Курганский государственный университет
Кафедра программного обеспечения автоматизированных систем

К У Р С Л Е К Ц И Й

по дисциплине

ВВЕДЕНИЕ В ПРОГРАММНУЮ ИНЖЕНЕРИЮ

для студентов направления 231000.62

«Программная инженерия»

Лекция 3.2. Введение в U.M.L.

План лекции

1. Объектно-ориентированный подход
2. Краткая история развития и стандартизации языка UML
3. Структура и базовые понятия языка UML
   - 3.1 Назначение и особенности языка UML
   - 3.2 Общая структура языка UML
   - 3.3 Элементы моделей языка UML
   - 3.4 Диаграммы языка UML
   - 3.5 Общие правила изображения UML-Диаграмм
4. Контрольные вопросы и задания

Объектно-ориентированный подход к созданию сложных программных систем

Необходимость анализа предметной области до начала написания программы была осознана задолго до появления методологии объектно-ориентированного анализа и проектирования (ООАП).

Было замечено, что процесс разработки компьютерной базы данных существенно отличается от написания программного кода для решения вычислительной (даже весьма алгоритмически сложной) задачи.

При проектировании базы данных возникает необходимость в предварительной разработке ***информационной модели предметной области*** - концептуальной схемы, которая отражала бы взаимосвязи между объектами предметной области, существенные с точки зрения поддержания целостности данных и реализации пользовательских поисковых запросов.

При выполнении масштабных программных проектов, включающих множество взаимосвязанных баз данных, клиентских приложений, ориентированных на несколько групп пользователей, необходимость в моделировании проявляется еще более явно.

Объектно-ориентированный подход к созданию сложных программных систем

Объектный подход применяется на всех основных стадиях жизненного цикла ПО и включает в себя три ключевых понятия:

- OOA (object oriented analysis) - объектно-ориентированный анализ.
- OOD (object oriented design) - объектно-ориентированное проектирование.
- OOP (object oriented programming) - объектно-ориентированное программирование.

Объектно-ориентированный анализ - это методология анализа предметной области, при которой требования к проектируемой системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.

Объектно-ориентированное проектирование - это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

Объектно-ориентированное программирование - это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Предметом рассмотрения нашей дисциплины являются первые две составляющие объектного подхода – объектно-ориентированный анализ и объектно-ориентированное проектирование (далее – ООАП).

Базовые принципы ООАП

1. Принцип декомпозиции

Декомпозиция – это разбиение целого на составные элементы.

В рамках объектного подхода рассматривают два вида декомпозиции: алгоритмическую и объектную.

- ***Алгоритмическая декомпозиция:** при анализе задачи разработчик пытается понять, какие алгоритмы необходимо разработать для ее решения, каковы спецификации этих алгоритмов (вход, выход), и как эти алгоритмы связаны друг с другом. В языках программирования данный подход в полной мере поддерживается средствами модульного программирования (библиотеки, модули, подпрограммы).*
- ***Объектная декомпозиция** предполагает выделение основных содержательных элементов задачи, разбиение их на типы (классы), определение свойств (данных) и поведения (операций) для каждого класса, а также взаимодействия классов друг с другом. Объектная декомпозиция поддерживаются всеми современными объектно-ориентированными языками программирования.*

Базовые принципы ООАП

2. Принцип абстрагирования

Абстрагирование применяется при решении многих задач - любая модель позволяет абстрагироваться от реального объекта, подменяя его изучение исследованием формальной модели.

Абстрагирование в ООАП позволяет выделить основные элементы предметной области, обладающие одинаковой структурой и поведением. Такое разбиение предметной области на абстрактные классы позволяет существенно облегчить анализ и проектирование системы.

Согласно принципу абстрагирования в модель включаются только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций.

Базовые принципы ООАП

3. ***Принцип иерархичности*** предписывает рассматривать процесс построения модели на разных уровнях детализации в рамках фиксированных представлений. Иерархия упорядочивает абстракции, помогает разбить задачу на уровни и постепенно ее решать по принципу "сверху – вниз" или "от общего – к частному", увеличивая детализацию ее рассмотрения на каждом очередном уровне.
4. ***Принцип многомодельности*** утверждает, что никакая единственная модель не может с достаточной степенью адекватности описывать различные аспекты сложной системы, и допускающий использование нескольких взаимосвязанных представлений, отражающих отдельные аспекты поведения или структуры систем.

