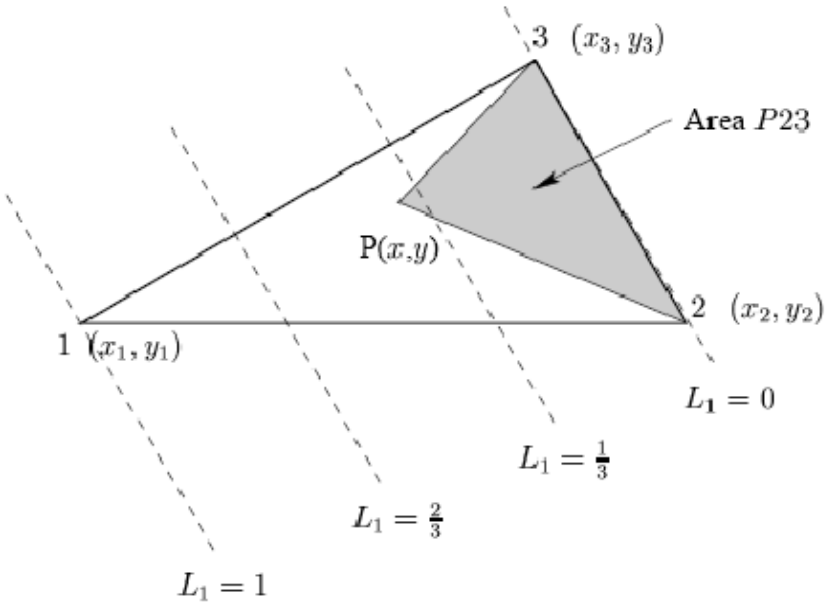
Билет 12.

1. Типы двумерных конечных элементов и их базисные функции.

1. Треугольные элементы.



$$L_1 = \frac{\text{Area} < P23 >}{\text{Area} < 123 >} = \frac{1}{2}\begin{vmatrix} 1 & x & y \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} / \Delta = (a_1 + b_1 x + c_1 y) / (2\Delta)$$

$$\Delta = \frac{1}{2}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}$$

$$L_2 = \frac{\text{Area} < P13 >}{\text{Area} < 123 >} = \frac{1}{2}\begin{vmatrix} 1 & x & y \\ 1 & x_3 & y_3 \\ 1 & x_1 & y_1 \end{vmatrix} / \Delta = (a_2 + b_2 x + c_2 y) / (2\Delta)$$

$$L_3 = \frac{\text{Area} < P12 >}{\text{Area} < 123 >} = \frac{1}{2}\begin{vmatrix} 1 & x & y \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{vmatrix} / \Delta = (a_3 + b_3 x + c_3 y) / (2\Delta)$$

$\mathbf{a_1 = x_2 y_3 - x_3 y_2}$ $\mathbf{b_1 = y_2 - y_3}$ $\mathbf{c_1 = x_3 - x_2}$

$$a_2 = x_3 y_1 - x_1 y_3, b_2 = y_3 - y_1, c_2 = x_1 - x_3$$
$$a_3 = x_1 y_2 - x_2 y_1, b_3 = y_1 - y_2, c_3 = x_2 - x_1.$$

$$L_1 + L_2 + L_3 = 1.$$

$$u(x,y) = \varphi_1(x,y)\,u_1 + \varphi_2(x,y)\,u_2 + \varphi_3(x,y)\,u_3$$

$$\varphi_1 = L_1, \varphi_2 = L_2$$
$$\varphi_3 = L_3 = 1 - L_1 - L_2.$$

2. Квадратичные элементы.

$$u(\xi_1,\xi_2) = \varphi_1(\xi_1,\xi_2)\,u_1 + \varphi_2(\xi_1,\xi_2)\,u_2 + \varphi_3(\xi_1,\xi_2)\,u_3 + \varphi_4(\xi_1,\xi_2)\,u_4$$

