

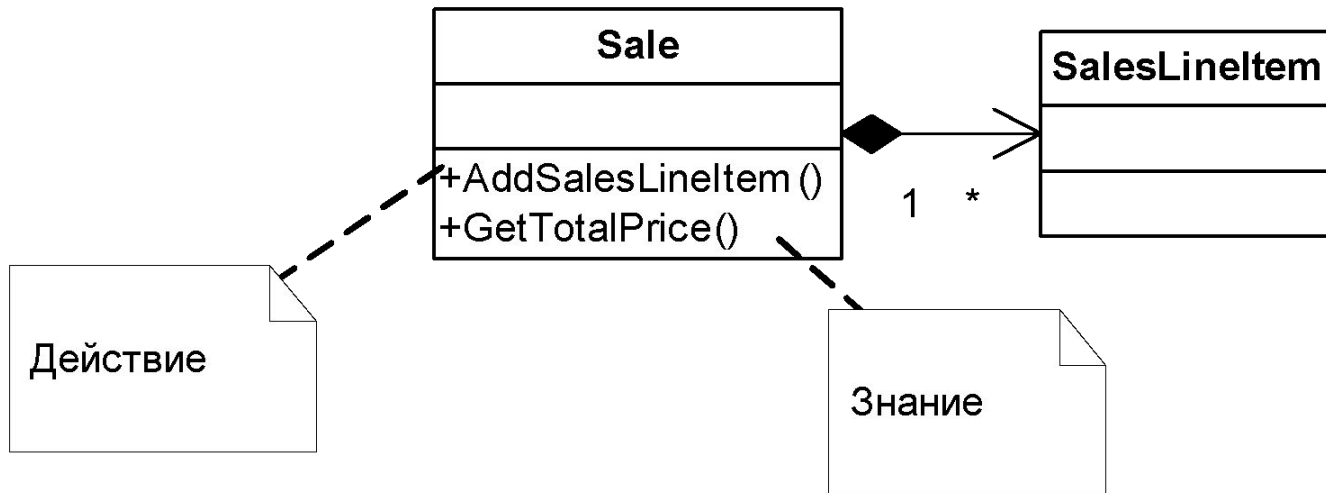
# Распределение обязанностей

С использованием UML

# Обязанности

- Обязанность – контракт или обязательство классификатора.
  - Действие
    - Выполнение действий другим объектом (создание экземпляра или выполнение вычислений)
    - Инициирование действий других объектов
    - Управление действиями других объектов или их координирование
  - Знание
    - Наличие информации о закрытых инкапсулированных данных
    - Наличие информации о связанных объектах
    - Наличие информации о следствиях или вычисляемых величинах

# Обязанности



# Обязанности

Операция =>

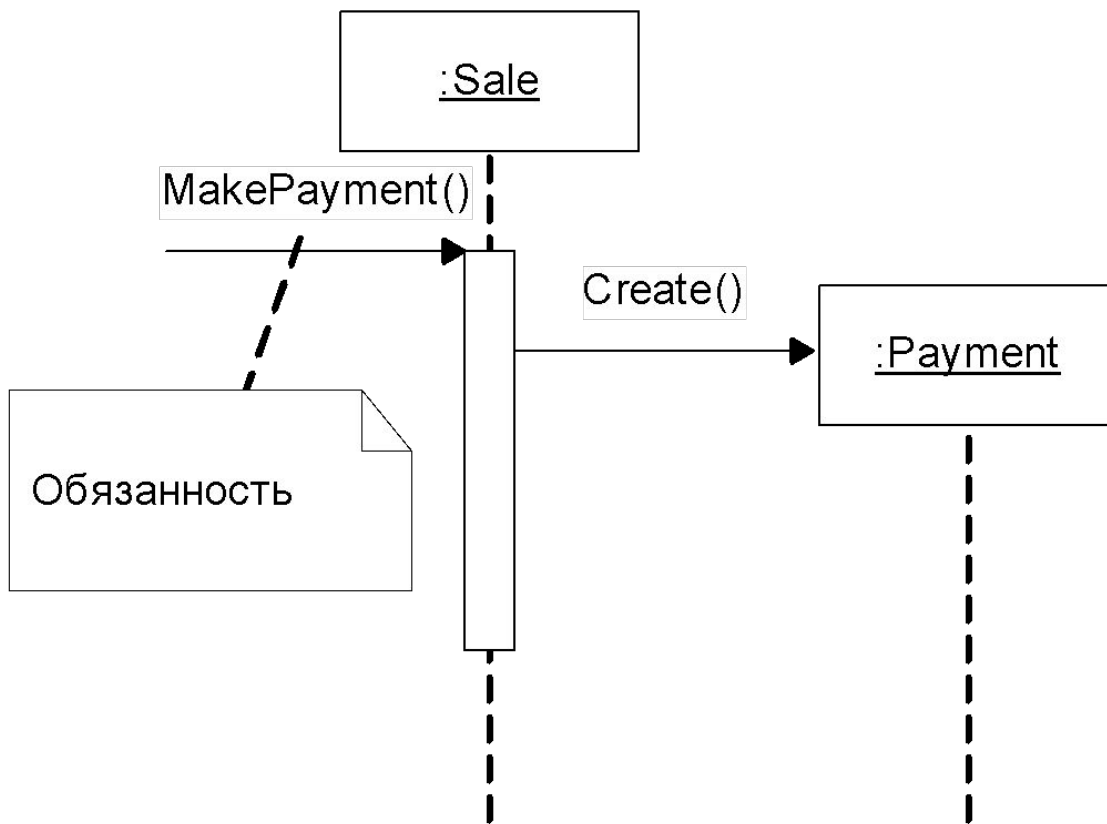
Обязанность

Обязанность  $\neq$

Операция

# Обязанности

на диаграммах взаимодействия



# GRASP

- Шаблоны **GRAS** или **GRASP** – **General Responsibility Assignment Software Patterns**
  - Информационный эксперт (Information Expert)
  - Создатель (Creator)
  - Высокое зацепление (High Cohesion)
  - Слабое связывание (Low Coupling)
  - Контроллер (Controller)
  - Полиморфизм (Polymorphism)
  - Чистая выдумка (Pure Fabrication)
  - Посредник (Indirection)
  - Соккрытие реализации (Protected Variations)

