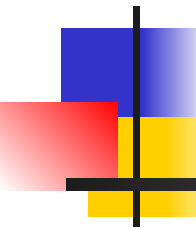
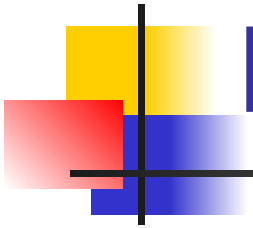


Design Patterns

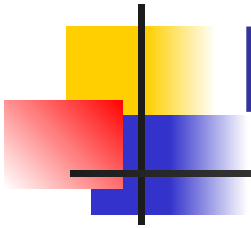


Программа курса

- История создания
- Что такое шаблоны проектирования?
- Шаблоны GRASP
- Шаблоны GoF
- Рефакторинг
- Анти паттерны

История создания

- В 70-х годах двадцатого века архитектор Кристофер Александр (Christopher Alexander) составил набор шаблонов проектирования. В области архитектуры эта идея не получила такого развития, как позже в области программной разработки.



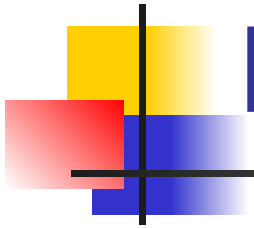
История создания

- В 1987 году Кент Бэк (Kent Beck) и Вард Каннигем (Ward Cunningham) взяли идеи Кристофер Александра и разработали шаблоны применительно к разработке программного обеспечения для разработки графических оболочек на языке Smalltalk.

История создания

- В 1988 году Эрих Гамма (Erich Gamma) начал писать докторскую работу при цюрихском университете об общей переносимости этой методики на разработку программ.

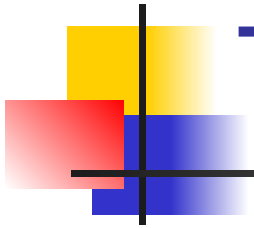
История создания



- В 1989—1991 годах Джеймс Коплин (James Coplien) трудился над разработкой идиом для программирования на C++ и опубликовал в 1991 году книгу Advanced C++ Idioms.

История создания

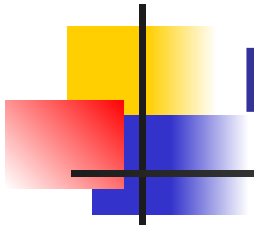
- В этом же году Эрих Гамма заканчивает свою докторскую работу и переезжает в США, где в сотрудничестве с Ричардом Хелмом (Richard Helm), Ральфом Джонсоном (Ralph Johnson) и Джоном Влиссидсом (John Vlissides) публикует книгу Design Patterns — Elements of Reusable Object-Oriented Software.
- В этой книге описаны 23 шаблона проектирования. Также команда авторов этой книги известна общественности под названием Банда четырёх (англ. Gang of Four, часто сокращается до GoF). Именно эта книга стала причиной роста популярности шаблонов проектирования.



Таксономия паттернов

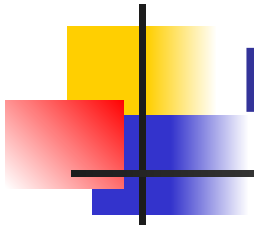
- Idiom
  - Напрямую связана с языком программирования
- Specific design
  - Решение частной задачи
- Standard design
  - Дополнительный уровень абстракции
- Design pattern
  - Объектно-ориентированные шаблоны – отношения, взаимодействие и распределение ответственности между классами или объектами для всего класса задач

Что такое шаблоны проектирования?

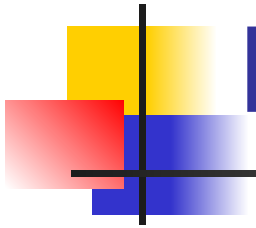


- "Каждый паттерн описывает некую повторяющуюся проблему и ключ к ее разгадке, причем таким образом, что этим ключом можно пользоваться при решении самых разнообразных задач". *Christopher Alexander*

Что такое шаблоны проектирования?