$\phi_1(x,y)=(1-\xi_1)(1-\xi_2)$
$\phi_2(x,y)=\xi_1(1-\xi_2)$
$\phi_3(x,y)=(1-\xi_1)\xi_2$
$\phi_4(x,y)=\xi_1\xi_2$
Интегральное уравнение:

$$\int_\Omega k\left(\frac{\partial u}{\partial x}\frac{\partial \omega}{\partial x}+\frac{\partial u}{\partial y}\frac{\partial \omega}{\partial y}\right)d\Omega=\int_\Gamma k\frac{\partial u}{\partial n}\omega d\Gamma$$

$$\sum_i \int_\Omega k\left(\frac{\partial u}{\partial x}\frac{\partial \omega}{\partial x}+\frac{\partial u}{\partial y}\frac{\partial \omega}{\partial y}\right)d\Omega=\int_\Gamma k\frac{\partial u}{\partial n}\omega d\Gamma$$

$E_{mn}u_n=F_m$

E_{mn} - матрица жесткости, а F_m - вектор нагрузки.

Таки м образом строится матрица жесткости. Для треугольных элементов [3x3], для квадратичных - [4x4].

2. GRASP: шаблоны для распределения обязанностей.

Аббревиатура GRASP означает General Responsibility Assignment Software Patterns (Общие шаблоны распределения обязанностей в программных системах). Такая аббревиатура очень символична, поскольку подчеркивает важность изучения этих принципов для успешного проектирования объектно-ориентированного приложения.

Сейчас рассмотрим первые пять шаблонов GRASP.

- Expert
- Creator
- High Cohesion
- Low Coupling
- Controller

Эти шаблоны касаются основных и фундаментальных вопросов проектирования.

Шаблон Expert

Решение. Назначить обязанность информационному эксперту — классу, у которого имеется информация, требуемая для выполнения обязанности.
Проблема. Каков наиболее общий принцип распределения обязанностей между объектами при объектно-ориентированном проектировании?

В модели системы могут быть определены десятки или сотни программных классов, а в приложении может потребоваться выполнение сотен или тысяч обязанностей. Во время объектно-ориентированного проектирования при формулировке принципов

взаимодействия объектов необходимо распределить обязанности между классами, При правильном выполнении этой задачи система становится гораздо проще для понимания, поддержки и расширения. Кроме того, появляется возможность повторного использования уже разработанных компонентов в последующих приложениях.

Преимущества
• Шаблон Expert поддерживает инкапсуляцию. Для выполнения требуемых задач объекты используют собственные данные. Подобную возможность обеспечивает также шаблон Low Coupling, применение которого приводит к созданию более надежных и приспособленных к обслуживанию систем. (Low Coupling является шаблоном GRASP, который будет рассмотрен чуть ниже.)
• Нужное поведение системы обеспечивается несколькими классами, содержащими требуемую информацию. Это приводит к определениям классов, которые гораздо проще понимать и поддерживать. Кроме того, поддерживается шаблон High Cohesion (рассматриваемый ниже).
Другие названия и аналогичные принципы. "Хранение обязанностей вместе с данными", "Кто знает, тот и выполняет", "Оживление", "Сделай сам", "Размещайте службы вместе с их атрибутами".

Шаблон Creator

Решение. Назначить классу B обязанность создавать экземпляры класса A, если выполняется одно из следующих условий

- Класс B агрегирует (aggregate) объекты A
- Класс B содержит (contains) объекты A
- Класс B записывает (records) экземпляры объектов A
- Класс B активно использует (closely uses) объекты A
- Класс B обладает данными инициализации (has the initializing data), которые будут передаваться объектам A при его создании (т.е. при создании объектов A класс B является экспертом)

Класс B — создатель (creator) объектов A.

Если выполняется несколько из этих условий, то лучше использовать класс B, агрегирующий или содержащий класс A.