# Информационный эксперт

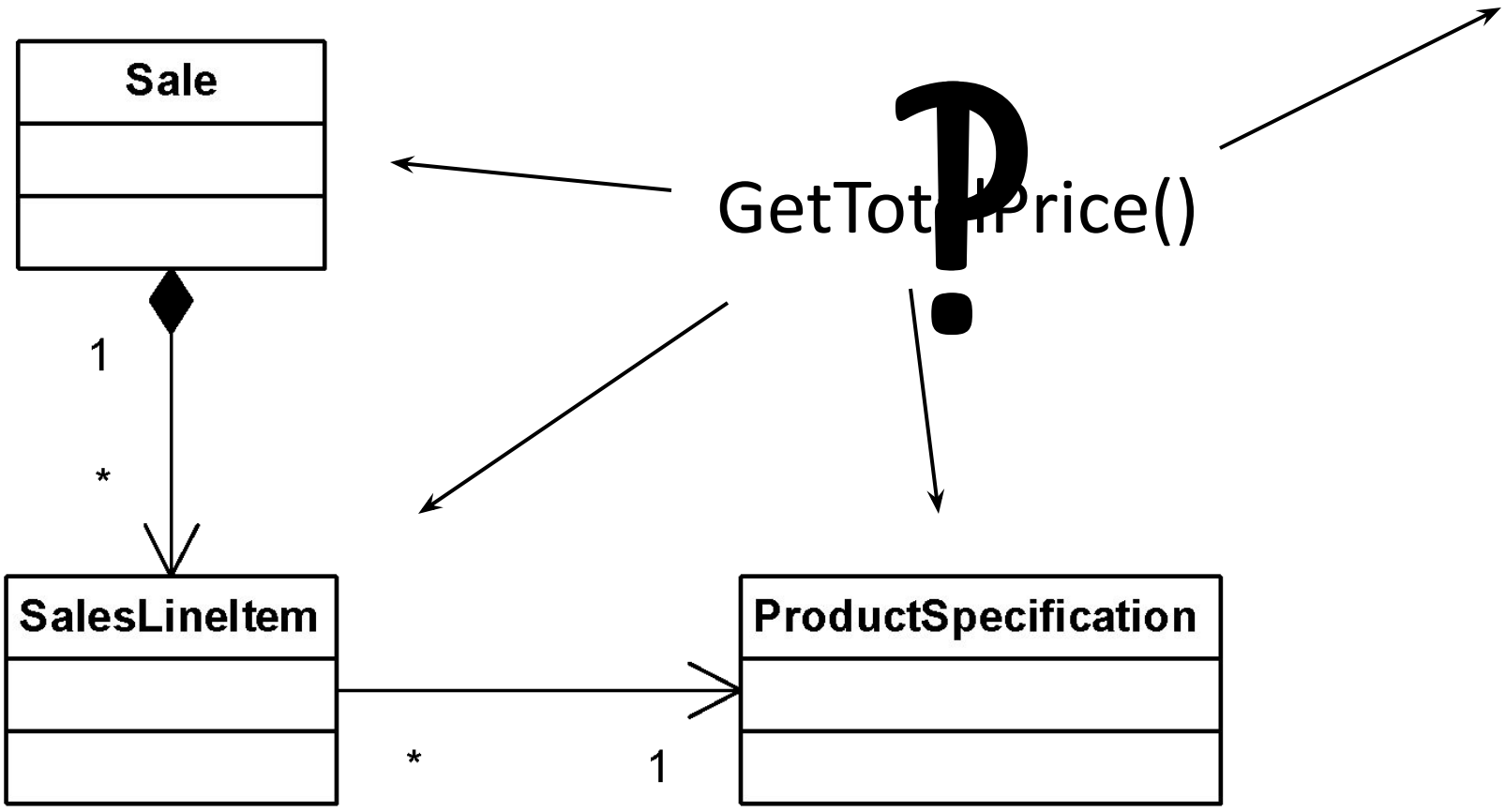
## **Проблема:**

Необходимо выявить наиболее общий принцип распределения обязанностей между объектами.

## **Решение:**

Назначить обязанность информационному эксперту – классу, у которого имеется информация, требуемая для выполнения обязанности.

# Информационный эксперт





# Информационный эксперт



# Информационный эксперт

## Требуемая обязанность:

Знать и предоставлять общую сумму продажи.

## Итоговое распределение обязанностей:

Класс	Обязанность
Sale	Знание общей суммы продажи
SalesListItem	Знание промежуточной суммы для данного товара
ProductSpecification	Знание цены товара

# Информационный эксперт

## Выводы

- Самый часто используемый шаблон распределения обязанностей.
- Аналогия с реальным миром.
- Существование «частичных» экспертов.

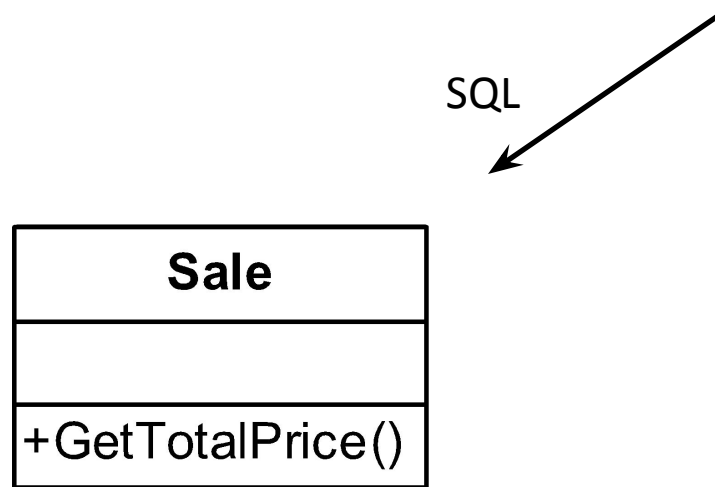
# Информационный эксперт

## Преимущества

- Поддерживает инкапсуляцию
- Поведение обеспечивается несколькими классами, содержащими требуемую информацию => простота понимания и поддержки.

# Информационный эксперт не следует применять

В тех случаях, когда это нарушает  
связывание и сцепление.



# Создатель

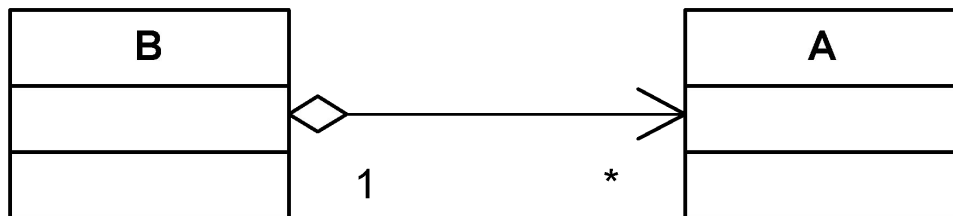
## Проблема:

Кто должен отвечать за создание нового экземпляра некоторого класса?

## Решение:

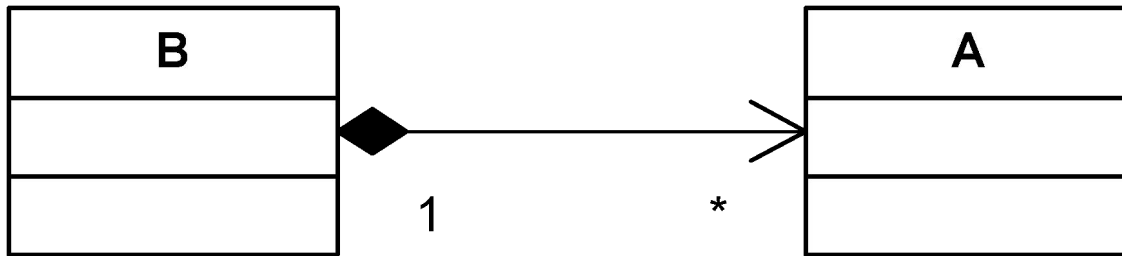
Назначить классу В обязанность создавать экземпляры класса А, если выполняется одно из следующих условий:

- Класс В *агрегирует* объекты А.