Схема взаимосвязей представлений и моделей сложных систем в процессе ООАП

На двух уровнях иерархии (концептуальном и физическом) используются статические и динамические представления сложной системы

ООАП и CASE

Методология ООАП тесно связана с концепцией автоматизированной разработки ПО, однако появление первых CASE-средств было встречено программистами с определенной настороженностью.

Причины:

Первая причина заключается в том, что ранние CASE-средства были простой надстройкой над некоторой СУБД и решали задачу визуализации процесса разработки концептуальных схем баз данных (построения ER-диаграмм). При всей важности этой задачи в программном проекте, она не решает проблем разработки приложения в целом.

Вторая причина связана с графической нотацией, реализованной в том или ином CASE-средстве. Если языки программирования имеют строгий синтаксис, то попытки предложить подходящий синтаксис для визуального представления концептуальных схем БД были восприняты далеко не однозначно.

U.M.L. – основа разработки CASE-средств

Появление унифицированного языка моделирования (UML), ориентированного на комплексное решение задачи построения модели программной системы, которую, согласно современным концепциям ООАП, следует считать результатом первых двух этапов ЖЦ ПО, было воспринято сообществом корпоративных программистов с большим оптимизмом.

История развития и стандартизации языка UML

Первые языки объектно-ориентированного моделирования стали появляться в конце 1970-х, к 1994 году их число возросло до 50 и создалась ситуация непримиримой конкуренции между ними, которая даже получила название "войны методов".

Ни один из языков не удовлетворял всем требованиям, предъявляемым к построению моделей сложных систем.

К середине 1990-х некоторые из методов были существенно улучшены и приобрели самостоятельное значение при решении различных (частных) задач ООАП:

- Метод Гради Буча, получивший название ***Booch'93*** - нашел наибольшее применение на этапах проектирования и разработки различных программных систем.
- Метод Джеймса Румбаха, получивший название ***Object Modeling Technique – OMT-2*** - наиболее подходил для анализа процессов обработки данных в информационных системах.
- Метод Айвара Джекобсона, получивший название ***Object-Oriented Software Engineering – OOSE*** - содержал средства представления вариантов использования, которые имеют существенное значение на этапе анализа требований в процессе проектирования бизнес-приложений.

История развития и стандартизации языка UML

1994 - 1995 *Начало работы по унификации* объектно-ориентированных методов Booch, OMT и OOSE. Компании Rational Software Corporation (США) и Objectory AB (Швеция), начиная работу по унификации методов с целью объединения их достоинств, сформулировали следующие ***требования к языку моделирования:***

- язык должен быть универсальным, то есть позволять моделировать не только программные системы, но и более широкие классы систем и бизнес-приложений, с использованием объектно-ориентированных понятий;
- язык должен явным образом обеспечивать взаимосвязь между базовыми понятиями моделей концептуального и физического уровней;
- язык должен обеспечивать масштабируемость моделей, что является важной особенностью сложных многоцелевых систем;
- язык должен быть понятен аналитикам и программистам, а также должен поддерживаться специальными инструментальными средствами, реализованными на различных компьютерных платформах.

История развития и стандартизации языка UML

1995 *К разработке UML подключается консорциум OMG* (Object Management Group – сегодня в состав OMG входят более 800 компаний и организаций).

1996 *Вышло первое описание языка UML* версии 0.9, имевшее статус *запроса предложений* (*RTP - Request For Proposals*), и с этого момента началось широкое обсуждение языка UML различными категориями специалистов.

1996 *Компания Rational Software учредила консорциум партнеров UML*, в который вошли такие компании, как DEC, HP, i-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI, Unisys. Эти компании обеспечили поддержку последующей работы по более точному и строгому определению нотации языка UML.