Проблема. Кто должен отвечать за создание нового экземпляра некоторого класса? Создание объектов в объектно-ориентированной системе является одним из наиболее стандартных видов деятельности. Следовательно, при назначении обязанностей, связанных с созданием объектов, полезно руководствоваться некоторым основным принципом. Правильно распределив обязанности при проектировании, можно создать слабо связанные независимые компоненты с возможностью их дальнейшего использования, упростить их, а также обеспечить инкапсуляцию данных.

Обсуждение. Шаблон Creator определяет способ распределения обязанностей, связанный с процессом создания объектов. В объектно-ориентированных системах эта задача является одной из наиболее распространенных. Основным назначением шаблона Creator является выявление объекта-создателя, который при возникновении любого события должен быть связан со всеми созданными им объектами.

Целое агрегирует свои части, контейнер хранит свое содержимое, регистратор ведет учет. Все эти взаимосвязи являются очень распространенными способами взаимодействия классов на диаграмме классов. В шаблоне Creator определяется,

что внешний контейнер и класс-регистратор — это хорошие кандидаты на выполнение обязанностей, связанных с созданием сущностей, которые они будут содержать или регистрировать. Конечно, это утверждение является лишь рекомендацией.

Обратите внимание, что понятие "агрегация" (aggregation) используется в соответствии с его определением в шаблоне Creator. Агрегация подразумевает, что сущности находятся во взаимосвязи "целое — часть" или "агрегат — часть" подобно тому, как туловище агрегирует ноги или абзац содержит отдельные предложения.

В некоторых случаях в качестве создателя выбирается класс, который содержит данные инициализации, передаваемые объекту во время его создания. На самом деле это пример использования шаблона Expert. В процессе создания инициализирующие данные передаются с помощью метода инициализации некоторого вида, такого как конструктор языка Java с параметрами,

Например, предположим, что при создании экземпляра объекта Payment нужно выполнить инициализацию с использованием общей суммы, содержащейся в объекте Sale. Поскольку объекту Sale эта сумма известна, он является кандидатом на выполнение обязанности, связанной с созданием экземпляра объекта Payment.

Преимущества

• Поддерживается шаблон Low Coupling (рассматриваемый ниже), способствующий снижению затрат на сопровождение и обеспечивающий возможность использования созданных компонентов в дальнейшем. Применение шаблона Creator не повышает степень связанности, поскольку созданный (created) класс, как правило, оказывается видимым для класса-создателя посредством имеющихся ассоциаций.

Шаблон Low Coupling

Решение. Распределить обязанности таким образом, чтобы степень связанности оставалась низкой**.**

Проблема. Степень связанности (coupling) — это способ измерения того, насколько жестко один класс связан с другими классами либо каким количеством данных о других классах он обладает. Класс с низкой степенью связанности (или слабым связыванием) зависит от не очень большого числа других классов. Выражение "очень много" зависит от контекста, однако необходимо провести его оценку.

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.

• Изменения 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

Обсуждение: В шаблоне Low Coupling описывается принцип, о котором нельзя забывать на протяжении всех стадий работы над проектом. Он является предметом постоянного обсуждения. Шаблон Low Coupling представляет собой средство, которое разработчик применяет при оценке всех принимаемых в процессе проектирования решений.

В объектно-ориентированных языках программирования, таких как C++, Java и Smalltalk, имеются следующие стандартные способы связывания объектов TypeX и TypeY.

