Билет 5.

1. Представление одномерного уравнения волны в виде системы уравнений с частными производными первого порядка.

Задачи с начальными условиями для уравнения сохранения потока.

Большой класс одномерных задач с начальными условиями можно привести к уравнению сохранения потока: $\frac{\partial u}{\partial t}=-\frac{\partial F(u)}{\partial x}$ (2.1) где **u** и **F** векторы, и в некоторых случаях **F** зависит не только от **u**, но и от пространственных производных от **u**.

Например для одномерного волнового уравнения с постоянной скоростью распространения волны

$\frac{\partial^2 u}{\partial t^2}=v^2\frac{\partial^2 u}{\partial^2 x}$ (2.2) можно написать два уравнения первого порядка $\frac{\partial r}{\partial t}=v\frac{\partial s}{\partial x}$ $\frac{\partial s}{\partial t}=v\frac{\partial r}{\partial x}$ (2.3)

где $r=v\frac{\partial u}{\partial x}$ $s=\frac{\partial u}{\partial t}$ (2.4) В этом случае r и s – две компоненты **u**, а поток определяется линейным матричным уравнением $F(u)=\begin{pmatrix}0 & -v\\ -v & 0\end{pmatrix}\cdot u$ (2.5)

Уравнения (2.3) аналогичны уравнениям Максвелла для одномерной электромагнитной волны. Рассмотрим уравнение для скалярной величины u $\frac{\partial u}{\partial t}=-v\frac{\partial u}{\partial x}$ (2.6) Общее решение этого уравнения – волна, распространяющаяся в положительном направлении оси x. $u=f(x\text{-}vt)$ (2.7) Введем дискретные точки по x и t $x_j=x_0+j\Delta x \quad j=0,1,\ldots,J \qquad t_n=t_0+n\Delta t \quad n=0,1,\ldots,N$ (2.8). Пусть u_j обозначает $u(t_n,x_j)$. Мы можем различными способами представить производную по времени, самый очевидный:

$\frac{\partial u}{\partial t}|_{j,n}=\frac{u_j^{n+1}-u_j^n}{\Delta t}+O(\Delta t)$ (2.9) Это приближение первого порядка точности по временному шагу. Для пространственной производной используем приближение второго порядка.

$\frac{\partial u}{\partial x}|_{j,n}=\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}+O(\Delta x^2)$ (2.10) Тогда конечно-разностная аппроксимация рассматриваемого уравнения, называемая FTCS (Forward Time Centered Space) примет вид: $\frac{u_j^{n+1}-u_j^n}{\Delta t}=-v\left(\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}\right)$ (2.11) Это выражение легко представляется как выражение для u_j в момент времени n+1. Это явная схема, т.е. u_j в момент времени n+1 можно вычислить явно по известным величинам. FTCS алгоритм - это пример одноуровневой схемы, т.е. для нахождения u_j в момент времени n+1 достаточно значений в момент времени n. *Анализ стабильности фон Неймана.*

Представим, что коэффициенты рассматриваемых уравнений с частными производными меняются очень слабо в пространстве и времени, и их можно считать

постоянными. В этом случае независимые решения или собственные моды уравнений можно искать в виде $u_j^n = \xi^n e^{ikj\Delta x}$ (2.12) где k действительное пространственное волновое число, которое может принимать любое значение, и ξ=ξ (k) – комплексное волновое число, которое зависит от k. Временная зависимость единственной собственной моды ни что иное, как последовательные (возрастающие) целые степени комплексной величины ξ. Следовательно, разностные уравнения нестабильны, если |ξ(k)|>1 для каких-то значений k. Подставляя (.1.12) в (.1.11), получим $\xi(k) = 1 - i\frac{v\Delta t}{\Delta x}\sin k\Delta x$ (2.13) Как видно из этого выражения |ξ(k)|>1 **для всех k**, т.е. схема FTCS нестабильна. Несмотря на возможно недостаточную строгость метода фон Неймана, он практически всегда дает верный ответ

2. GRASP: шаблоны для распределения обязанностей.

Шаблоны

Опытные разработчики объектно-ориентированных систем сформулировали общие принципы и стандартные решения, помогающие в разработке программного обеспечения. Если эти принципы и идиомы систематизировать и структурировать, а также присвоить им имена, то их можно применять в качестве шаблонов (patterns). Приведем пример одного из таких шаблонов.