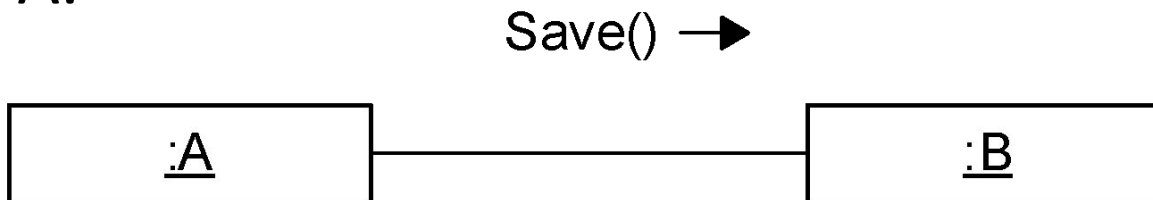


# Создатель

- Класс В *содержит* объекты А.

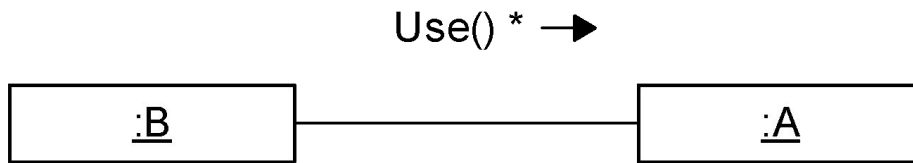


- Класс В *записывает* экземпляры объектов А.

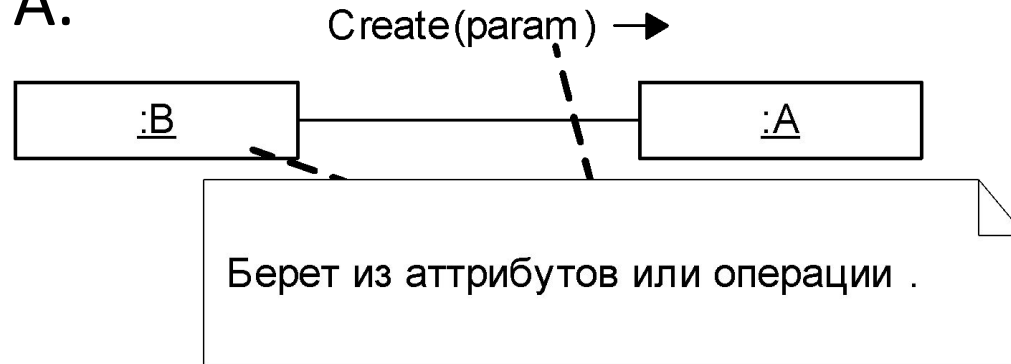


# Создатель

- Класс В *активно использует* объекты А.



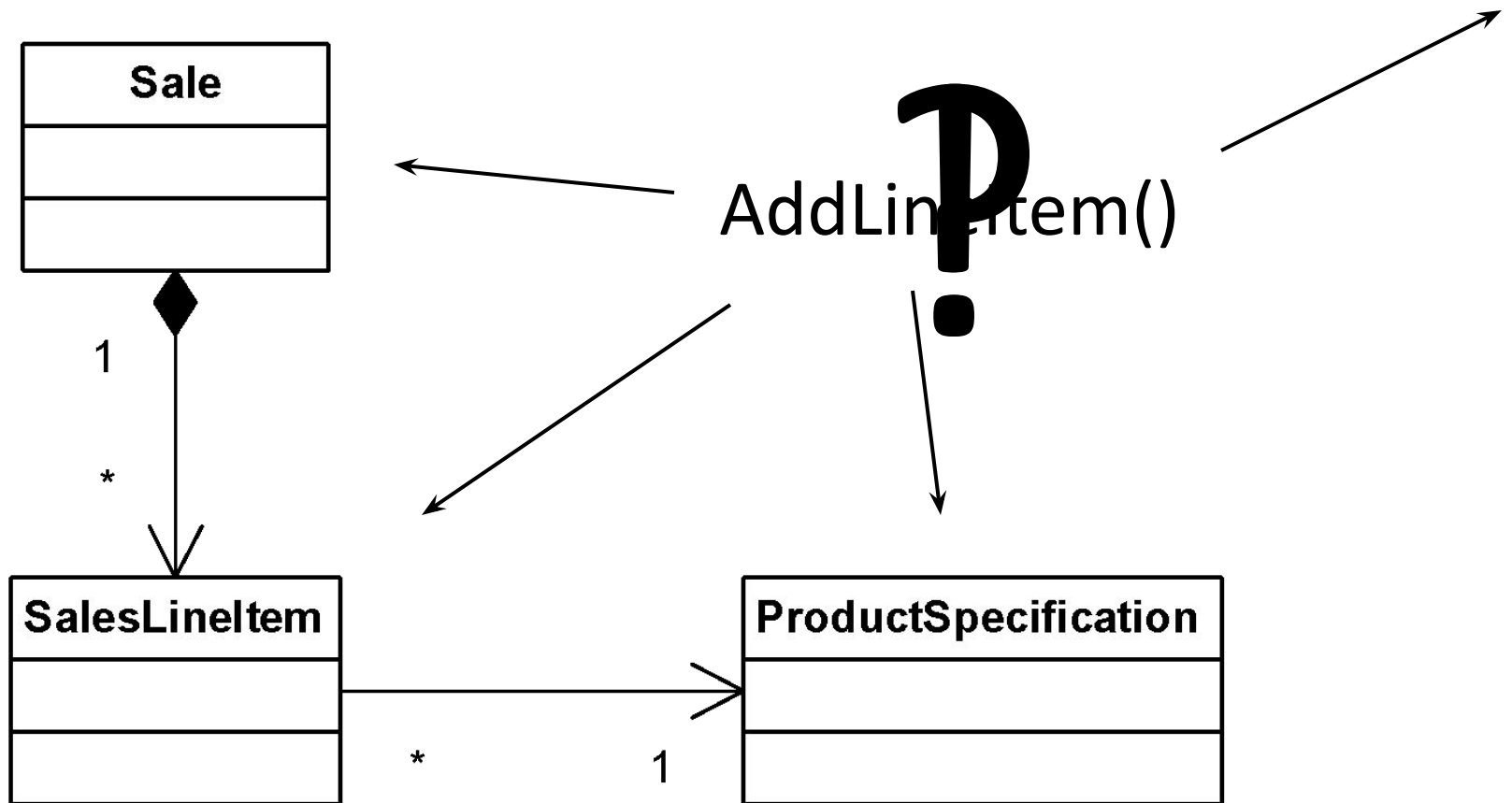
- Класс В *обладает данными инициализации* объектов А.



