Software Engineering

Лекция 13: Рефакторинг

Тимофей Брыксин
timofey.bryksin@gmail.com

Рефакторинг

- Изменение во внутренней структуре программного обеспечения, имеющее целью облегчить понимание его работы и упростить модификацию, не затрагивая наблюдаемого поведения
  - приведение кода в порядок
  - борьба с деградацией архитектуры
  - **не** связан с оптимизацией работы кода
- Декомпозиция на большое количество элементарных действий
- Альтернатива проектированию архитектуры? 🤔

Зачем нужно делать рефакторинг

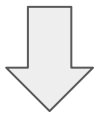
- Улучшение структуры программного обеспечения
- Облегчение понимания кода
- Помощь в поиске ошибок
- Ускорение разработки нового кода
- Изменение культуры кодирования

Code smells

- Дублирование кода
- Длинный метод
- Большой класс
- Длинный список параметров
- Нарушение SRP
- «Стрельба дробью»
- «Завистливые функции»
- Группы данных
- Операторы типа switch
- «Ленивый класс»
- Временное поле
- Цепочки сообщений
- Неуместная близость
- Классы данных
- Комментарии

Выделение метода (Extract Method)

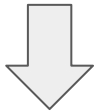
```
1.  void printOwing(double amount) {
2.      printBanner();
3.      // вывод деталей
4.      System.out.println (“name: ” + _name);
5.      System.out.println (“amount ” + amount);
6.  }
```



```
1.  void printOwing( double amount) {
2.      printBanner();
3.      printDetails(amount);
4.  }
5.  void printDetails (double amount) {
6.      System.out.println (“name: ” + _name);
7.      System.out.println (“amount ” + amount);
8.  }
```

Встраивание метода (Inline Method)

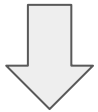
```
1.  int getRating() {
2.      return (moreThanFiveLateDeliveries()) ? 2 : 1;
3.  }
4.  boolean moreThanFiveLateDeliveries() {
5.      return _numberOfLateDeliveries > 5;
6.  }
```



```
1.   int getRating() {
2.       return (_numberOfLateDeliveries > 5) ? 2 : 1;
3.   }
```

Введение поясняющей переменной

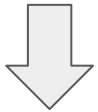
```
1.  if ( (platform.toUpperCase().indexOf(“MAC”) > -1 )
2.      && (browser.toUpperCase().indexOf(“IE”) > -1 )
3.      && wasInitialized() && resize > 0 ) {
4.
5.      // do something
6.  }
```



```
1.  final boolean isMacOS = platform.toUpperCase().indexOf(“MAC”) > -1;
2.  final boolean isIEBrowser = browser.toUpperCase().indexOf(“IE”) > -1;
3.  final boolean isResized = resize > 0;
4.  if(isMacOS && isIEBrowser && wasInitialized() && isResized) {
5.      // do something
6.  }
```

Декомпозиция условного оператора

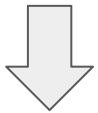
```
1.  if (date.before (SUMMER_START) || date.after(SUMMER_END))
2.      charge = quantity * _winterRate + _winterServiceCharge;
3.  else
4.      charge = quantity * _summerRate;
```



```
1.  if (notSummer(date))
2.      charge = winterCharge(quantity);
3.  else
4.      charge = summerCharge (quantity);
```

Расщепление временной переменной

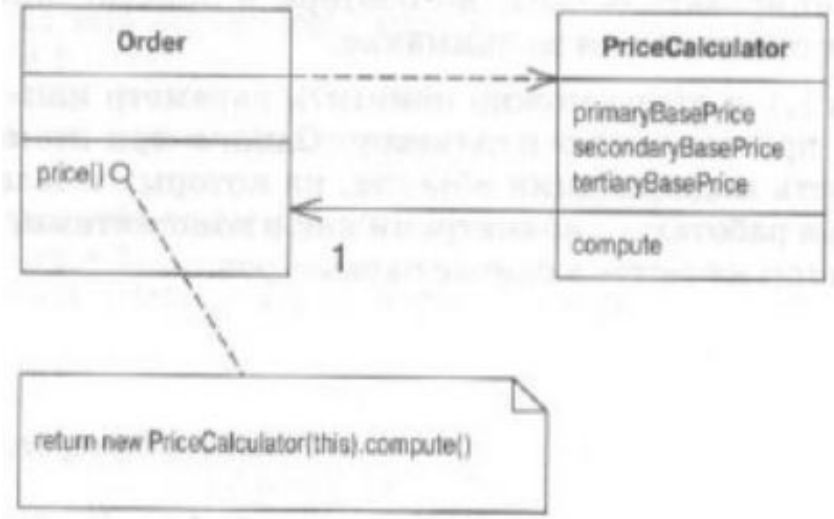
```
1.  double temp = 2 * (_height + _width);
2.  System.out.println (temp);
3.  temp = _height * _width;
4.  System.out.println (temp);
```



