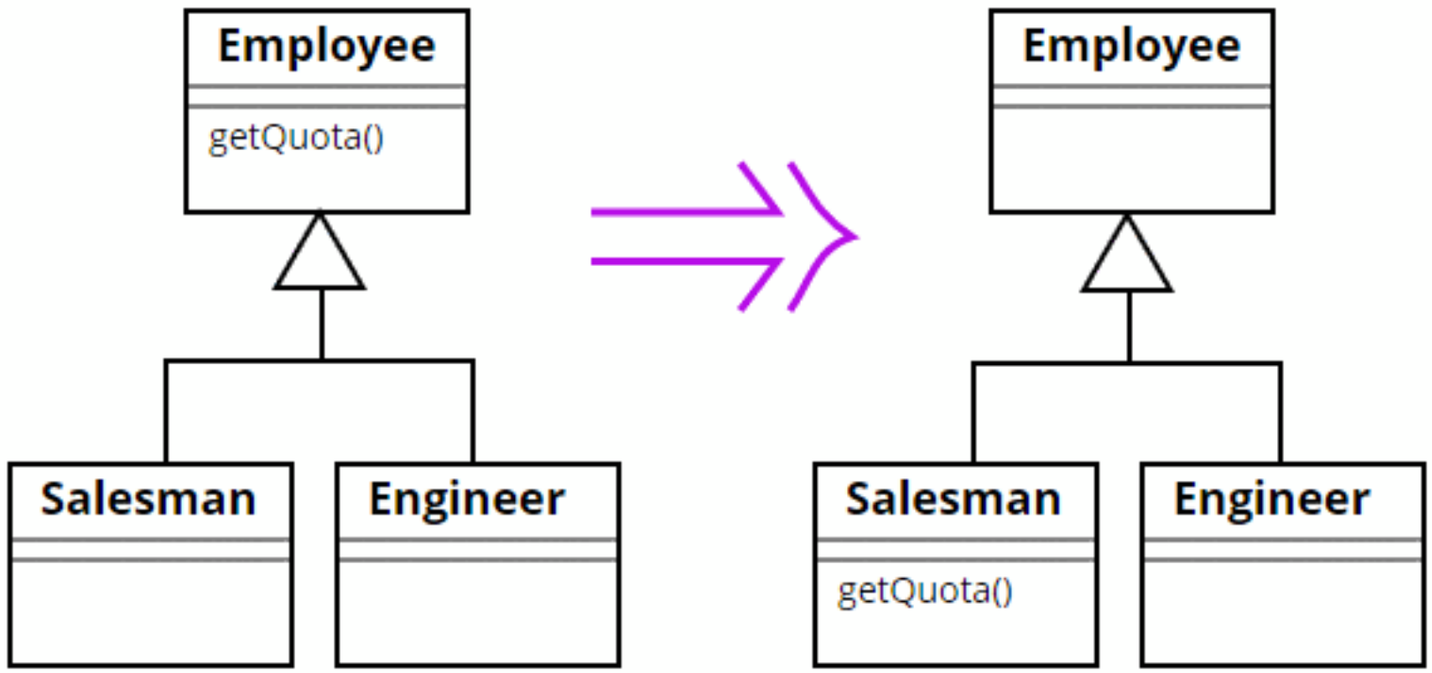
СПУСК МЕТОДА (PUSH DOWN METHOD)

Метод в базовом классе, используется только одним или несколькими подклассами.



Переместите этот метод в подклассы.

СПУСК МЕТОДА (PUSH DOWN METHOD)



ПРИЧИНЫ РЕФАКТОРИНГА

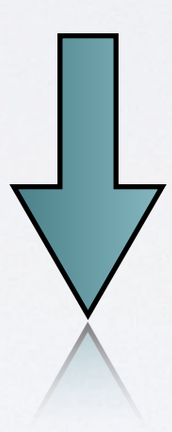
Метод перестал быть уникальным. Например, после извлечения (или удаления) части функциональности из иерархии классов.

ДОСТОИНСТВА

Улучшает связность внутри классов. Метод находится там, где вы ожидаете его увидеть.