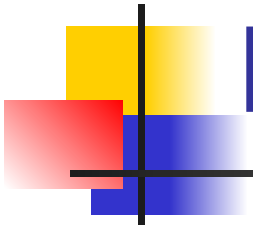


- Шаблоны проектирования (паттерн, pattern) — это эффективные способы решения характерных задач проектирования, в частности проектирования компьютерных программ. Паттерн не является законченным образцом проекта, который может быть прямо преобразован в код, скорее это описание или образец для того, как решить задачу, таким образом чтобы это можно было использовать в различных ситуациях.



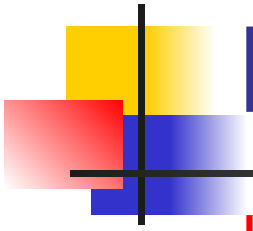
Польза

- Описывает решение целого класса абстрактных проблем
- Унификация терминологии, названий модулей и элементов проекта
- Позволяют, отыскав удачное решение, пользоваться им снова и снова
- В отличие от идиом, шаблоны независимы от применяемого языка программирования



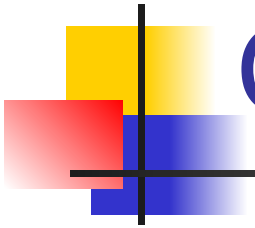
Недостатки

- шаблоны могут консервировать громоздкую и малоэффективную систему понятий, разработанную узкой группой
- Когда количество шаблонов возрастает, превышая критическую сложность, исполнители начинают игнорировать шаблоны и всю систему, с ними связанную
- Есть мнение, что слепое применение шаблонов из справочника, замедляет профессиональный рост программиста, так как подменяет творческую работу механическим подставлением шаблонов.



Итоги

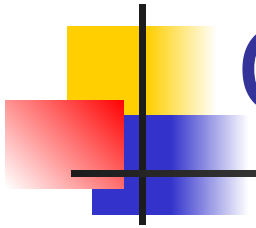
- Шаблоны проектирования (паттерн, pattern) — это эффективные способы решения характерных задач проектирования;
- Шаблоны - не являются законченным образцом проекта, они лишь способ решения, «повод подумать»;
- Шаблоны - не панацея, но дают возможность сильно повысить свой уровень разработчика, использовать лучший опыт;
- Шаблоны – ступенька к становлению Computer Science как науки, а не ремесленечества.



GRASP

- Craig Larman
- Книга Applying UML and Patterns, 3ed. GRASP stands for **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns.

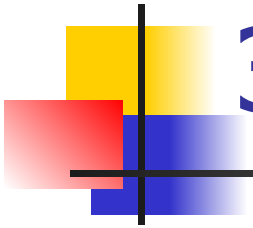
Полный список шаблонов GRASP



- Information Expert
- Creator
- Controller
- Low Coupling
- High Cohesion
- Polymorphism
- Pure Fabrication
- Indirection
- Protected Variations