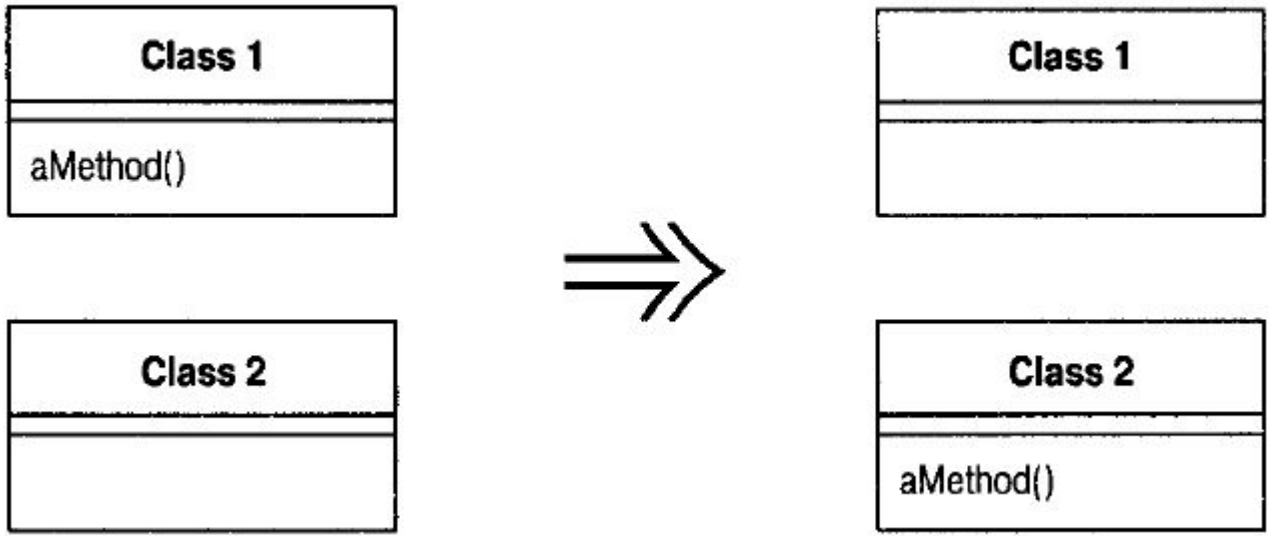
```
1.  final double perimeter = 2 * (_height + _width);
2.  System.out.println (perimeter);
3.  final double area = _height * _width;
4.  System.out.println (area);
```

Замена метода объектом

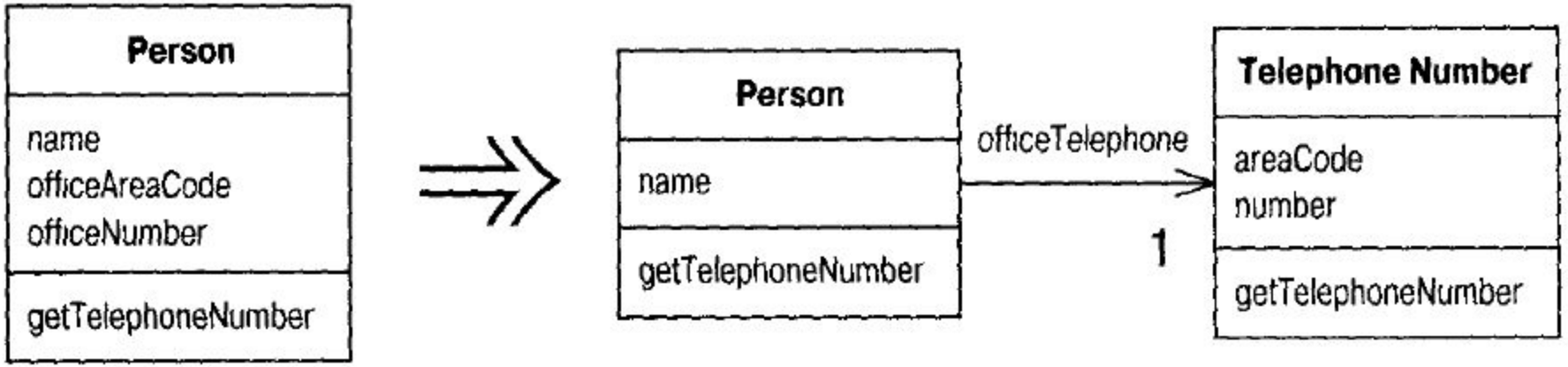
```
1.  class Order {
2.      double price() {
3.          double primaryBasePrice;
4.          double secondaryBasePrice;
5.          double tertiaryBasePrice;
6.          // длинные вычиспвния;
7.      }
8.  }
```