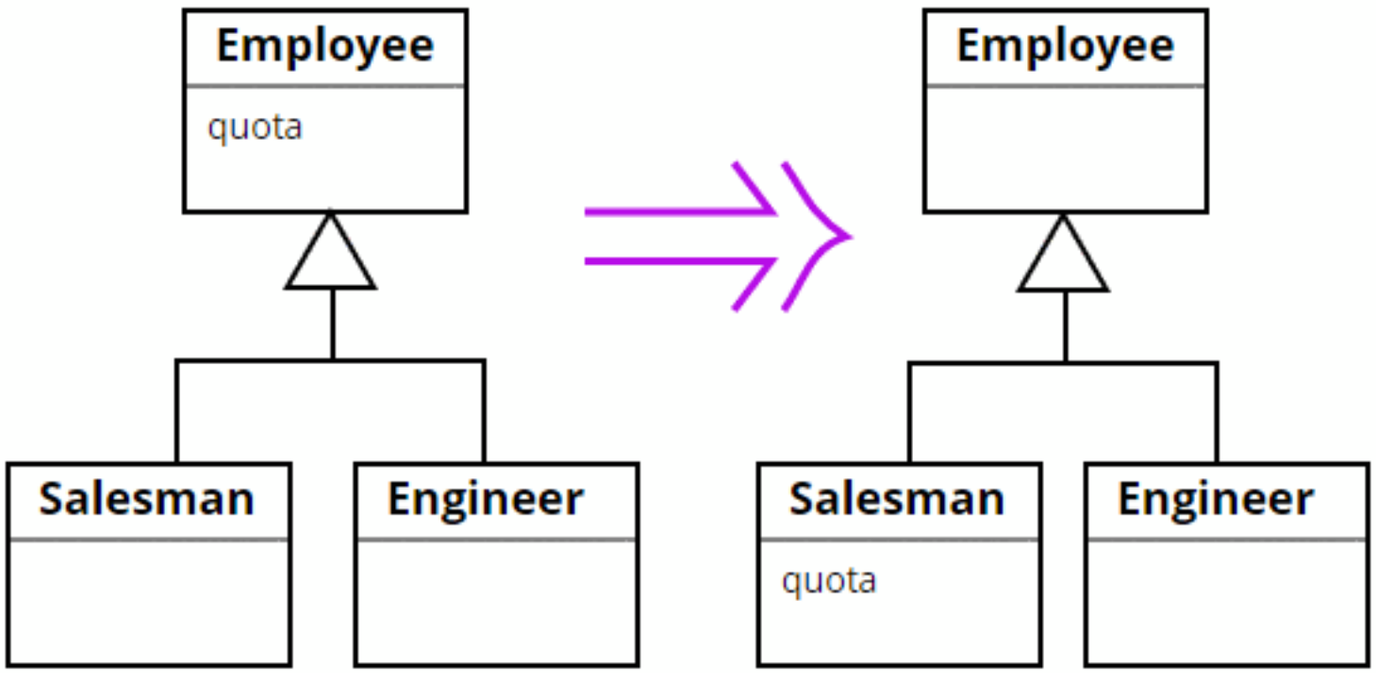
СПУСК ПОЛЯ (PUSH DOWN FIELD)

Поле используется только в некоторых подклассах.



Переместите поле в эти подклассы.

СПУСК ПОЛЯ (PUSH DOWN FIELD)



ПРИЧИНЫ РЕФАКТОРИНГА

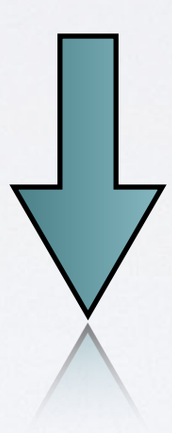
Поле перестало быть уникальным. Например, после извлечения (или удаления) части функциональности из иерархии классов.

ДОСТОИНСТВА

- **Улучшает связность внутри классов. Поле находится там, где оно используется.**
- **Возможность развивать поля независимо друг от друга.**