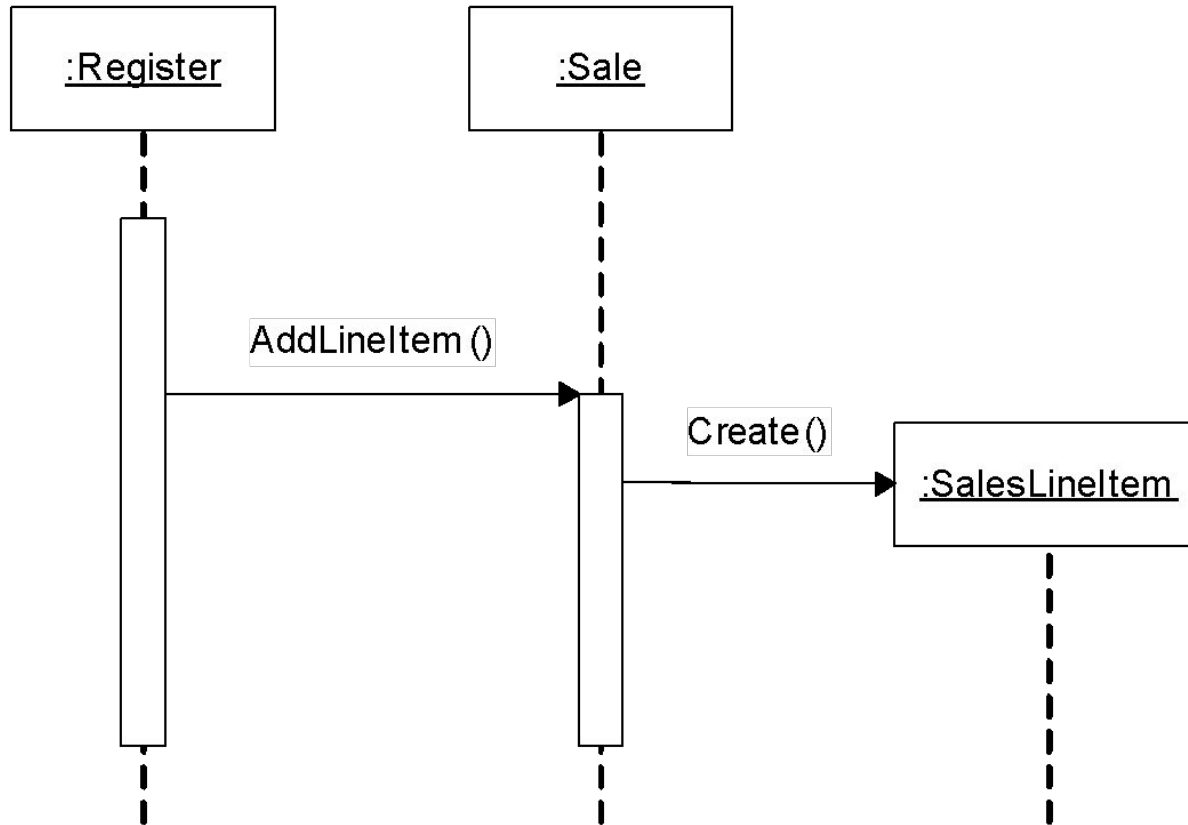
Класс В – *создатель* объектов А.



# Создатель



# Создатель



# Создатель

## Выводы

- Самый распространенный способ распределения обязанностей, связанных с созданием объектов – простота выявления объекта-создателя, при сохранении низкой степени связанности.
- Создатель, содержащий данные инициализации – шаблон Информационный эксперт.

# Создатель не следует применять

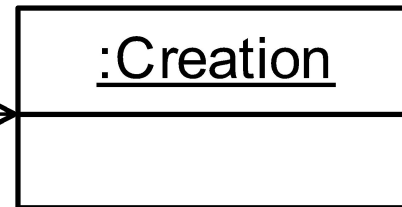
Когда создание – сложная операция, выполняемая при реализации некоторого условия на основе каких-либо внешних свойств.

В этом случае предпочтительнее использовать шаблон Фабрика (Factory).

# Создатель

## Преимущества

- Не повышает степень связанности, так как созданный объект обычно оказывается видимым для объекта-создателя



# Слабое связывание(Low Coupling)

## Проблема:

Как обеспечить минимальную зависимость, минимальный риск изменений и повышенное повторное использование?

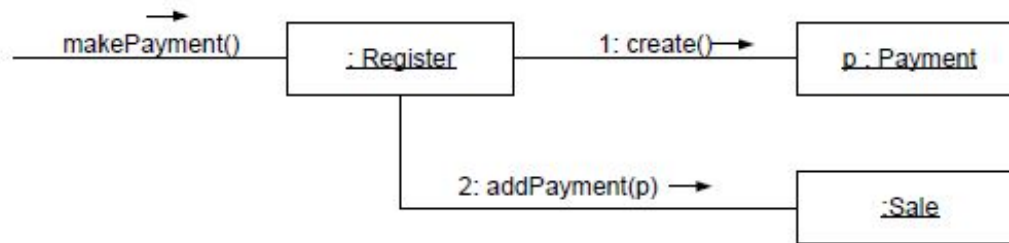
**Связывание** – мера того, насколько сильно элементы связаны, зависят друг от друга.

## **Сильное связывание приводит к проблемам:**

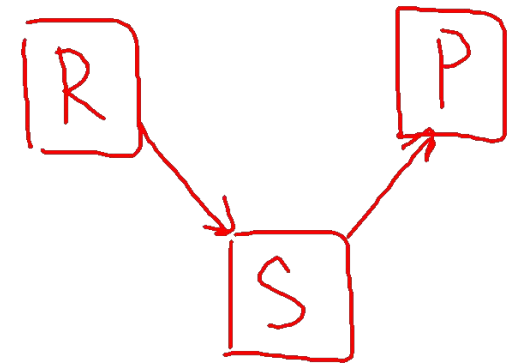
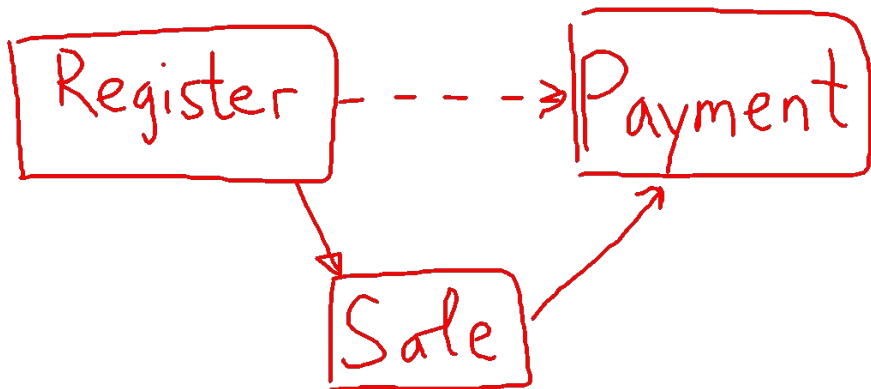
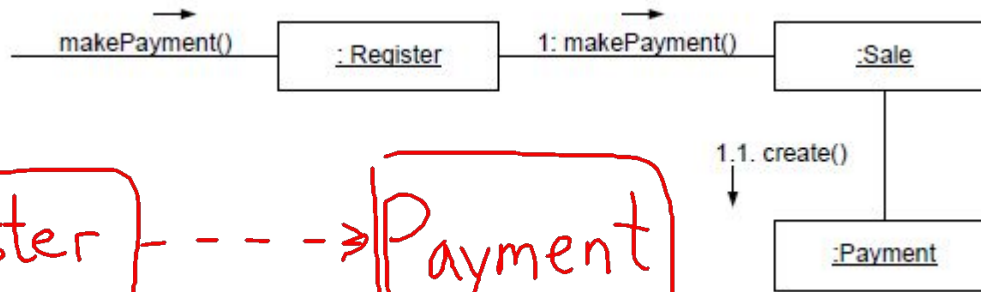
- Изменения в одном классе влекут необходимость в изменении другого
- Сложность понимания элемента в изоляции от других
- Сложность повторного использования