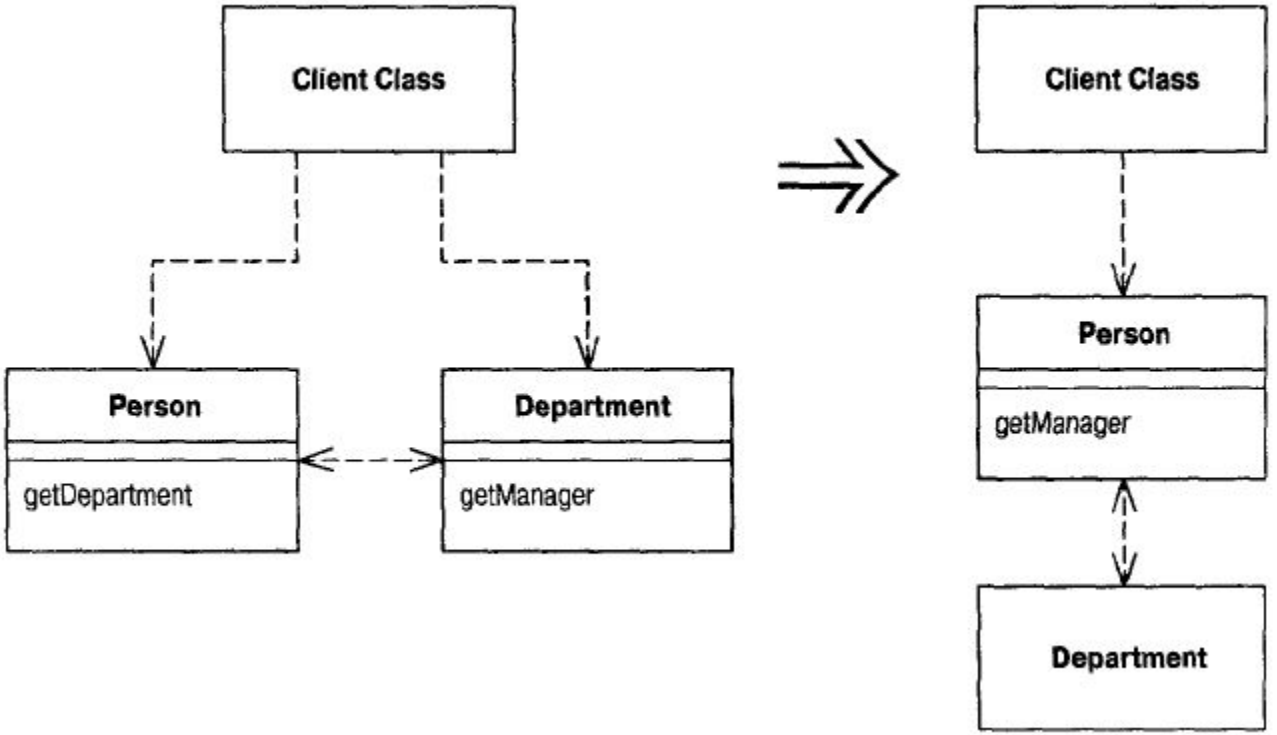
Перемещение метода (Move Method)