• Объект TypeX содержит атрибут (переменную-член), который ссылается на экземпляр объекта TypeУ или сам объект TypeX

и Объект TypeX содержит метод, который каким-либо образом ссылается на экземпляр объекта TypeY или сам объект TypeX (обычно это подразумевает использование TypeУ в качестве типа параметра, локальной переменной или возвращаемого значения)

• Объект TypeX является прямым или непрямым подклассом объекта TypeY
• Объект TypeY является интерфейсом, а TypeX реализует этот интерфейс Шаблон Low Coupling подразумевает , такое распределение обязанностей, которое не влечет за собой чрезмерное повышение степени связывания, приводящее к отрицательным результатам. Шаблон Low Coupling поддерживает независимость классов, что, в свою очередь, повышает возможности повторного использования и обеспечивает более высокую эффективность приложения. Его нельзя рассматривать изолированно от других шаблонов, таких как Expert и High Cohesion. Скорее, он обеспечивает один из основных принципов проектирования, применяемых при распределении обязанностей.
Если повторное использование кода не является основной целью при создании приложения, то степень связывания играет не столь существенную роль. Прежде чем применять принцип слабого связывания для улучшения возможностей повторного использования компонентов, следует задуматься о том, насколько это необходимо. Например, если нет полной уверенности в том, что данные компоненты придется когда-либо повторно использовать, не стоит тратить слишком много усилий на обеспечение их слабого связывания. Отсюда не следует, что обеспечение возможностей повторного использования кода является пустой тратой времени, просто затраты всегда должны быть оправданными.
Подкласс жестко связан со своим суперклассом. Поэтому, принимая решение о наследовании свойств объектов, следует учитывать
, что отношение наследования повышает степень связанности классов. Например, предположим, объекты необходимо постоянно хранить в реляционной или объектной базе данных. В этом случае зачастую создают абстрактный суперкласс PersistentObject, от которого наследуют свои свойства другие классы. Недостатком такого подхода является жесткое связывание объектов с конкретной службой, а преимуществом — автоматическое наследование поведения. Однако "брак по расчету" редко бывает удачным.
Не существует абсолютной меры для определения слишком высокой степени связывания. Важно лишь понимать степень связанности объектов на текущий момент и не упустить тот момент, когда дальнейшее повышение степени связанности может привести к возникновению проблем. В целом, следует руководствоваться таким принципом: классы, которые являются, достаточно общими по своей природе и с высокой вероятностью будут повторно использоваться в дальнейшем, должны иметь минимальную степень связанности с другими классами.
Крайним случаем при реализации шаблона Low Coupling является полное отсутствие связывания между классами. Такая ситуация тоже нежелательна, поскольку базовой идеей объектного подхода является система связанных объектов, которые общаются между собой посредством передачи сообщений. При слишком усердном использовании принципа слабого связывания система будет состоять из нескольких изолированных сложных активных объектов, самостоятельно выполняющих все операции, и множества пассивных объектов, основная функция которых сводится к хранению данных. Поэтому при создании объектно-ориентированной системы должна присутствовать некоторая оптимальная степень связывания между объектами, позволяющая выполнять основные функции посредством взаимодействия этих объектов.

Шаблон High Cohesion

В терминах объектно-ориентированного проектирования зацепление (cohesion) – это мера связанности и сфокусированности обязанностей класса. Считается, что класс обладает высокой степенью зацепления, если его обязанности тесно связаны между собой и он не выполняет работ непомерных объемов.

Предположим, приложение должно выполнять пятьдесят системных операций, и все они возложены на класс POST. Если этот объект будет выполнять все операции, то он станет чрезмерно "раздутым" и не будет обладать свойством зацепления. И дело не в том, что одна задача создания экземпляра объекта Payment сама по себе снизила степень зацепления объекта POST; она является частью общей картины распределения обязанностей.

Здесь функция создания экземпляра платежа делегирована объекту Sale. Благодаря этому поддерживается более высокая степень зацепления объекта POST. Поскольку такой вариант распределения обязанностей обеспечивает низкий уровень связывания и более высокую степень зацепления, он является более предпочтительным.

На практике уровень зацепления не рассматривают изолированно от других обязанностей и принципов, обеспечиваемых шаблонами Expert и Low Coupling.

Обсуждение. Как и о принципе слабого связывания, о высокой степени зацепления следует помнить в течение всего процесса проектирования. Этот шаблон необходимо применять при оценке эффективности каждого проектного решения.