Шаблон информационный эксперт (Information Expert)- GRASP

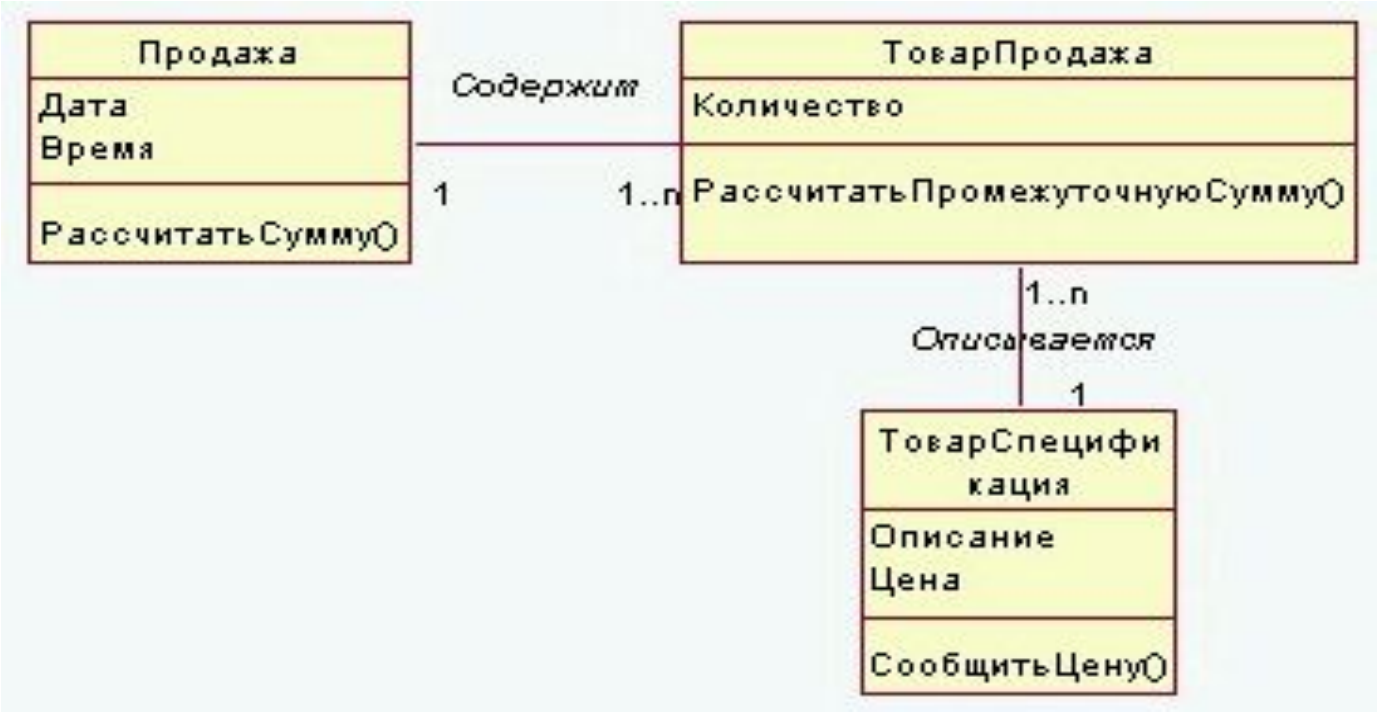
- Проблема
  - В системе должна аккумулироваться, рассчитываться и т. п. необходимая информация.
- Решение
  - Назначить обязанность информационному эксперту - классу, у которого имеется информация, требуемая для выполнения обязанности.
- Рекомендации
  - Информационным экспертом может быть не один класс, а несколько.



Эксперт. Пример

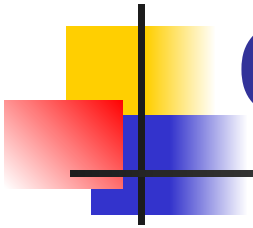
- Необходимо рассчитать общую сумму продажи. Имеются классы проектирования "Продажа", "ТоварПродажа" (продажа отдельного вида товара в рамках продажи в целом), "ТоварСпецификация" (описание конкретного вида товара).
- Необходимо распределить обязанности по предоставлению информации и расчету между этими классами.
- Таким образом, все три объекта являются информационными экспертами.