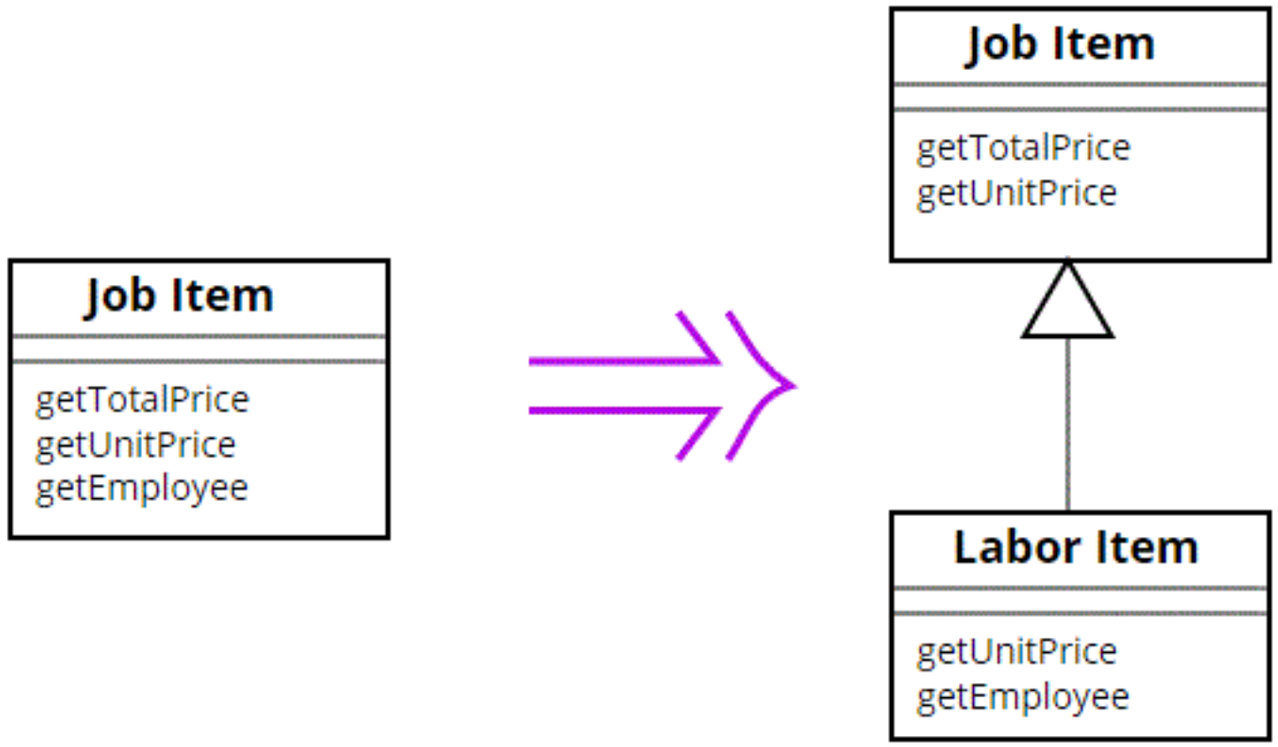
ИЗВЛЕЧЕНИЕ ПОДКЛАССА (EXTRACT SUBCLASS)

Класс имеет фичи, которые используются только в определённых случаях.



Создайте подкласс и используйте его в этих случаях.

ИЗВЛЕЧЕНИЕ ПОДКЛАССА (EXTRACT SUBCLASS)



ПРИЧИНЫ РЕФАКТОРИНГА

В основном классе находятся методы и поля для реализации какого-то редкого случая использования класса.

ДОСТОИНСТВА

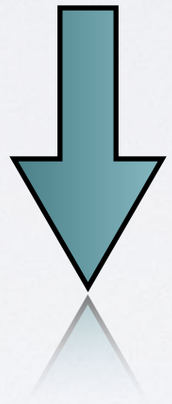
- **Создать подкласс довольно легко и быстро.**
- **Можно выделить несколько разных подклассов, если основной класс реализует несколько подобных особых случаев.**

НЕДОСТАТКИ

Несмотря на всю очевидную простоту, *Наследование* может завести в тупик, если придётся выделить несколько различных иерархий классов.