История развития и стандартизации языка UML

1997 *Опубликован документ с описанием языка UML 1.0*. Эта версия языка была достаточно хорошо определена, обеспечивала требуемую выразительность и мощность и предполагала решение широкого класса задач.

1997 *Принята в качестве стандарта OMG новая версия языка - UML 1.1.* Основное внимание при разработке этой версии было уделено достижению большей ясности семантики языка по сравнению с UML 1.0, а также учету предложений новых партнеров.

1998 Компания Rational Software Corporation разработала и выпустила в продажу одно из первых CASE-средств ***Rational Rose 98***, в котором был реализован язык UML.

1999 Консорциум OMG опубликовал описание языка UML 1.3

2001 Выпуск версии UML 1.4.

2003 Выпуск версии UML 1.5.

2004 Выпуск версии UML 2.0.

2005 *Выпуск стандарта ISO/IEC 19501*:2005 Information technology - Open Distributed Processing - Unified Modeling Language (UML).

Назначение и особенности языка UML

UML не является языком программирования - он служит средством для решения задач объектно-ориентированного моделирования систем.

Основное назначение языка UML - *визуальное моделирование и документирование моделей сложных систем самого различного целевого назначения*. По замыслу OMG, *язык UML должен содержать средства описания достаточно тонких деталей реализации моделей и* тем самым заполнить разрыв между общей методологией моделирования сложных систем и конкретными инструментальными средствами быстрой разработки приложений.

Для более точного представления моделей систем в конкретной предметной области в описании языка UML заложен механизм *расширения* его *базовых понятий*, который является самостоятельным элементом языка и имеет собственное описание в форме правил расширения.

Описание языка UML поддерживает такую спецификацию моделей, которая не зависит от конкретных языков программирования и инструментальных средств проектирования программных систем.

Предполагается, что программная поддержка конструкций языка UML осуществляется специальными инструментальными CASE-средствами, наличие которых имеет принципиальное значение для широкого распространения языка UML.

С другой стороны, сам этот язык призван поощрять развитие рынка инструментальных средств, поддерживающих объектно-ориентированные технологии, и способствовать распространению и продвижению этих технологий.

Общая структура языка UML

Описание языка UML состоит из двух взаимосвязанных и дополняющих друг друга частей:

- ***Семантика*** *языка UML*. Представляет собой метамодель, определяющую абстрактный синтаксис и семантику понятий объектного моделирования на UML (элементов языка).
- ***Графическая нотация*** *языка UML*. Представляет собой систему обозначений для визуального представления элементов языка на *UML-диаграммах*.

Семантика языка определяется для двух категорий объектных моделей: структурных моделей и моделей поведения.

- ***Структурные модели*** (статические) описывают структуру сущностей или компонентов некоторой системы, включая их классы, интерфейсы, атрибуты и отношения.
- ***Модели поведения*** (динамические) описывают поведение или функционирование объектов системы, включая их методы, взаимодействие и сотрудничество между ними, а также процесс изменения состояний отдельных компонентов и системы в целом.

Элементы языка UML

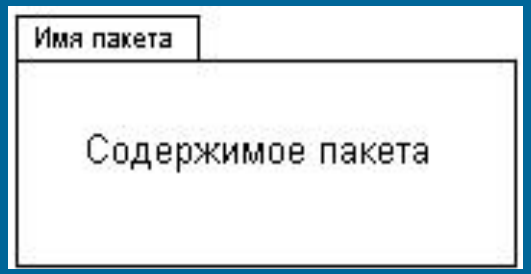
Язык UML использует более 200 элементов для описания моделей, число элементов возрастает с появлением новых версий языка.

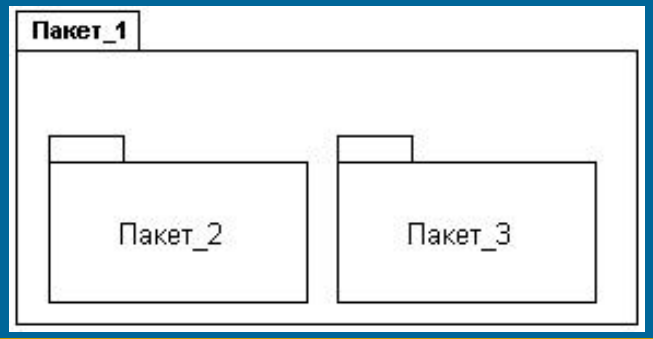
Все элементы языка UML иерархически сгруппированы в логические блоки – так называемые *пакеты*.