# Слабое связывание (Low Coupling)

Паттерн «Создатель» нам диктует:



Но можно поразмыслить:



# Слабое связывание(Low Coupling)

**Классы ClassX и ClassY могут быть связаны:**

- ClassX имеет атрибут, который ссылается на ClassY
- ClassX содержит операцию, которая ссылается на ClassY (использует экземпляр класса в качестве параметра, переменной или возвращаемого значения)
- ClassX – непосредственный потомок ClassY
- ClassY – интерфейс, а ClassX его реализует



# Слабое связывание(Low Coupling)

## Применение

- Применяется вместе с другими паттернами (такими как *Высокое зацепление* и *Эксперт*)
- Применяется в случае «нестабильности» в эволюционном смысле слова. **Не** применяется в отношении к классам статических библиотек.
- Применяется в разумных пределах

# Слабое связывание(Low Coupling)

## Результаты

### 1.Решение проблемы

1. Классы не зависят от изменений в других классах
2. Классы легко понимаемы вне контекста
3. Классы удобны для повторного использования

# Высокое сцепление (High cohesion)

## Проблема

Как управлять необъятным?

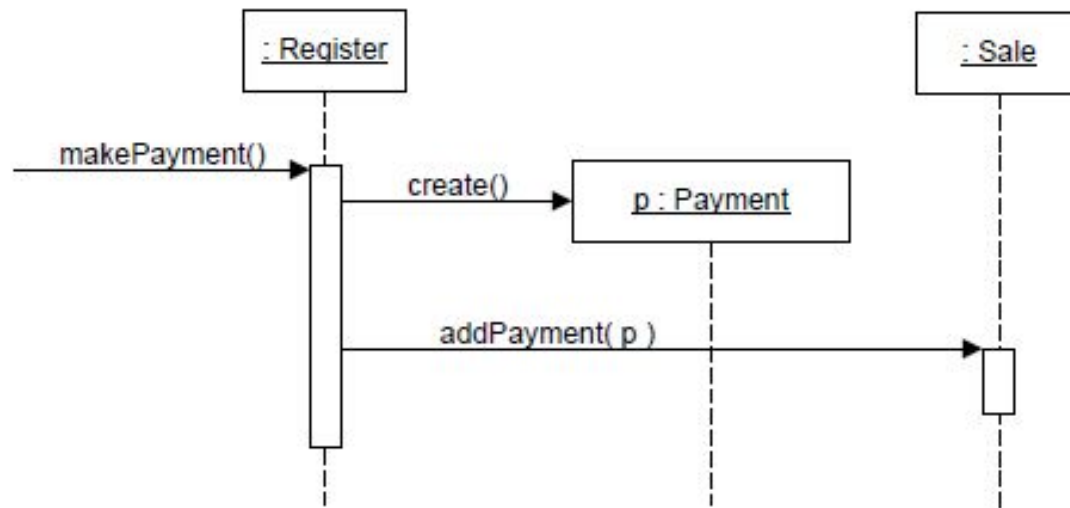
**Сцепление** – мера того, насколько взаимосвязаны и целенаправленны обязанности, возложенные на элемент. Элемент с взаимосвязанными обязанностями обладает *высоким сцеплением*.

## **Слабое сцепление приводит к проблемам:**

- Класс трудно «познать»
- Класс трудно использовать повторно
- Класс трудно поддерживать
- Класс «нежен»

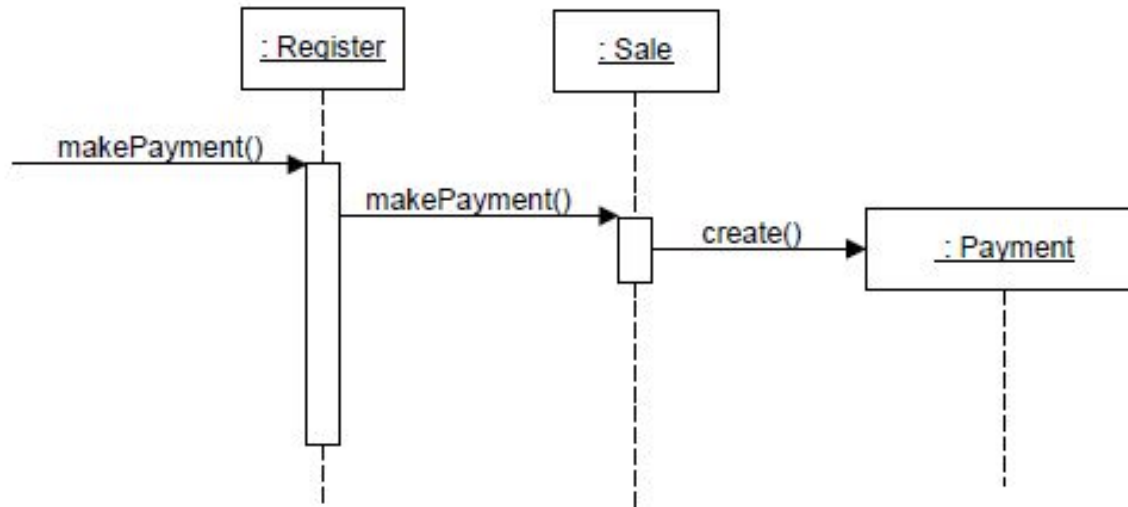
# Высокое зацепление (High cohesion)

Мы снова доверяемся паттерну «Создатель»



# Высокое сцепление (High cohesion)

Но тогда Register может разбухнуть!  
Попробуем все же вот так:



# Высокое зацепление (High cohesion)

Классификация зацеплений (by Grady Booch)

- Очень низкое зацепление (*RDB-RPC-Interface*)
- Низкое зацепление (*RDB-Interface*)
- Высокое зацепление
- Умеренное зацепление (*Company*)

# Высокое зацепление (High cohesion)

Зацепление и связывание  
Инь и Янь программной инженерии



## Исключения

- Упрощение обслуживания одним человеком (*ICQ Expert*)
- Удаленные операции

# Высокое зацепление (High cohesion)

## Результаты

- Ясность и простота понимания дизайна
- Упрощение поддержки и эволюции
- Слабое связывание в подарок
- Повторное использование