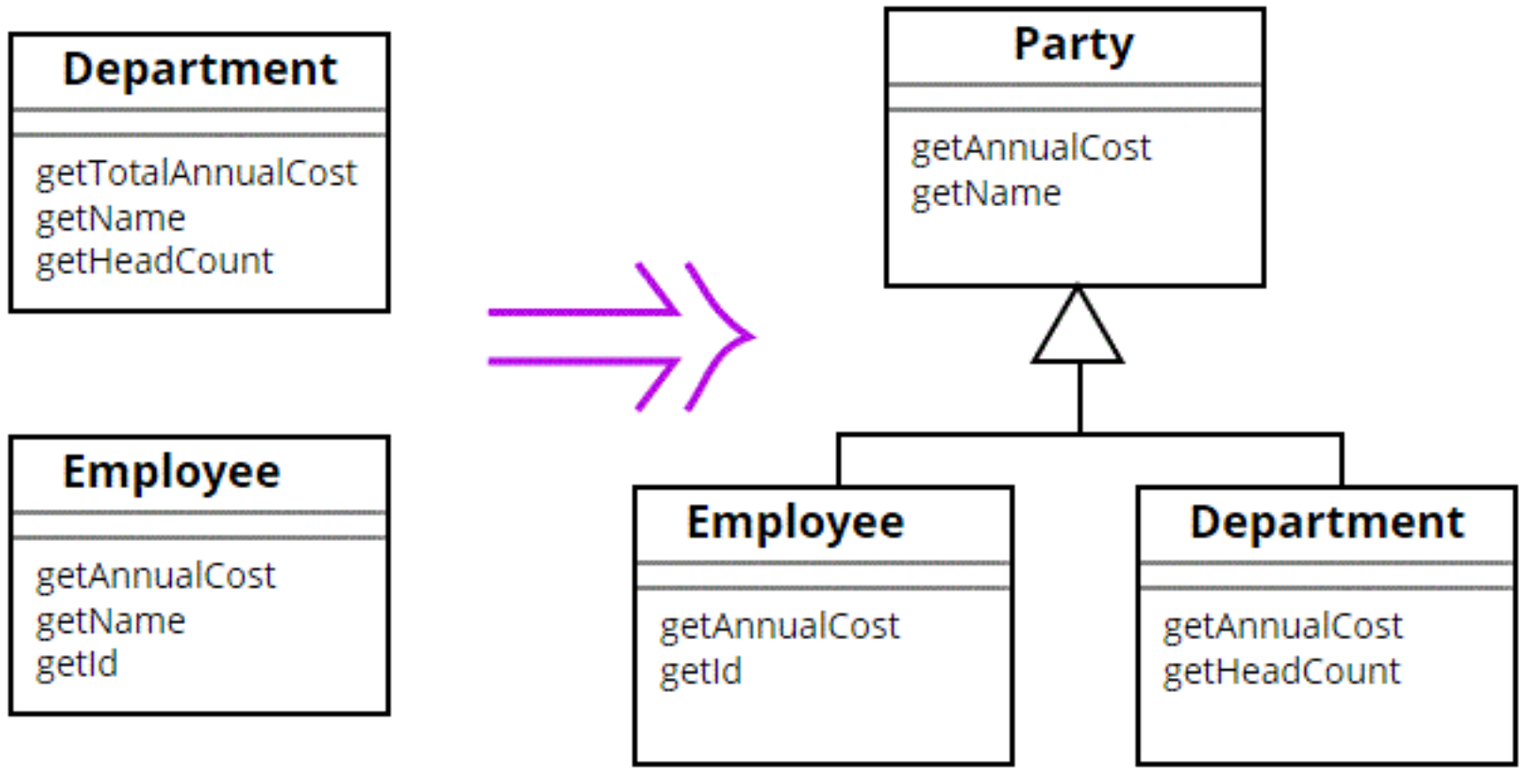
ИЗВЛЕЧЕНИЕ СУПЕРКЛАССА (EXTRACT SUPERCLASS)

Есть два класса с общими полями и методами.



Создайте для них базовый класс и перенесите туда одинаковые поля и методы.

ИЗВЛЕЧЕНИЕ СУПЕРКЛАССА (EXTRACT SUPERCLASS)



ПРИЧИНЫ РЕФАКТОРИНГА

Дублирование кода.

ДОСТОИНСТВА

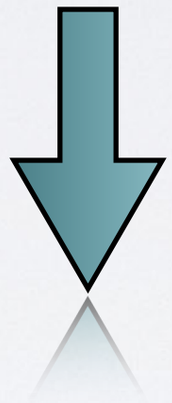
Убирает дублирование кода. Все находится в одном месте.

КОГДА НЕЛЬЗЯ ПРИМЕНИТЬ

Нельзя применить к классам, которые уже имеют базовый класс.