Выделение класса (Extract Class)

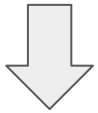


Сокрытие делегирования (Hide Delegate)



Введение внешнего метода

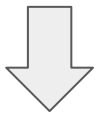
```
1.  Date newStart = new Date (previousEnd.getYear(),
2.      previousEnd.getMonth(), previousEnd.getDate()+ 1;
```



```
1.  Date newStart = nextDay(previousEnd);
2.  static Date nextDay(Date arg) {
3.      return new Date (arg.getYear(), arg.getMonth(), arg.getDate() + 1);
4.  }
```

Самоинкапсуляция поля

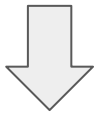
```
1.  private int _loW, _high;
2.  boolean includes (int arg) {
3.      return arg >= _low && arg <= _high;
4.  }
```



```
1.  private int _low, _high;
2.  int getLow() {return _low;}
3.  int getHigh() {return _high;}
4.  boolean includes (int arg) {
5.      return arg >= getLow() && arg <= getHigh();
6.  }
```

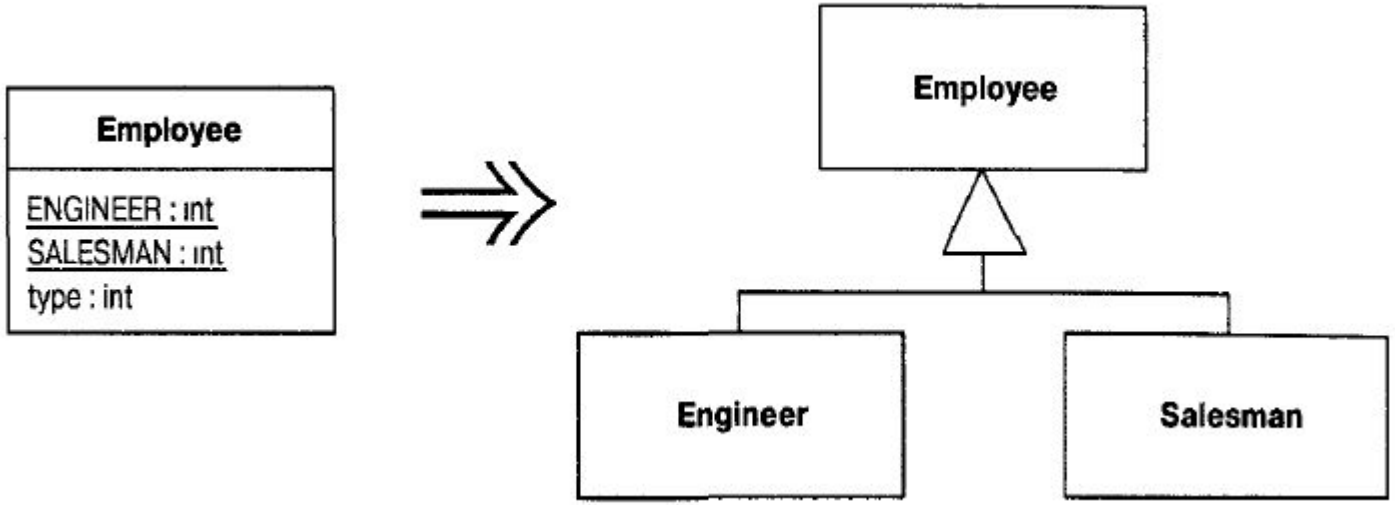
Замена магического числа именованной константой

```
1.  double potentialEnergy(double mass, double height) {
2.      return mass * 9.81 * height;
3.  }
```



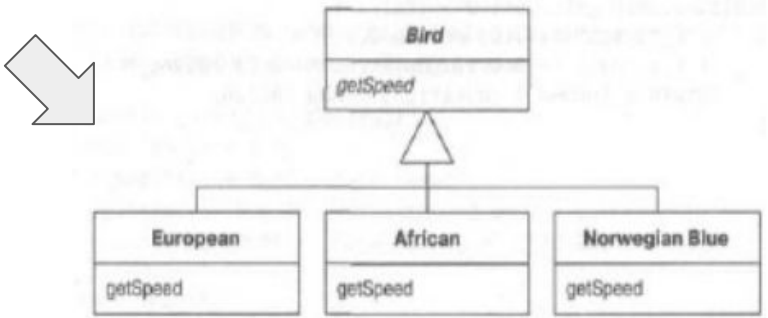
```
1.  double potentialEnergy(double mass, double height) {
2.      return mass * GRAVITATIONAL_CONSTANT * height;
3.  }
4.  static final double GRAVITATIONAL_CONSTANT = 9.81;
```