Эксперт. Диаграмма классов



Создатель экземпляров класса (Creator) - GRASP

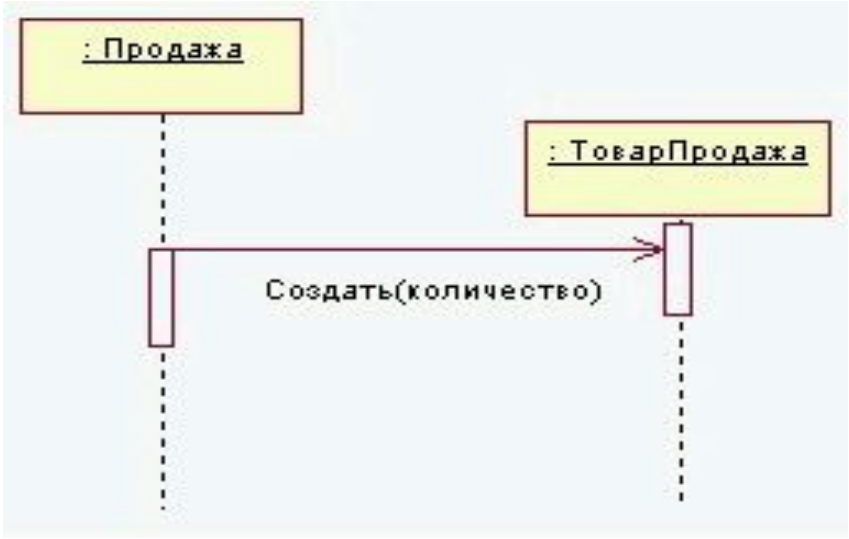
- Проблема
  - "Кто" должен отвечать за создание экземпляров класса.
- Решение
  - Назначить классу B обязанность создавать объекты другого класса A
- Рекомендации
  - Логично использовать паттерн если класс B содержит, агрегирует, активно использует и т. п. объекты класса A.

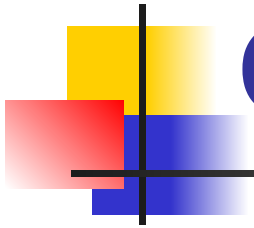


Creator. Пример.

- необходимо определить, какой объект должен отвечать за создание экземпляра "ТоварПродажа".
- Логично, чтобы это был объект "Продажа", поскольку он содержит (агрегирует) несколько обьектов "ТоварПродажа".

Creator. Диаграмма последовательности

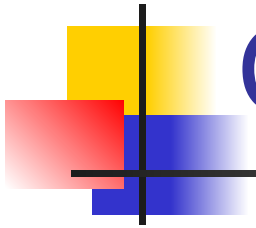




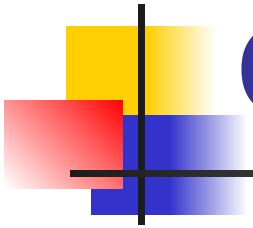
Creator. Критика

- Преимущества
  - Использование этого паттерна не повышает связанности, поскольку созданный класс, как правило, виден только для класса - создателя.
- Недостатки
  - Если процедура создания объекта достаточно сложная (например выполняется на основе некоего внешнего условия), логично использовать паттерн "Абстрактная Фабрика",, то есть, делегировать обязанность создания объектов специальному классу.

Контроллер (Controller) - GRASP



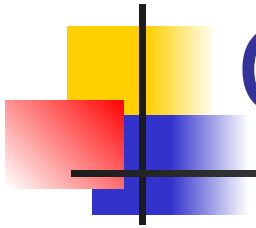
- Проблема
  - "Кто" должен отвечать за обработку входных системных событий?
- Решение
  - Обязанности по обработке системных сообщений делегируются специальному классу. Контроллер - это объект, который отвечает за обработку системных событий и не относится к интерфейсу пользователя. Контроллер определяет методы для выполнения системных операций.
- Рекомендации
  - Для различных прецедентов логично использовать разные контроллеры (контроллеры прецедентов) - контроллеры не должны быть перегружены. Внешний контроллер представляет всю систему целиком, его можно использовать, если он будет не слишком перегруженным (то есть, если существует лишь несколько системных событий).



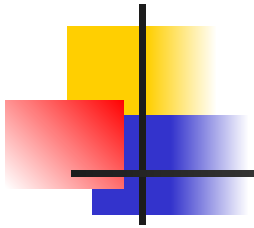
Controller. Критика

- Преимущества
  - Удобно накапливать информацию о системных событиях (в случае, если системные операции выполняются в некоторой определенной последовательности).
  - Улучшаются условия для повторного использования компонентов (системные события обрабатываются Контроллером а не элементами интерфейса пользователя).
- Недостатки
  - Контроллер может оказаться перегружен.