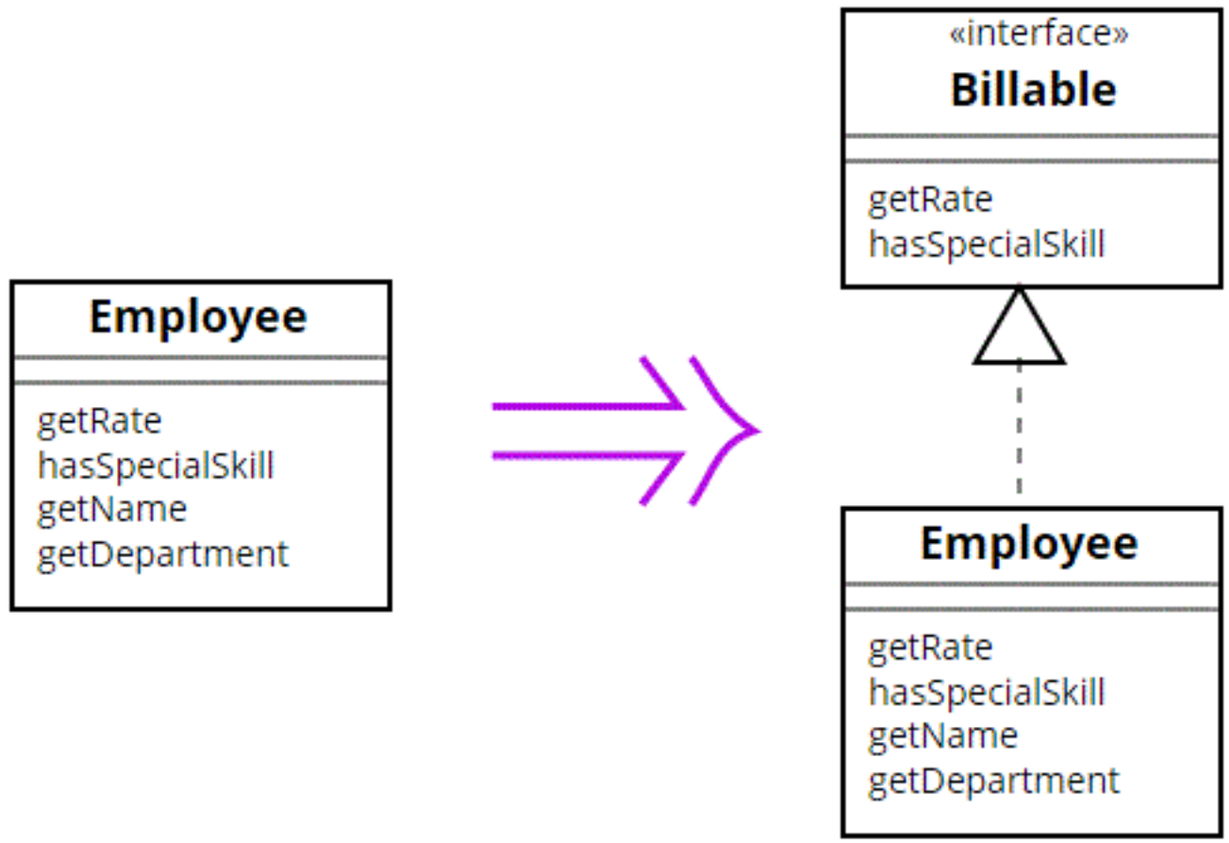
ИЗВЛЕЧЕНИЕ ИНТЕРФЕЙСА (EXTRACT INTERFACE)

В двух классах часть интерфейса оказалась общей.



Выделите эту общую часть в интерфейс.

ИЗВЛЕЧЕНИЕ ИНТЕРФЕЙСА (EXTRACT INTERFACE)



ПРИЧИНЫ РЕФАКТОРИНГА

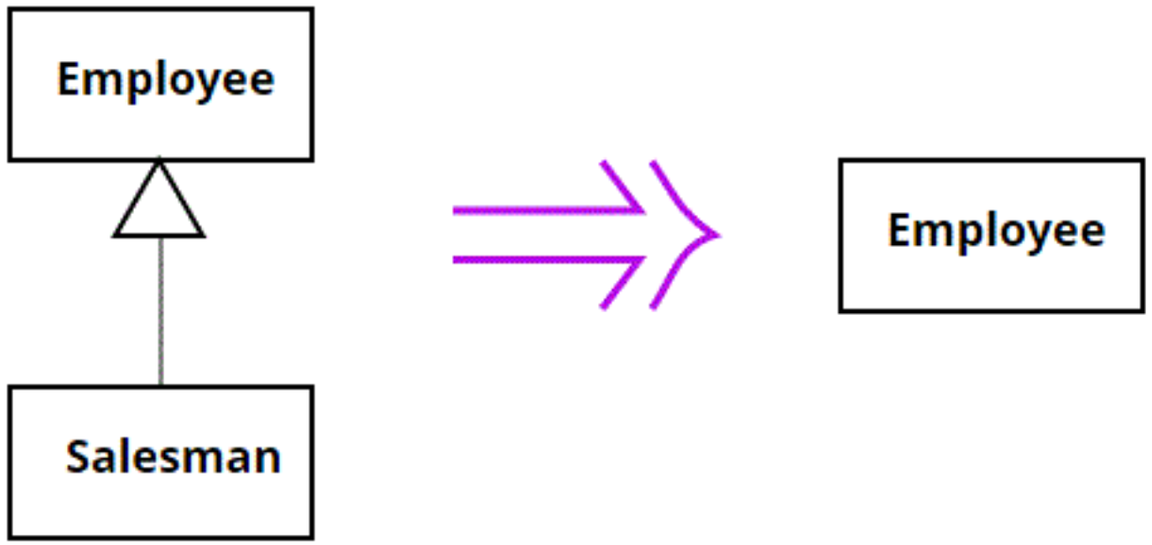
Требуется описать операции, которые выполняют несколько классов.

СВЁРТЫВАНИЕ ИЕРАРХИИ (COLLAPSE HIERARCHY)

Есть некая иерархия классов, в которой подкласс мало чем отличается от базового класса.