Замена кода типа подклассами



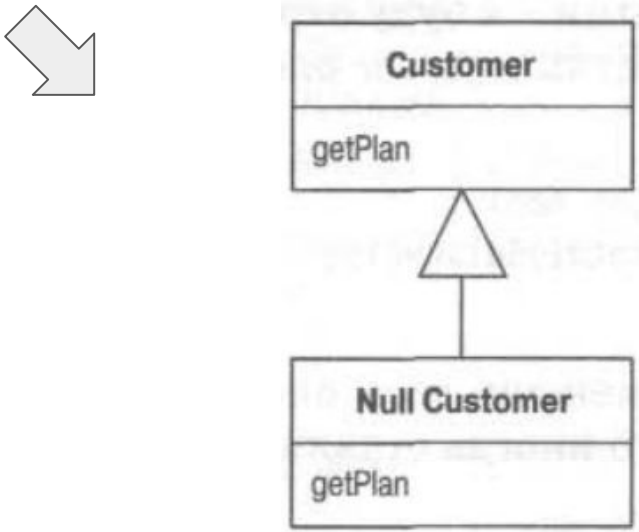
Замена условного оператора полиморфизмом

```
1.  double getSpeed() {
2.      switch (_type) {
3.          case EUROPEAN:
4.              return getBaseSpeed();
5.          case AFRICAN:
6.              return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
7.          case NORWEGIAN_BLUE:
8.              return (_isNailed) ? 0 : getBaseSpeed(_voltage);
9.      }
10.     throw new RuntimeException ("Should be unreachable");
11. }
```

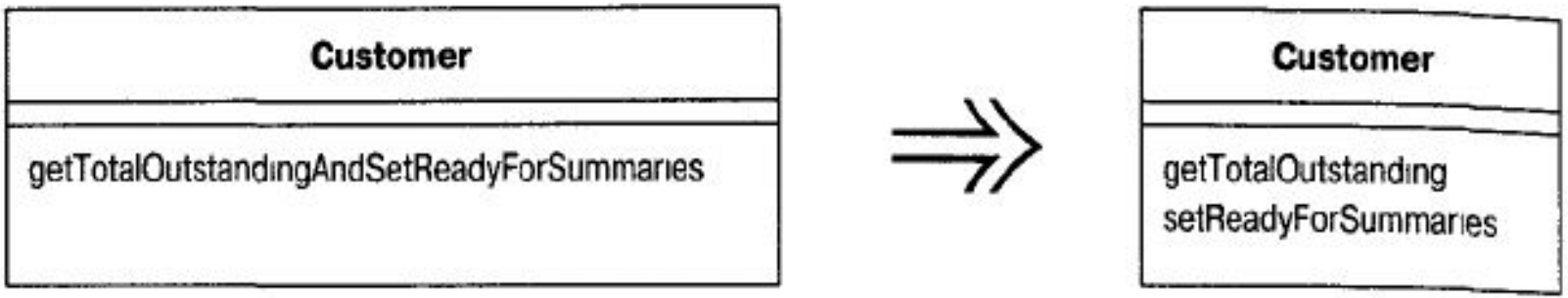


Введение Null-объекта (Introduce Null Object)