Низкая связанность (Low Coupling)

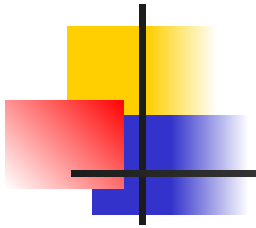


- Проблема
  - Обеспечить низкую связанность при создании экземпляра класса и связывании его с другим классом.
- Решение
  - Распределить обязанности между объектами так, чтобы степень связанности оставалась низкой.



Степень связанности (coupling) - это мера, определяющая насколько жестко один элемент связан с другими элементами, либо каким количеством данных о других элементах он обладает.

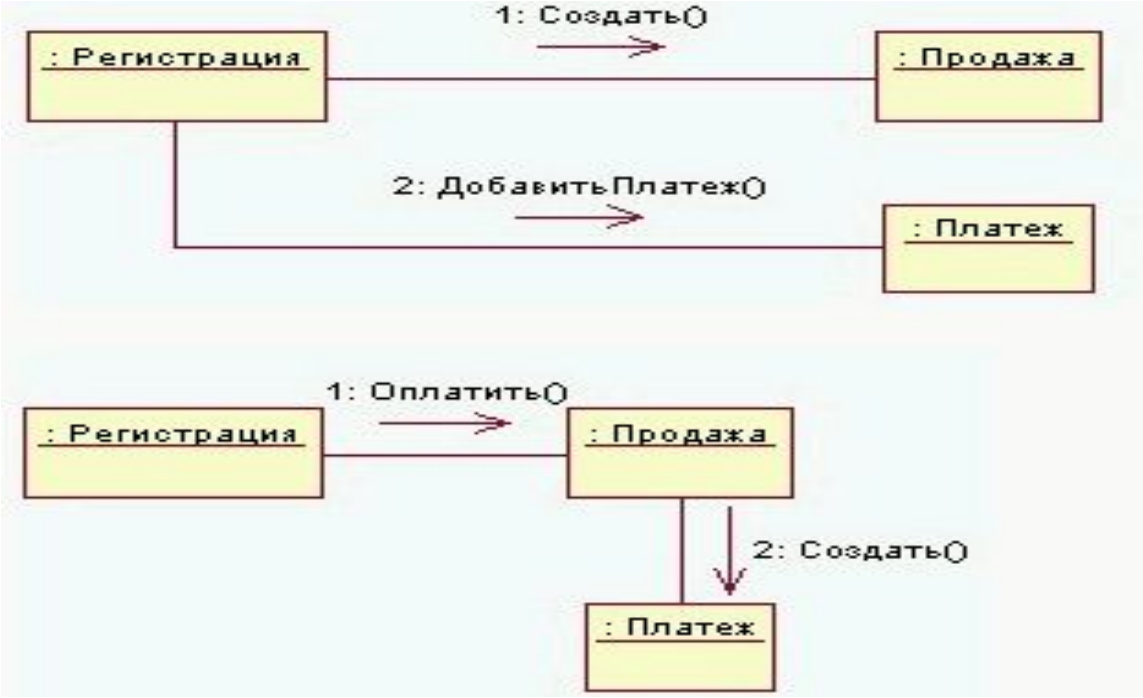
Элемент с низкой степенью связанности (или слабым связыванием) зависит от не очень большого числа других элементов.



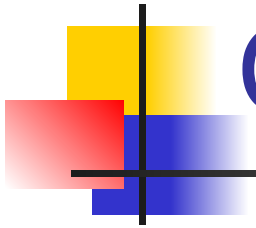
Класс с высокой степенью связанности (или жестко связанный) зависит от множества других классов. Однако наличие таких классов нежелательно, поскольку оно приводит к возникновению следующих проблем.

- Изменение в связанных классах приводят к локальным изменениям в данном классе.
- Затрудняется понимание каждого класса в отдельности.
- Усложняется повторное использование, поскольку для этого требуется дополнительный анализ классов, с которыми связан данный класс.