Слейте подкласс и базовый класс воедино.

СВЁРТЫВАНИЕ ИЕРАРХИИ (COLLAPSE HIERARCHY)



ПРИЧИНЫ РЕФАКТОРИНГА

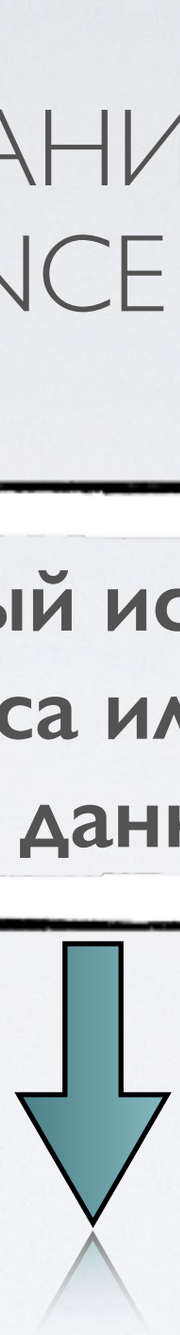
Развитие программы привело к тому, что подкласс и базовый класс стали очень мало отличаться друг от друга.

ДОСТОИНСТВА

- **Уменьшает сложность программы.**
- **Навигация по коду становится проще.**

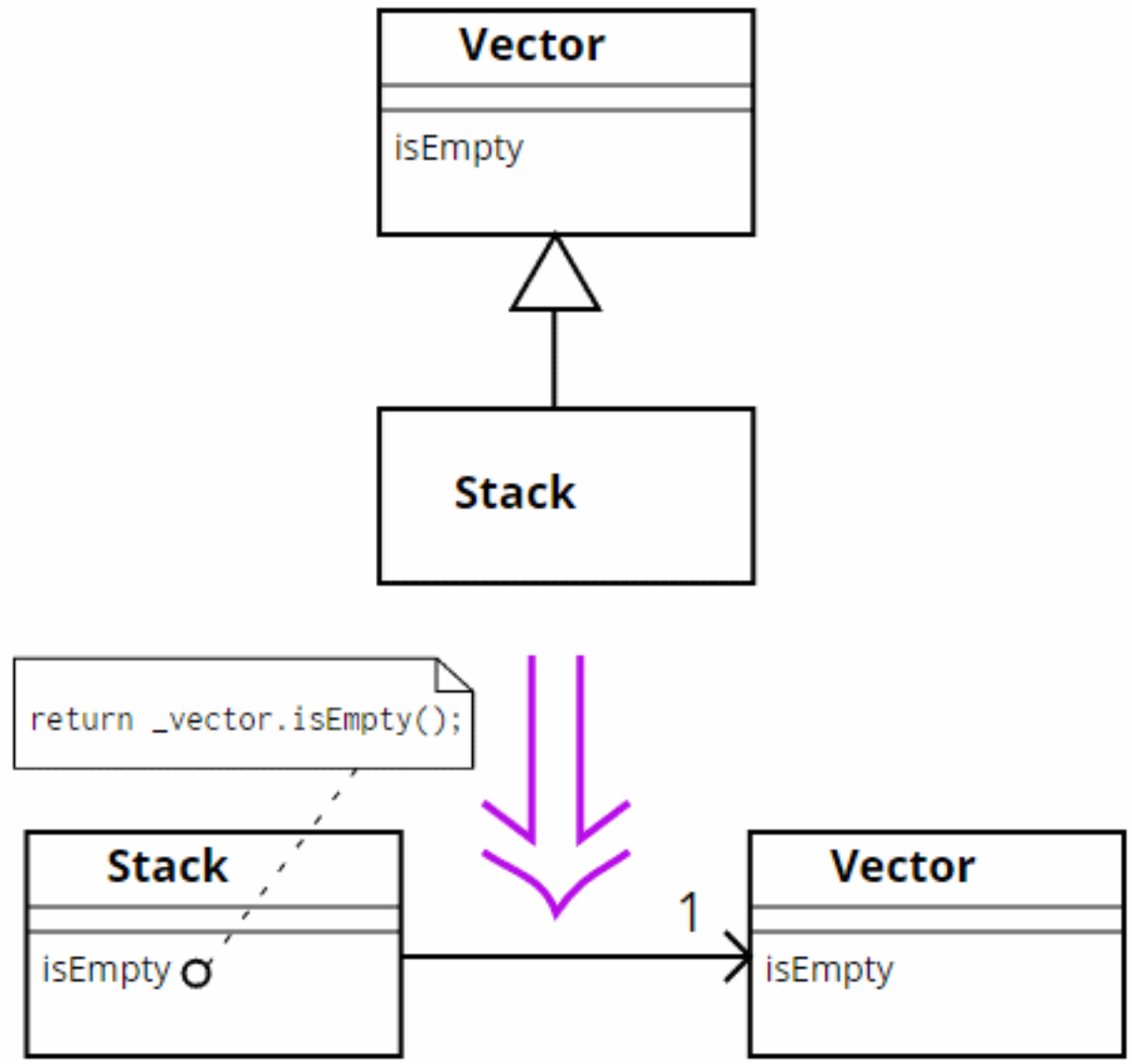
ЗАМЕНА НАСЛЕДОВАНИЯ ДЕЛЕГИРОВАНИЕМ (REPLACE INHERITANCE WITH DELEGATION)