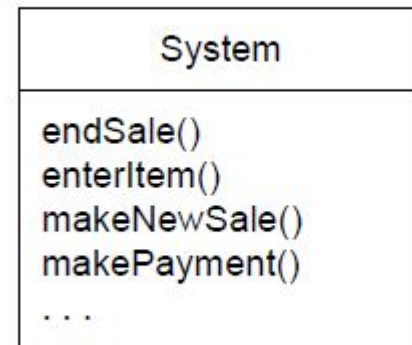


# Контроллер

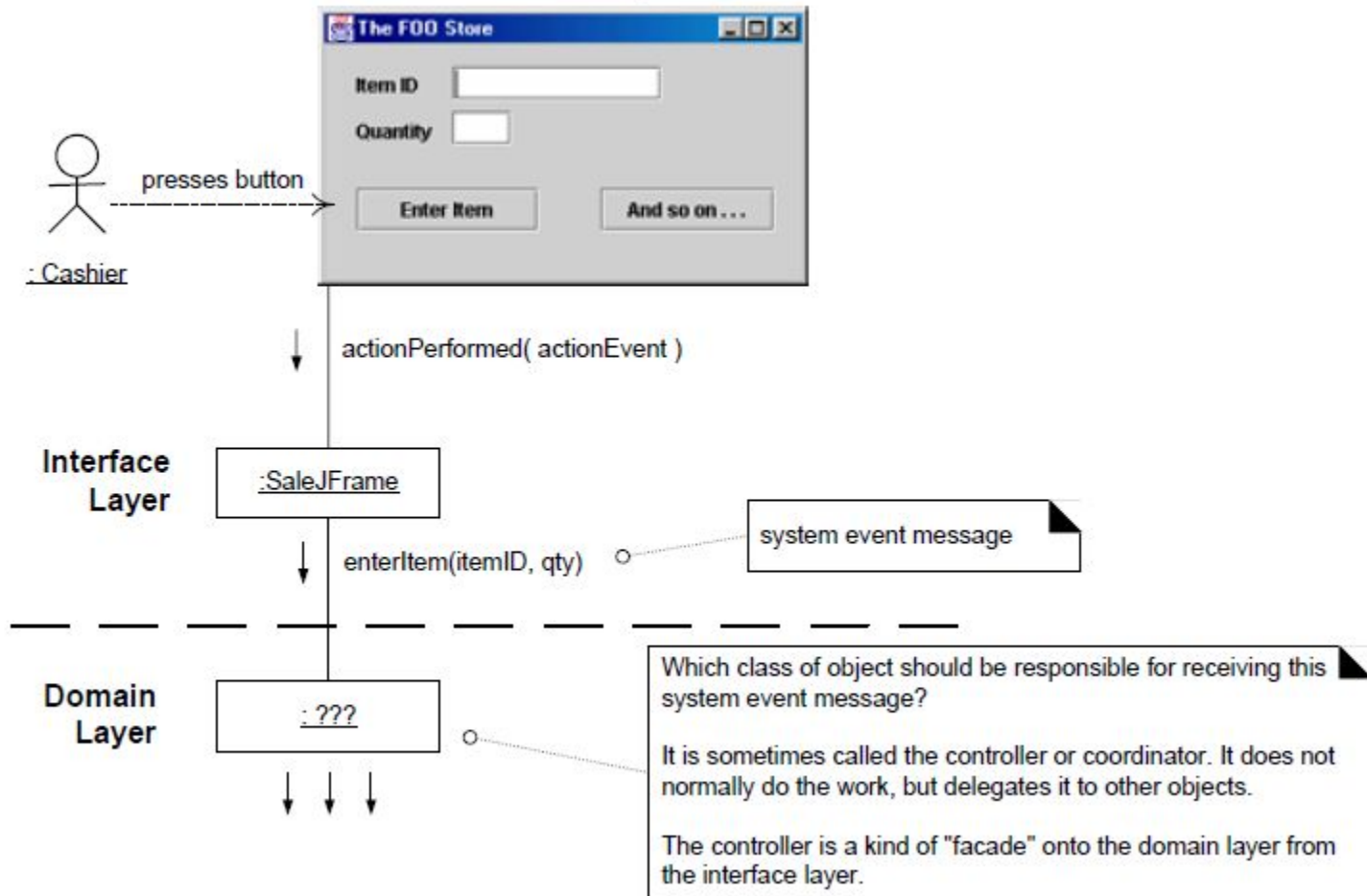
## Проблема

Кто должен быть ответственен за обработку системных событий ввода?

**Системное сообщение ввода**– сообщение, сгенерированное внешним актером. Они ассоциированы с **операциями** с событием.



# Контроллер



# Контроллер

## Варианты контроллера

- **Фасадный контроллер** - представляет всю систему(или подсистему) в совокупности
  - *Register*
  - Название, соответствующее системе (*ChessGame*)
- **Контроллер варианта использования** представляет ВИ, в пределах которого возникают сообщения.
  - `<UseCaseName>Handler`

# Контроллер

## Выбор контроллера

### •Фасадный контроллер

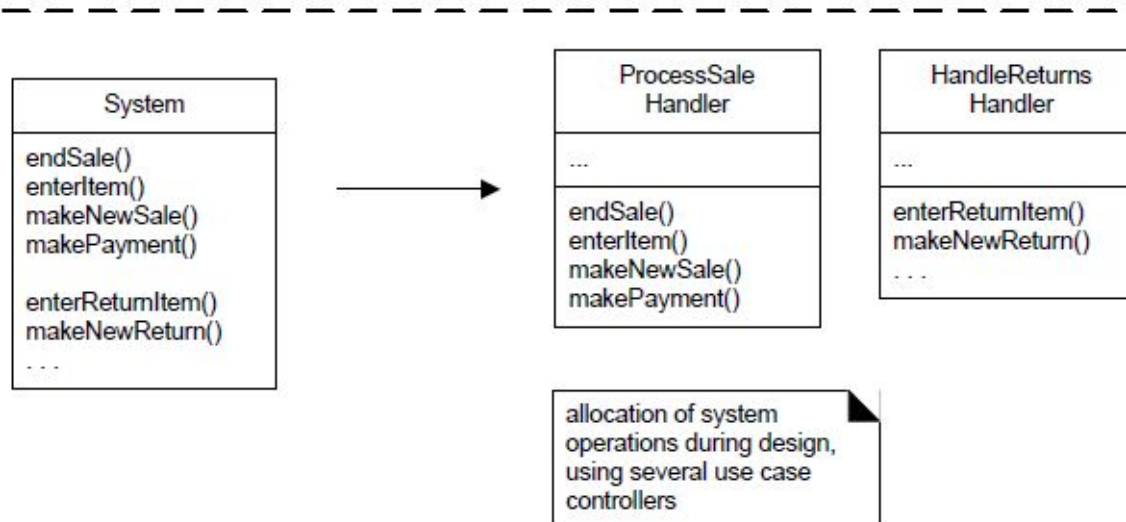
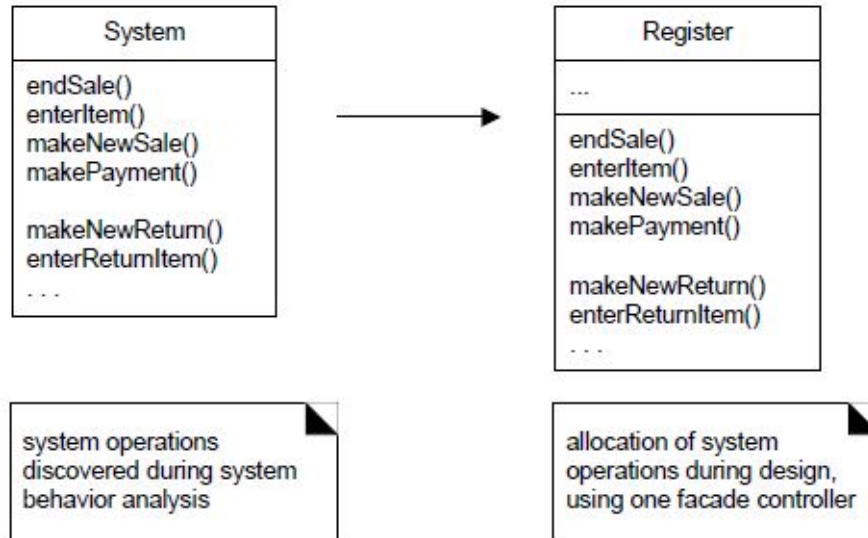
- Малое количество событий
- UI не может перенаправить системные сообщения на переменные контроллеры (как в системах обработки сообщений)