В объектно-ориентированной технологии проектирования шаблоном называют описание проблемы и ее решения, которые можно применить при разработке других систем. В идеале, шаблон должен содержать советы по поводу его применения в различных ситуациях. Многие шаблоны содержат рекомендации по распределению обязанностей между объектами с учетом специфики задачи. Проще говоря, шаблон — это именованная пара "проблема/решение", которую можно применять в различных контекстах, и содержащая рекомендации для применения в различных конкретных ситуациях.

Шаблоны обычно не содержат новых идей. Шаблоны не предназначены для изучения и выражения новых принципов разработки программного обеспечения. Скорее, наоборот. Они призваны систематизировать существующие знания, идиомы и принципы. Чем шире они используются, тем лучше. Следовательно, шаблоны GRASP (с которыми мы вскоре познакомимся) не содержат новых идей, а лишь постулируют широко используемые базовые принципы.

Шаблоны GRASP: общие принципы распределения обязанностей.

шаблоны GRASP - это шаблоны, описывающие фундаментальные принципы распределения обязанностей между объектами.

Аббревиатура GRASP означает General Responsibility Assignment Software Patterns (Общие шаблоны распределения обязанностей в программных системах). Сейчас рассмотрим первые пять шаблонов GRASP.

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

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?

Обсуждение. При распределении обязанностей шаблон Expert используется гораздо чаще любого другого шаблона. Он отражает обычный интуитивно понятный подход: заключается в том, что объекты выполняют действия, связанные с имеющейся у них информацией.

Преимущества

• Шаблон Expert поддерживает инкапсуляцию.

• Нужное поведение системы обеспечивается несколькими классами, содержащими требуемую информацию. Это приводит к определениям классов, которые гораздо проще понимать и поддерживать.

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? Правильно распределив обязанности при проектировании, можно создать слабо связанные независимые компоненты с возможностью их дальнейшего использования, упростить их, а также обеспечить инкапсуляцию данных.

Обсуждение. Основным назначением шаблона Creator является выявление объекта-создателя, который при возникновении любого события должен быть связан со всеми созданными им объектами.

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.

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.

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.

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

Вот несколько сценариев, иллюстрирующих различную степень функционального зацепления.

1. Очень слабое зацепление. Класс единолично отвечает за выполнение множества операций в самых различных функциональных областях.

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

3. Сильное зацепление. Класс имеет среднее количество обязанностей из одной функциональной области и для выполнения своих задач взаимодействует с другими классами.

4. Среднее зацепление. Класс имеет несложные обязанности в нескольких различных областях, логически связанных с концепцией этого класса, но не связанных между собой.

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.

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.

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.

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

Обсуждение. Большинство систем получает внешние события. Обычно они связаны с графическим интерфейсом пользователя. Кроме того, системе могут передаваться внешние сообщения, например, при обработке телекоммуникационных сигналов или сигналов от датчиков в системах управления.

Во всех случаях при использовании объектно-ориентированного подхода для обработки внешних событий применяются контроллеры.

Преимущества

• Улучшение условий для повторного использования компонентов. Этот шаблон обеспечивает обработку процессов предметной области на уровне реализации объектов, а не на уровне интерфейса.

• Контроль состояния прецедента. Иногда необходимо удостовериться, что системные операции выполняются в определенной последовательности. Например, нужно гарантировать, чтобы операция MakePayment выполнялась только после операции EndSale, для чего необходимо накапливать информацию о последовательности событий. Для этого удобно использовать контроллер, особенно контроллер прецедента.

Обязанности, роли и карты CRC.

Для распределения обязанностей и выявления взаимодействия между объектами иногда используется еще одно средство, формально не являющееся частью языка UML, — карты CRC: внесли значительный вклад в повышение абстрактности мышления разработчиков объектно-ориентированных систем и уделили большое внимание распределению обязанностей и взаимодействию объектов.

Карты CRC — это индексные карты (по одной на каждый класс), на которых кратко описаны обязанности классов и перечислены объекты, взаимодействующие с данным классом при выполнении этих обязанностей. Обычно они разрабатываются в процессе небольших ролевых игр (role play), участники которых играют роли различных классов. Каждый человек выбирает карты CRC для тех классов, роль которых он исполняет,