Есть подкласс, который использует только часть методов базового класса или не хочет наследовать его данные.



Создайте поле и поместите в него объект суперкласса, делегируйте выполнение, уберите наследование.

ЗАМЕНА НАСЛЕДОВАНИЯ ДЕЛЕГИРОВАНИЕМ (REPLACE INHERITANCE WITH DELEGATION)



ПРИЧИНЫ РЕФАКТОРИНГА

- **Наследование возникло только ради объединения общего кода.**
- **Подкласс использует только часть методов базового класса.**

ДОСТОИНСТВА

- **Класс не содержит лишних методов, которые достались ему в наследство от суперкласса.**
- **В поле-делегат можно подставлять разные объекты, имеющие различные реализации функциональности.**

НЕДОСТАТКИ

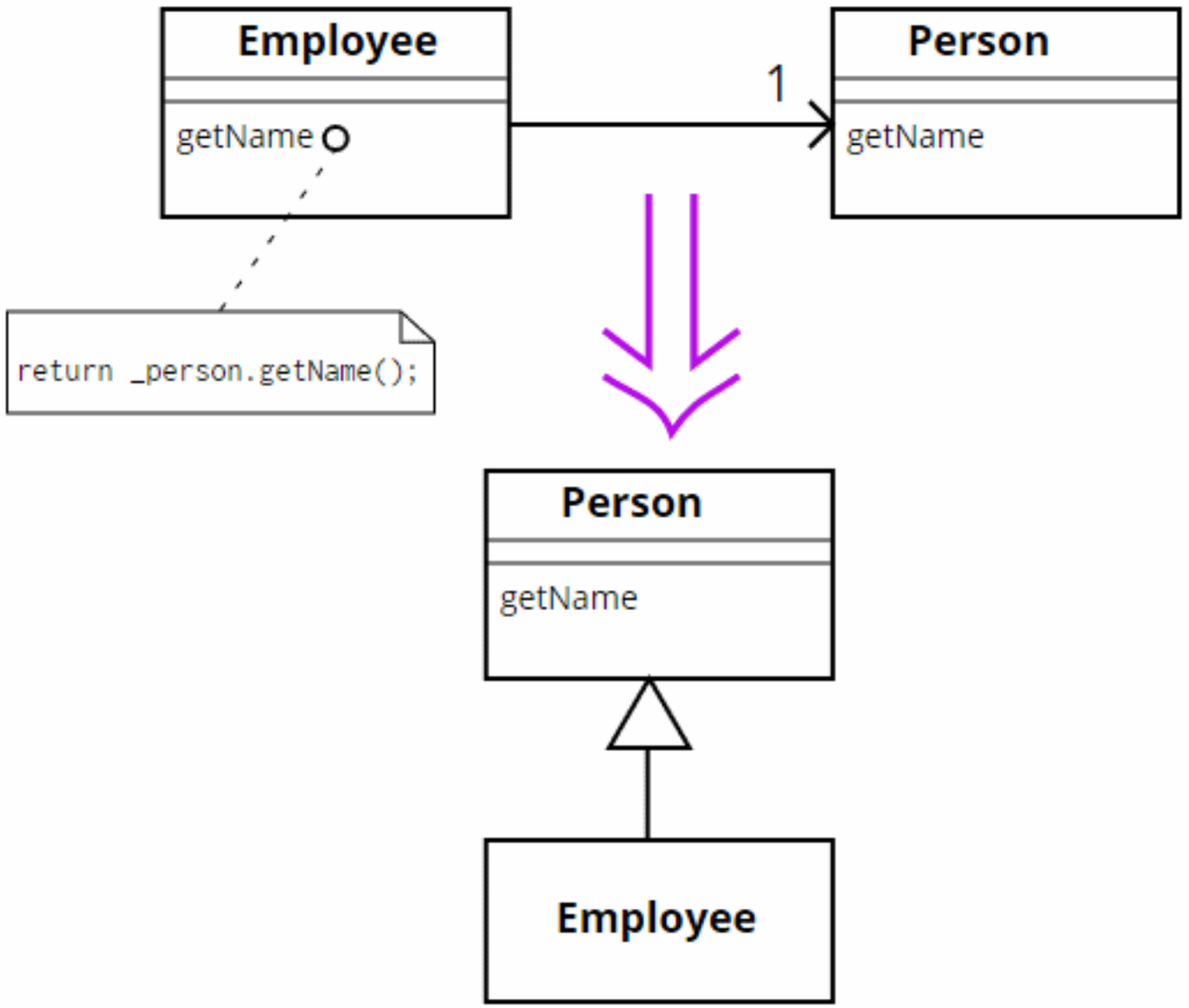
Приходится писать очень много простых делегирующих методов.