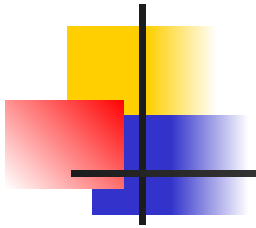
Низкая связанность. Пример



Высокое зацепление (High Cohesion) - GRASP

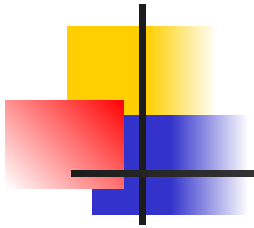


- Проблема
  - Необходимо обеспечить выполнение объектами разнородных функций.(Как обеспечить возможность управления сложностью?)
- Решение
  - Обеспечить распределение обязанностей с высоким зацеплением.



Зацепление (cohesion) (или более точно, функциональное зацепление) - это мера связанности и сфокусированности обязанностей класса.

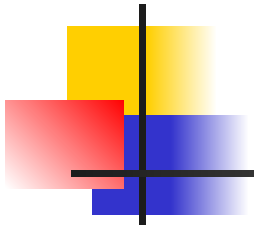
Считается что элемент обладает высокой степенью зацепления, если его обязанности тесно связаны между собой и он не выполняет огромных объемов работы. В качестве элемента может выступать класс, подсистема ...



Класс с низкой степенью зацепления выполняет много разнородных функций или несвязанных между собой обязанностей. Такие классы создавать нежелательно, поскольку они приводят к возникновению следующих проблем.

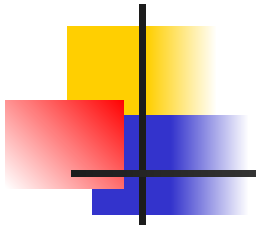
- Трудность понимания.
- Сложность при повторном использовании.
- Сложность поддержки.
- Надежность, постоянная подверженность изменениям.

Классы со слабым зацеплением, как правило, являются слишком "абстрактными" или выполняют обязанности, которые можно легко распределить между другими объектами.



Как правило, класс с высокой степенью зацепления содержит сравнительно небольшое количество методов, которые функционально тесно связаны между собой, и не выполняют слишком много функций. Они взаимодействуют с другими объектами для выполнения более сложных задач. Такие классы являются предпочтительными, поскольку просты в понимании, поддержке и повторном использовании.

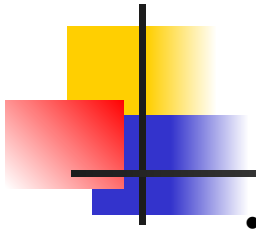
Аналогия - менеджер, который не умеет распределять обязанности между своими подчиненными. Человек страдает от "низкой степени зацепления" и может легко "расклеиться"



Несколько сценариев, иллюстрирующих различную степень функционального зацепления.

- Очень слабое зацепление. Только один класс отвечает за выполнение множества операций в самых различных функциональных областях. Допустим, существует класс RDB-RPC-Interface, полностью отвечающий за взаимодействие между реляционными базами данных и вызов удаленных процедур. Это две абсолютно разные функциональные области.

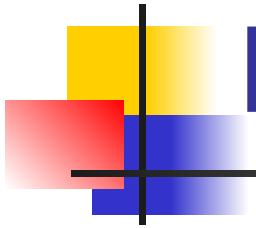
- Слабое зацепление. Класс несет единоличную ответственность за выполнение сложной задачи из одной функциональной области. Допустим, некий класс RDBInterface полностью отвечает за взаимодействие с рел. базами данных. Методы класса связаны между собой однако их слишком много, и их реализация требует огромных объемов кодов. Таких методов может быть несколько сотен или больше. Данный класс следует разделить на несколько более мелких классов.



- Сильное зацепление. Класс имеет среднее количество обязанностей из одной функциональной области и для выполнения своих задач взаимодействует с другими классами. Допустим некоторый класс RDBInterface лишь частично отвечает за взаимодействие с рел. базами данных. Для извлечения и хранения объектов в БД он взаимодействует с с десятком других классов.

- Среднее зацепление. Класс имеет несложные обязанности в нескольких различных областях, логически связанных с концепцией этого класса, но не связанных между собой. Допустим существует класс Company, который несет полную ответственность за (а) знание всех сотрудников компании и (б) всю финансовую информацию. Эти две области не слишком связаны между собой, однако логически связаны понятием "компания". К тому же предполагается, что такой класс содержит небольшое количество открытых методов и требует незначительных объемов кода для их реализации.