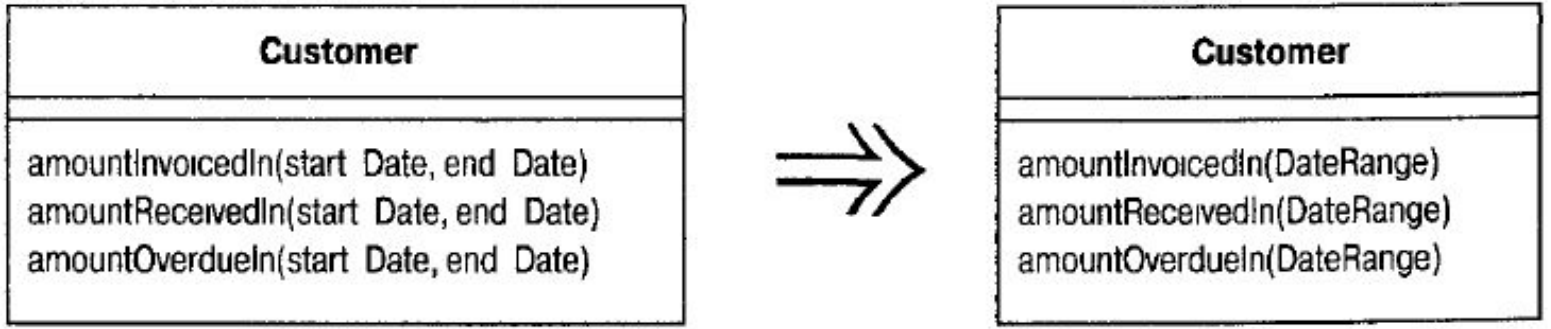
```
1.  if (customer == null)
2.      plan = BillingPlan.basic();
3.  else
4.      plan = customer.getPlan();
```



Разделение запроса и модификатора

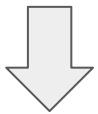


Введение объекта-параметра



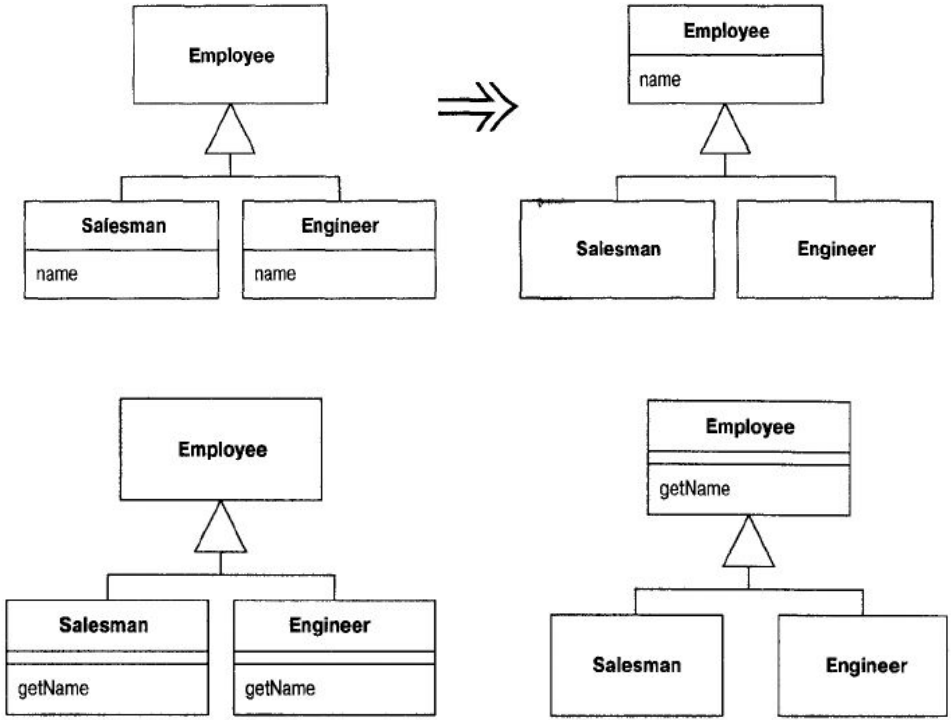
Замена конструктора фабричным методом

```
1.  Employee (int type) {
2.      _type = type;
3.  }
```