ЗАМЕНА ДЕЛЕГИРОВАНИЯ НАСЛЕДОВАНИЕМ (REPLACE DELEGATION WITH INHERITANCE)

Класс содержит множество простых делегирующих методов ко всем методам другого класса.

Сделайте класс наследником делегата, после чего делегирующие методы потеряют смысл.

ЗАМЕНА ДЕЛЕГИРОВАНИЯ НАСЛЕДОВАНИЕМ (REPLACE DELEGATION WITH INHERITANCE)



ПРИЧИНЫ РЕФАКТОРИНГА

Делегирование является более гибким подходом, чем наследование. НО применение делегирования перестаёт быть выгодным, если вы делегируете действия только одному классу, причём всем его публичным методам.

ДОСТОИНСТВА

Уменьшает количество кода.

КОГДА НЕЛЬЗЯ ПРИМЕНИТЬ

- **Если класс содержит делегирование только к части публичных методов класса-делегата.**
- **Если класс имеет родителя.**

CODE SMELLS

РАЗДУВАЛЬЩИКИ

ДЛИННЫЙ МЕТОД (LONG METHOD)

Метод содержит слишком большое число строк кода. Длина метода более десяти строк должна начинать вас беспокоить.

БОЛЬШОЙ КЛАСС (LARGE CLASS)

Класс содержит множество полей/методов/строк кода.

ОДЕРЖИМОСТЬ ЭЛЕМЕНТАРНЫМИ ТИПАМИ (PRIMITIVE OBSESSION)

- **Использование элементарных типов вместо маленьких объектов для небольших задач.**
- **Использование строковых констант в качестве названий полей в массивах.**

ДЛИННЫЙ СПИСОК ПАРАМЕТРОВ (LONG PARAMETER LIST)

Количество параметров метода больше трёх-четырёх.

ГРУППЫ ДАННЫХ (DATA CLUMPS)

В разных частях кода встречаются одинаковые группы переменных.

УТЯЖЕЛИТЕЛИ ИЗМЕНЕНИЙ

РАСХОДЯЩИЕСЯ МОДИФИКАЦИИ (DIVERGENT CHANGE)

При внесении изменений в класс приходится изменять большое число различных методов.

СТРЕЛЬБА ДРОБЬЮ (SHOTGUN SURGERY)

При выполнении любых модификаций приходится вносить множество мелких изменений в большое число классов.

ЗАМУСОРИВАТЕЛИ

КОММЕНТАРИИ (COMMENTS)

Метод содержит множество поясняющих комментариев.

ДУБЛИРОВАНИЕ КОДА (DUPLICATE CODE)

Два фрагмента кода выглядят почти одинаковыми.

МЁРТВЫЙ КОД (DEAD CODE)

Переменная, параметр, поле, метод или класс больше не используются.

ТЕОРЕТИЧЕСКАЯ ОБЩНОСТЬ (SPECULATIVE GENERALITY)

Переменная, параметр, поле, метод или класс больше не используются.

ОПУТЫВАТЕЛИ СВЯЗЯМИ

ЗАВИСТЛИВЫЕ ФУНКЦИИ (FEATURE ENVY)

Метод обращается к данным другого объекта чаще, чем к собственным данным.

НЕУМЕСТНАЯ БЛИЗОСТЬ (INAPPROPRIATE INTIMACY)

Один класс использует служебные поля и методы другого класса.

ЦЕПОЧКА ВЫЗОВОВ (MESSAGE CHAINS)

В коде присутствуют большие цепочки вызовов.

ПОСРЕДНИК (MIDDLE MAN)

Если класс выполняет одно действие – делегирует работу другому классу – стоит задуматься, зачем он вообще существует.

КОНЕЦ ТРЕТЬЕЙ ЧАСТИ