### •Контроллер ВИ

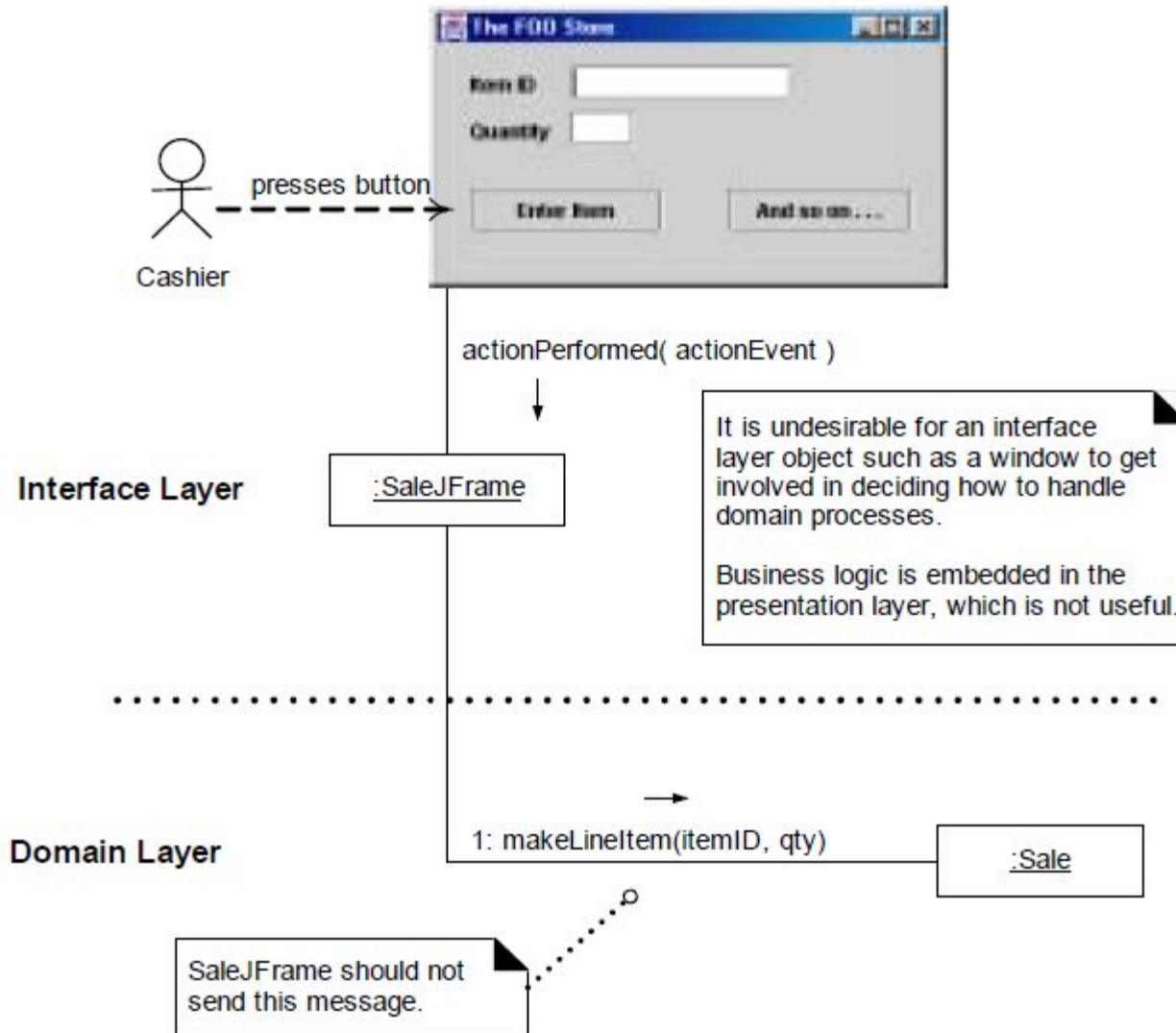
- Большое количество событий
- Необходимо поддерживать логику ВИ

UI не должны быть ответственны за события ввода!

# Контроллер



# Не контроллер



# Контроллер

## Результаты

- Логика приложения отделена от интерфейса
- Повторное использование
- Учет состояния варианта использования

# Контроллер

## Проблемы

- Один единственный контроллер, принимающий орды сообщений
- Контроллер берет на себя часть обязанностей по обработке сообщений
- Контроллер содержит много атрибутов
- Контроллер хранит дубликаты информации из системы

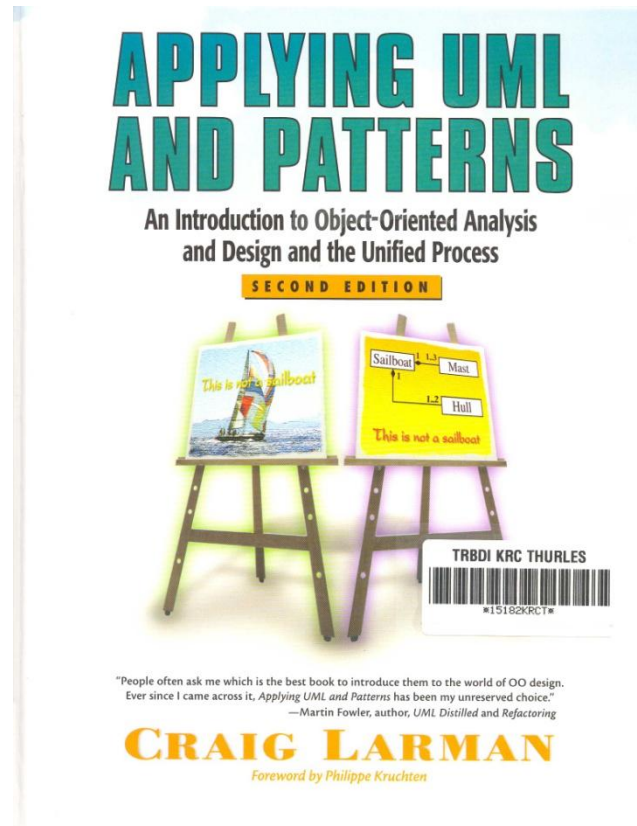
## Решения

- Увеличить количество контроллеров
- Проектировать контроллер чтобы он только делегировал обязанности другим объектам



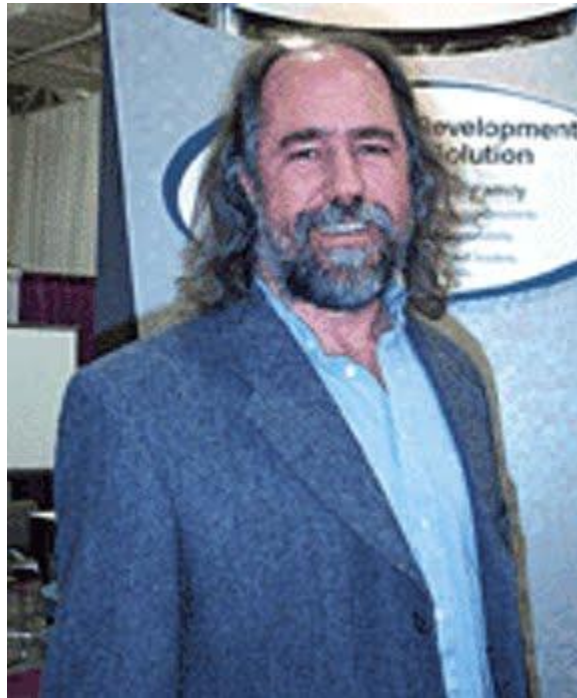
# Благодарность

- Creig Larman, за его интереснейшую книгу “Applying UML and patterns”.