Высокое зацепление. Критика

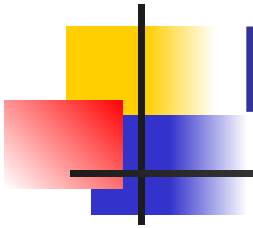


- Преимущества
  - Классы с высокой степенью зацепления просты в поддержке и повторном использовании.
- Недостатки
  - Иногда бывает неоправданно использовать высокое зацепление для распределенных серверных обьектов. В этом случае для обеспечения быстродействия необходимо создать несколько более крупных серверных обьектов со слабым зацеплением.

Полиморфизм (Polymorphism) - GRASP

- Проблема
  - Как обрабатывать альтернативные варианты поведения на основе типа?
  - Как заменять подключаемые компоненты системы?
- Решение
  - Обязанности распределяются для различных вариантов поведения с помощью полиморфных операций для этого класса.
  - Каждая внешняя система имеет свой интерфейс.

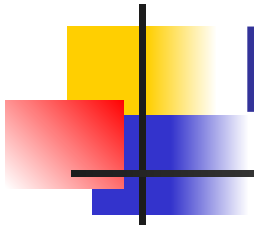
Полиморфизм. Пример



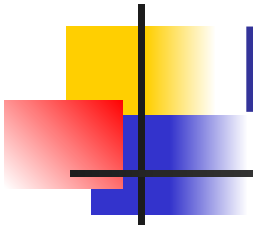
Полиморфизм. Критика

- Преимущества
  - Впоследствии легко расширять и модернизировать систему.
- Недостатки
  - Не следует злоупотреблять добавлением интерфейсов с применением принципа полиморфизма с целью обеспечения дееспособности системы в неизвестных заранее новых ситуациях.

Искусственный (Pure Fabrication) - GRASP



- Проблема
  - Какой класс должен обеспечивать реализацию паттернов "Высокое зацепление", и "Низкая связанность"?
- Решение
  - Присвоить группу обязанностей с высокой степенью зацепления классу, который не представляет конкретного понятия из предметной области (синтезировать искусственную сущность для обеспечения высокого зацепления и слабого связывания).

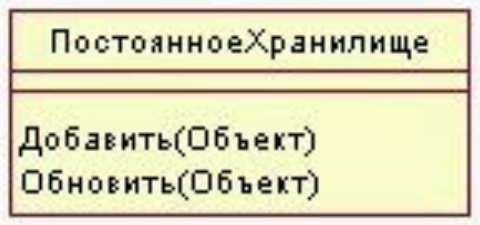


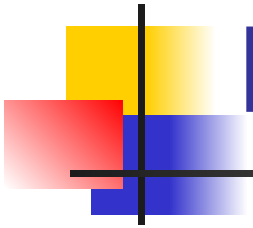
Искусственный. Пример

- Какой класс должен сохранять экземпляры класса "Продажа" в реляционной базе данных?
- Если возложить эту обязанность на класс "Продажа", то будем иметь низкую степень зацепления и высокую степень связывания (поскольку класс "Продажа" должен быть связан с интерфейсом реляционной базы данных.
- Хранение объектов в реляционной базе данных - это общая задача, которую придется решать для многих классов.

Искусственный. Пример

- Решением данной проблемы будет создание нового класса "ПостоянноеХранилище", ответственного за сохранение обьектов некоторого вида в базе данных.

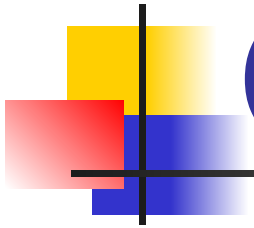




Искуственный. Критика

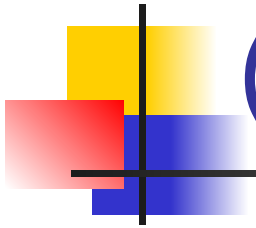
- Преимущества
  - Класс "ПостоянноеХранилище" будет обладать низкой степенью связывания и высокой степенью зацепления.
- Недостатки
  - Данным паттерном не следует злоупотреблять иначе все функции системы превратятся в объекты.