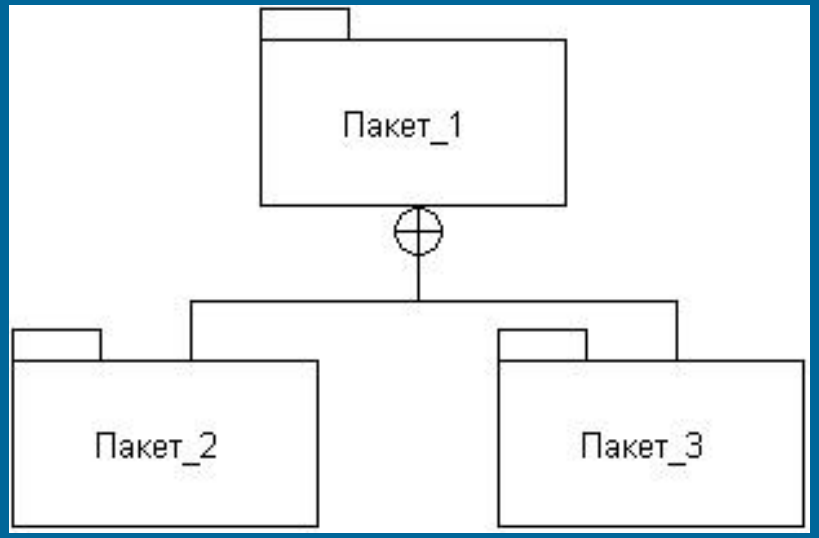
Пакет, включающий другие пакеты, называется *метапакетом* или *контейнером*.

Пакет, включенный в контейнер, называется *подчиненным пакетом*, или более коротко - *подпакетом*

Каждый пакет владеет всеми включенными в него элементами, при этом каждый элемент может принадлежать только одному пакету.







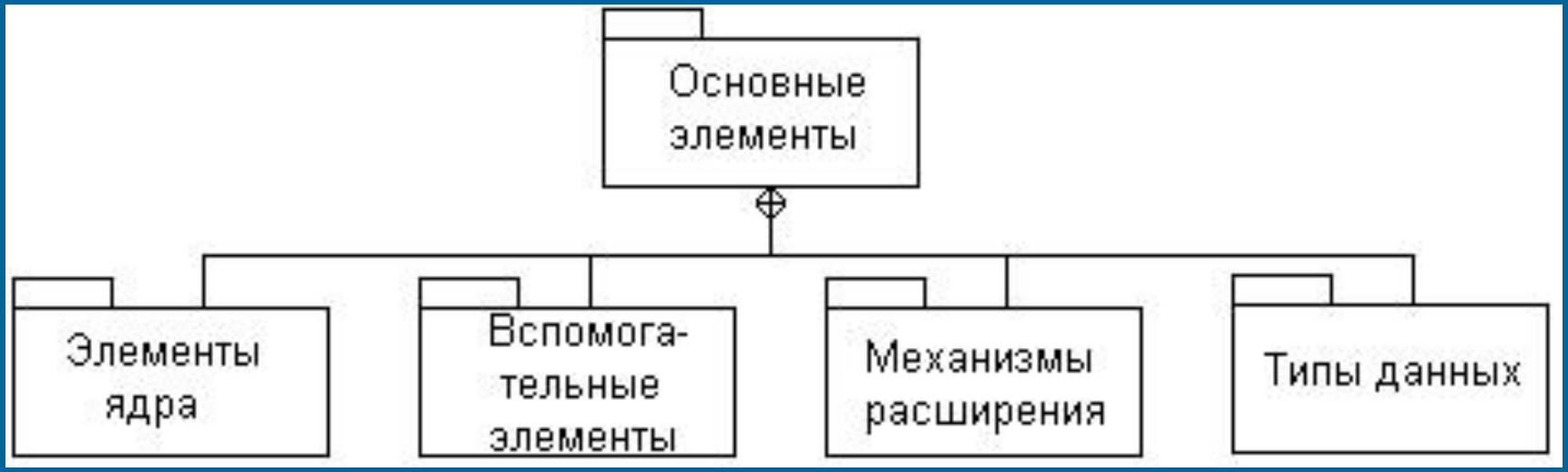
Элементы языка UML

Множество элементов моделей языка UML распределены по иерархически организованным пакетам.