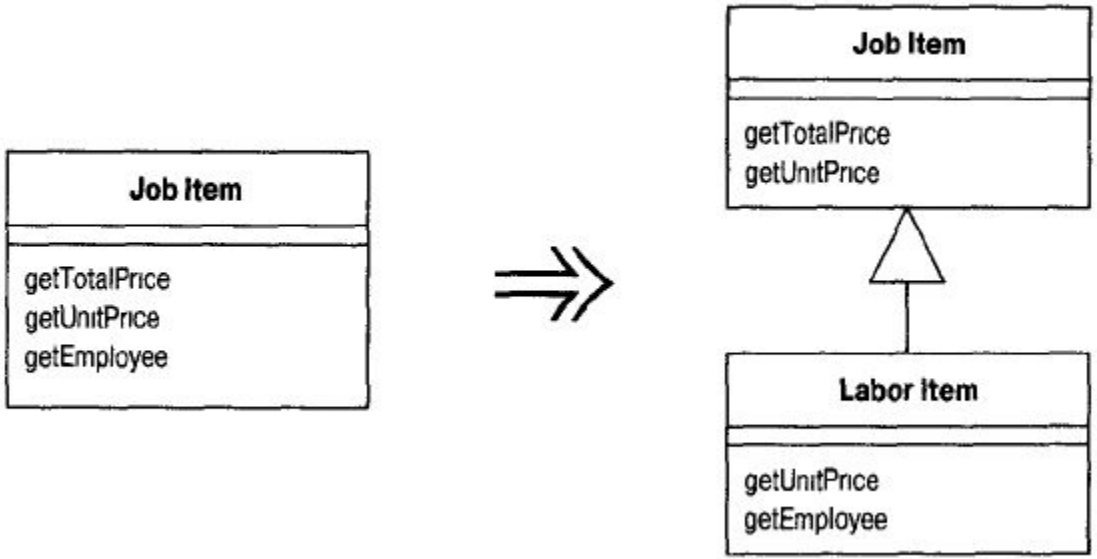


```
1.  static Employee create (int type){
2.      return new Employee(type);
3.  }
```

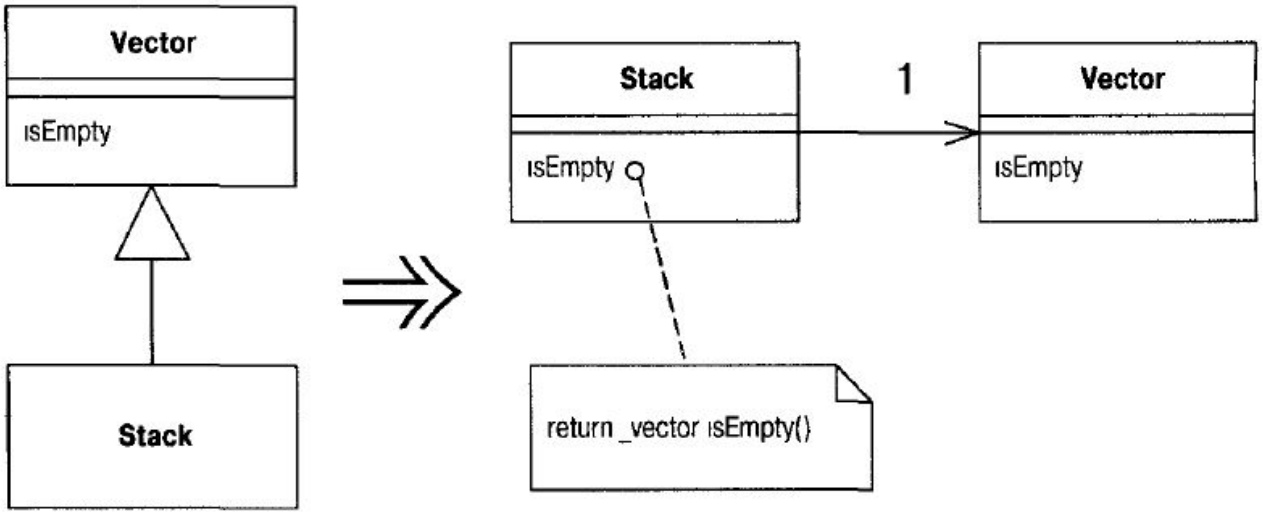
Подъем/спуск поля/метода



Выделение подкласса (Extract Subclass)



Замена наследования делегированием



Когда имеет смысл делать рефакторинг

- Отдельное планирование
- “Правило трёх ударов”
- При добавлении новой функциональности
- При исправлении ошибок
- При изучении и ревью кода

Проблемы при проведении рефакторинга

- Работа с данными
- Изменение интерфейсов сущностей
  - сохранение старого интерфейса
  - методы-обёртки
  - работа с исключениями
- Глобальные изменения архитектуры
- Рефакторинг vs Оптимизация

Когда рефакторинг делать точно не стоит

- Код проще переписать с нуля
- Близость дедлайнов

Что почитать