# Благодарность

- Гради Бучу, за то, что учил Крэга Лармана. И за то, что научил.



# Благодарность

- А.Ю. Шелестову, за перевод замечательной книги Крэга Лармана на русский язык («Применение UML и шаблонов проектирования

## ПРИМЕНЕНИЕ UML И ШАБЛОНОВ ПРОЕКТИРОВАНИЯ

Введение в объектно-ориентированный анализ,  
проектирование и унифицированный процесс UP

ВТОРОЕ ИЗДАНИЕ



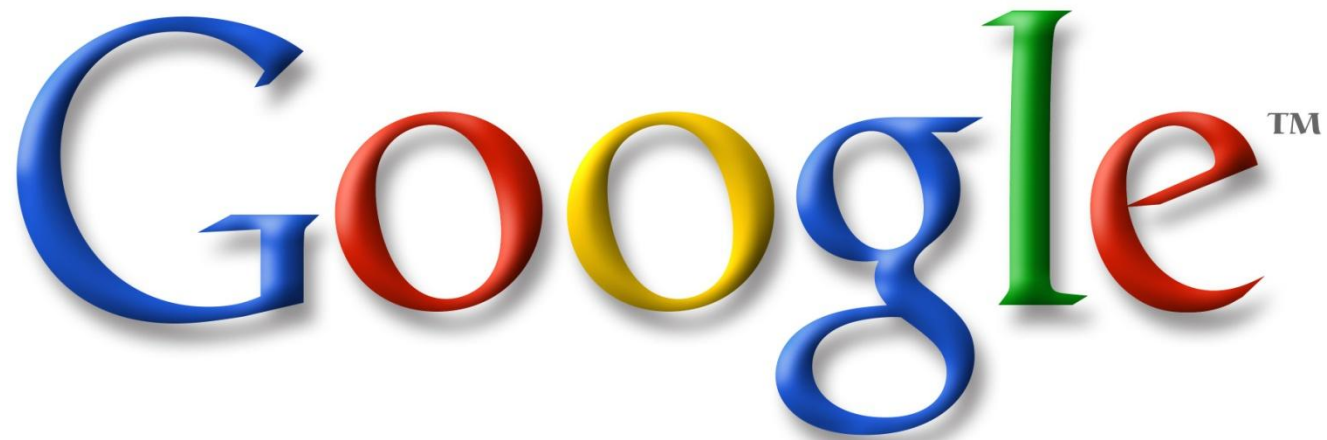
\*Люди часто спрашивают меня о том, с помощью какой книги лучше всего познакомиться с миром  
С тех пор, как я увидел книгу Применение UML и шаблонов проектирования, я постоянно возвращаюсь к ней.  
— Мартин Фаулер, автор книг UML, Design и Refactoring

**Крэг Ларман**

*Переводчик: Филиппа Кривитова*

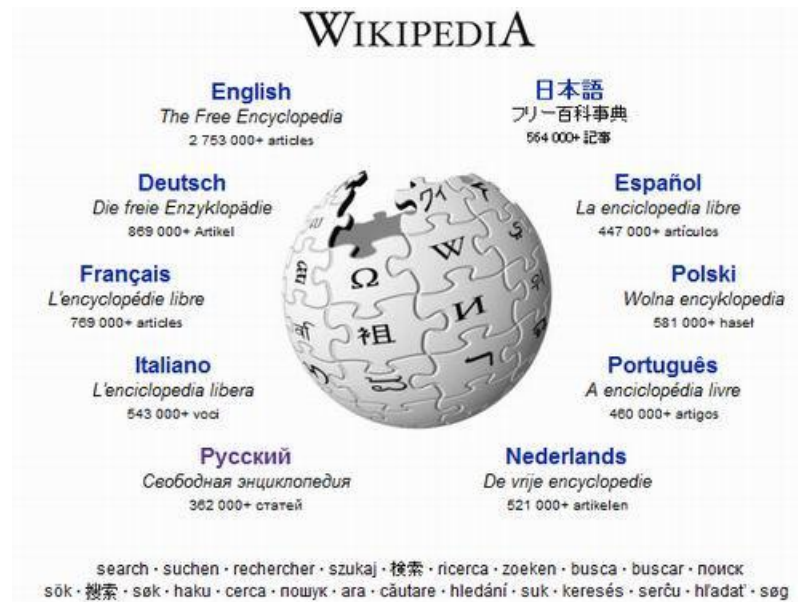
# Благодарность

- Google Search, за его релевантную выдачу.

The image shows the Google logo in its classic multi-colored font. The letters are 'G' (blue), 'o' (red), 'o' (yellow), 'g' (blue), 'l' (green), and 'e' (red). A small 'TM' trademark symbol is located to the right of the 'e'. The logo has a slight 3D effect with shadows.

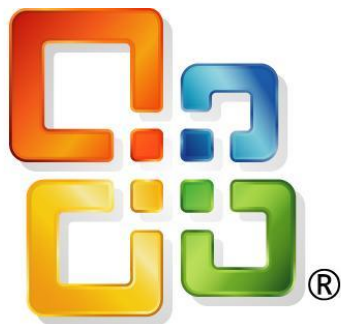
# Благодарность

- Ru.wikipedia.org, за понимание темы в целом.



# Благодарность

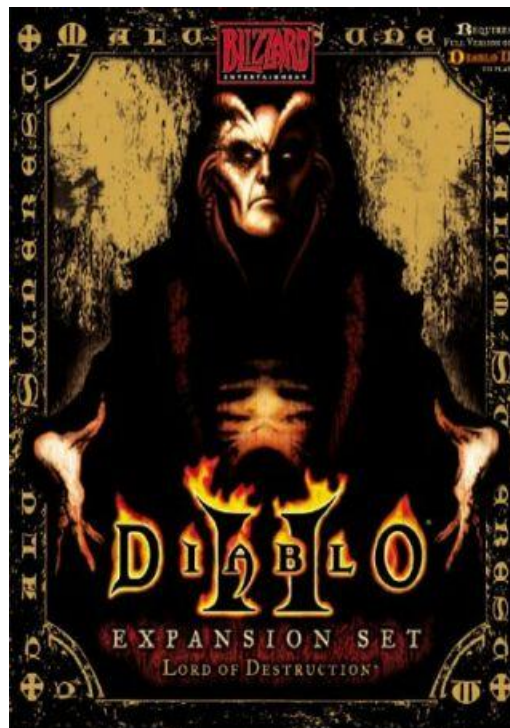
- Пакету Microsoft Office 2007, за прекрасные инструменты создания презентаций



Office Microsoft®

# Благодарность

- И, конечно, отдельное спасибо компании Blizzard, за ее бессмертные игры.



**Спасибо за внимание**