На верхнем (метамодельном) уровне язык UML содержит 3 пакета-контейнера:

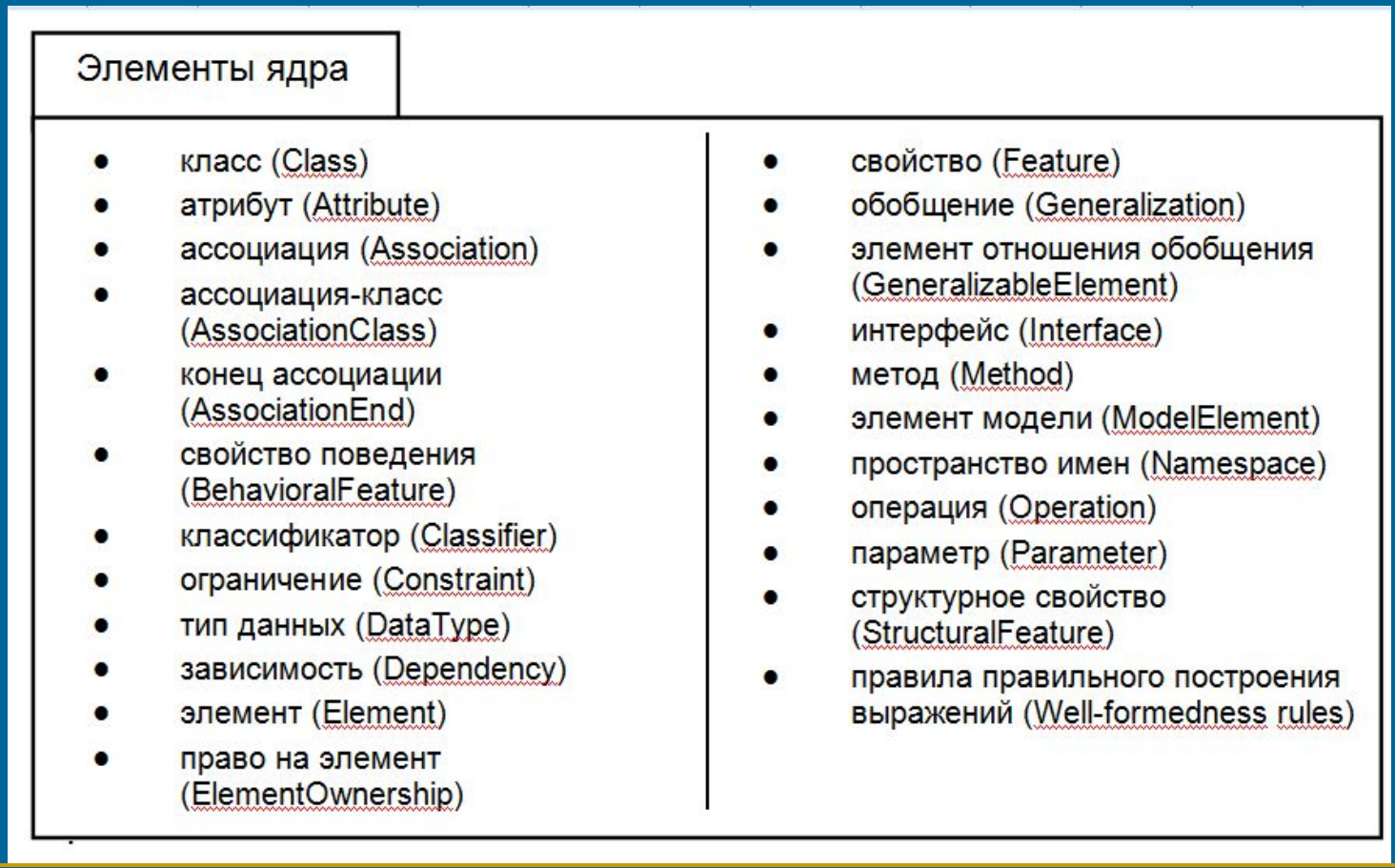
Пакет «*Основные элементы*»