Перенаправление (Indirection) - GRASP



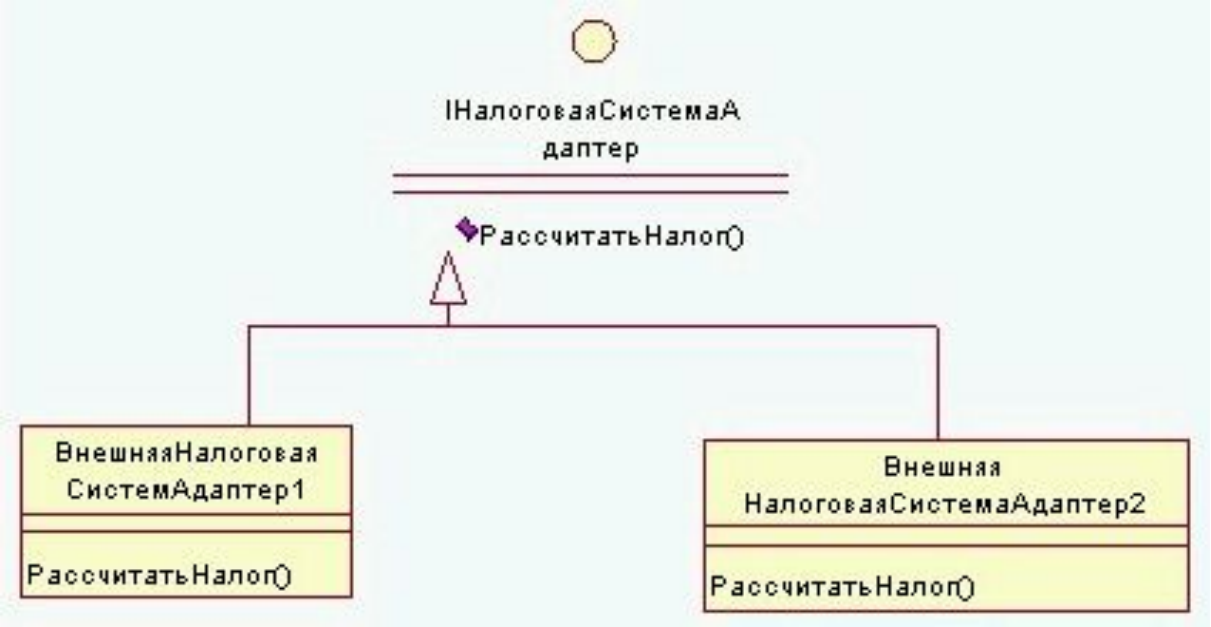
- Проблема
  - Как перераспределить обязанности обьектов, чтобы обеспечить отсутствие прямого связывания?
- Решение
  - Присвоить обязанности по обеспечению связи между службами или компонентами промежуточному объекту.
- Пример
  - См. пример к паттерну "Искусственный". Класс "Хранилище" выступает в роли промежуточного звена между классом "Продажа" и базой данных.

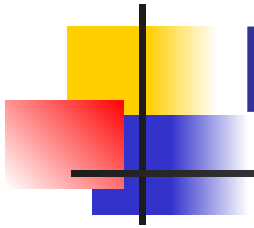
Устойчивый к изменениям (Protected Variations) - GRASP



- Проблема
  - Как спроектировать систему так, чтобы изменение одних ее элементов не влияло на другие?
- Решение
  - Идентифицировать точки возможных изменений или неустойчивости и распределить обязанности таким образом, чтобы обеспечить устойчивую работу системы.

Устойчивый к изменениям. Пример





Итоги

- Паттерны GRASP:
  - Information Expert
  - Creator
  - Controller
  - Low Coupling
  - High Cohesion
  - Polymorphism
  - Pure Fabrication
  - Indirection
  - Protected Variations