Вот несколько сценариев, иллюстрирующих различную степень функционального зацепления.

1. Очень слабое зацепление. Класс единолично 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 вызовом удаленных процедур. Это две абсолютно разные функциональные области, требующие кодов больших объемов. Такие обязанности можно распределить между семейством классов, связанных с доступом к реляционной базе данных, и семейством классов, отвечающих за поддержку удаленных процедур.

2. Слабое зацепление. Класс несет единоличную ответственность за выполнение сложной задачи из одной функциональной области.

Допустим, некий класс RDBlnterface полностью отвечает за взаимодействие с реляционными базами данных. Методы этого класса связаны между собой, однако, их слишком много и их реализация требует кодов огромных объемов. Таких методов может быть несколько сотен или даже тысяч. Данный класс следует разделить на несколько более мелких классов, совместно обеспечивающих доступ к реляционным базам данных.

3. 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 реляционными базами данных. Для извлечения и сохранения объектов в базе данных он взаимодействует с десятком других классов-

4. 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 обе логически связаны с понятием компании. К тому же такой класс должен содержать небольшое число открытых методов и требует незначительных объемов кода для их реализации.

Как правило, класс с высокой степенью зацепления содержит сравнительно небольшое

число методов, которые функционально тесно связаны между собой, и не выполняет слишком много функций. Он взаимодействует с другими объектами для выполнения более сложных задач. Классы с высокой степенью зацепления являются очень предпочтительными, поскольку они весьма просты в понимании, поддержке и повторном использовании. Высокая степень однотипной функциональности в сочетании с небольшим числом операций упрощают поддержку и модификацию класса, а также возможность его повторного использования.

Шаблон High Cohesion, как и другие понятия объектно-ориентированной технологии проектирования, имеет аналогию в реальном мире. Всем известно, что человек, выполняющий большое число разнородных обязанностей (особенно тех, которые можно легко распределить между другими людьми), работает не очень эффективно. Это касается менеджеров, которые не умеют распределять обязанности между своими подчиненными. Такие люди страдают от "низкой степени зацепления" и могут легко "расклеиться".

Преимущества

• Повышаются ясность и простота проектных решений

• Упрощаются поддержка и доработка

• Зачастую обеспечивается слабое связывание

•Улучшаются возможности повторного использования, поскольку класс с высокой степенью зацепления выполняет конкретную задачу

Шаблон Controller

Решение. Делегирование обязанностей по обработке системных сообщений классу, удовлетворяющему одному из следующих условий.
Класс представляет всю систему в целом (внешний контроллер)
Класс представляет всю организацию (внешний контроллер)
Класс представляет активный объект из реального мира, который может участвовать в решении задачи {контроллер роли).
Класс представляет искусственный обработчик всех системных событий некоторого прецедента и обычно называется <Прецедент> Handler (контроллер прецедента)

Для всех системных событий в рамках одного прецедента используется один и тот же контроллер. Заметим, что в этот перечень не включаются классы, реализующие окно, аплет, приложение, вид и документ. Такие классы не выполняют задачи, связанные с системными событиями. Они обычно получают сообщения и делегируют их контроллерам.

Проблема. Кто должен отвечать за обработку системных событий?

Системное событие (system event) — это событие высокого уровня, генерируемое внешним исполнителем (событие с внешним входом). Системные события связаны с системными операциями (system operation), т.е. операциями, выполняемыми системой в ответ на события. Например, когда кассир в системе розничной торговли нажимает кнопку End Sale, он генерирует системное событие, свидетельствующее о завершении торговой операции. Аналогично, когда пользователь текстового процессора выбирает команду Орфография, он генерирует системное событие "проверить орфографию".

Контроллер (controller) — это объект, не относящийся к интерфейсу пользователя и отвечающий за обработку системных событий. Контроллер определяет методы для выполнения системных операций.