Каждый из пакетов верхнего уровня включает подчиненные ему пакеты. Например, пакет *Основные элементы* включает четыре подпакета:



Пакет "*Элементы ядра*"

Пакет "*Элементы ядра*" является наиболее фундаментальным, он не содержит подчиненных пакетов и включает основные метаклассы языка UML



Пакет "*Элементы поведения*"

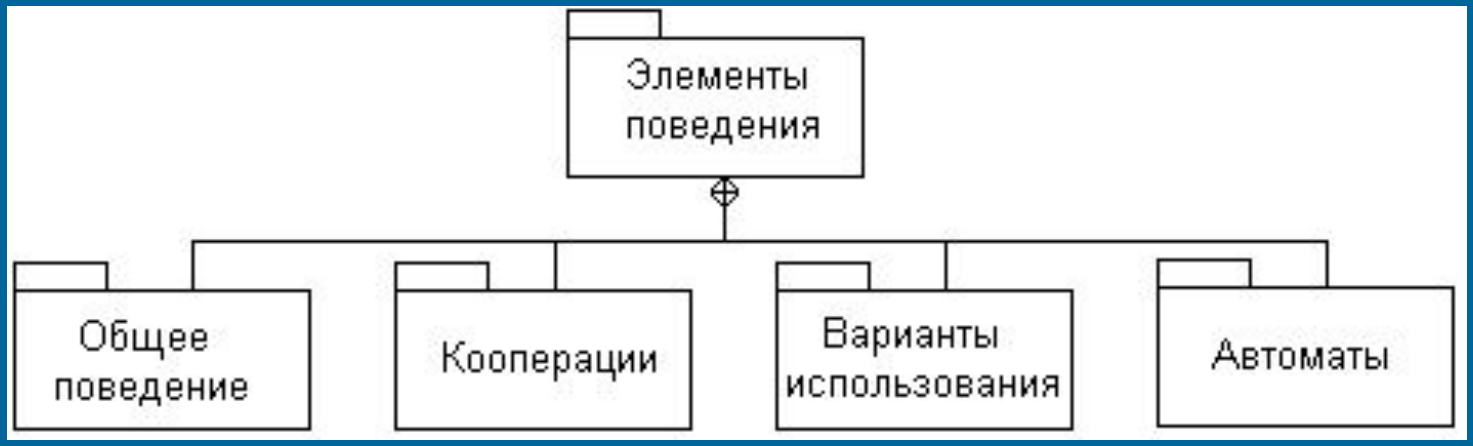
В качестве еще одного примера компонентов моделей языка UML рассмотрим структуру пакета "*Элементы поведения*". Этот пакет является самостоятельным компонентом языка UML (слайд №17) и, как следует из его названия, специфицирует поведение динамических элементов в нотации UML.

В языке UML под поведением понимается не только процесс изменения атрибутов объектов в результате выполнения операций над их значениями, но и такие процедуры, как создание и уничтожение самих объектов.

При этом динамика взаимодействия объектов, которая определяет их поведение, описывается с помощью специальных понятий – *сигналов* и *действий*.

Пакет "*Элементы поведения*" состоит из четырех подпакетов :

Пакет "*Элементы поведения*"



В подпакете **"*Общее поведение*"** определены базовые понятия, необходимые для всех динамических элементов, включенных в другие подпакеты контейнера "*Элементы поведения*".

В подпакет входит большое число элементов: объект, действие, аргумент, связь, сигнал и ряд других.

Наиболее важным из этих элементов является ***объект***.

Под объектом в языке UML понимается отдельный экземпляр класса, структура и поведение которого полностью определяется порождающим этот объект классом.

Пакет *"Кооперации"*

Пакет *"Кооперации"* определяет понятия, которые необходимы для ответа на вопрос: «*Как различные элементы модели взаимодействуют между собой с точки зрения структуры ?*»

Этот пакет использует конструкции, определенные в пакетах "*Основные элементы*" (Слайд №18) и "*Общее поведение*" (слайд №21).

В пакет входят следующие элементы:

- ***кооперация*** (Collaboration)
- ***взаимодействие*** (Interaction)
- ***сообщение*** (Message)
- ***роль ассоциации*** (AssociationRole)
- ***роль классификатора*** (ClassifierRole)
- ***роль конца ассоциации*** (AssociationEndRole).

Понятие кооперации имеет важное значение для представления взаимодействия элементов модели с точки зрения классификаторов и ассоциаций.

Элементы этого пакета непосредственно используются при построении ***диаграмм кооперации***.

Пакет *"Варианты использования"*

Пакет *"Варианты использования"* специфицирует поведение специальных конструкций модели, которые в языке UML называются *актерами* и *вариантами использования*.

Эти понятия служат для первоначального определения поведения (функциональности) моделируемой сущности без спецификации ее внутренней структуры.

В пакет входят следующие элементы:

- ***актер*** (Actor),
- ***вариант использования*** (UseCase),
- ***расширение*** (Extension),
- ***точка расширения*** (ExtensionPoint),
- ***включение*** (Include),
- ***экземпляр варианта использования*** (UseCaseInstance).

Более подробно эти понятия будут рассмотрены при изучении ***диаграмм вариантов использования***.

Пакет «*Автоматы*»

Пакет *Автоматы* содержит множество понятий, которые необходимы для представления поведения системы в виде конечного числа ее состояний и переходов между ними.

Под состоянием в языке UML понимается абстрактный метакласс, используемый для моделирования ситуации или процесса, в ходе которых имеет место выполнение некоторого условия. Примером такого условия может быть состояние ожидания объектом выполнения внешнего события (запроса, передачи управления, …).

Автоматы являются основным средством моделирования поведения различных элементов языка UML. Например, автоматы могут использоваться для моделирования поведения экземпляров классов, а также для спецификации взаимодействий между сущностями, таких, как кооперации.

С другой стороны, состояние может использоваться для моделирования процессов выполнения некоторой деятельности: момент начала выполнения деятельности является переходом объекта в соответствующее состояние.

В пакет включены следующие элементы:

- ***автомат*** (StateMachine),
- ***состояние*** (State),
- ***простое состояние*** (SimpleState),
- ***составное состояние*** (CompositeState),
- ***псевдосостояние*** (PseudoState),
- ***конечное состояние*** (FinalState),
- ***событие*** (Event)
- ***переход*** (Transition).

Диаграммы языка UML

UML позволяет описывать систему на различных уровнях абстракции (слайд №8) следующими моделями:

- *Модели функционирования* - показывают, как описывается функциональность системы с точки зрения пользователя.
- *Объектные модели* - показывают, как выглядит проект системы с точки зрения объектного подхода.
- *Динамические модели* - показывают, как взаимодействуют во времени компоненты системы, и демонстрирует процессы, происходящие в системе.

Все представления о моделях сложной системы в языке UML фиксируются в виде специальных графических конструкций, получивших название диаграмм.

UML-диаграммы предназначены для визуализации моделей и их компонентов.

Процесс ООАП неразрывно связан с процессом построения диаграмм. При этом совокупность UML-диаграмм, разработанных при проектировании сложной системы, содержит всю информацию, которая необходима для реализации проекта.

Диаграммы языка UML

Диаграммы языка UML представлены тремя категориями:

Структурные диаграммы:

- *Диаграмма классов* (class diagram) - показывает классы, их атрибуты и связи между классами.
- *Диаграмма компонентов* (component diagram) - показывает компоненты и связи между ними.
- *Диаграмма развертывания* (deployment diagram) - показывает, как ПО размещается на аппаратуре (серверах, рабочих станциях...).

Диаграммы поведения:

- *Диаграмма вариантов использования* (use case diagram) - показывает работу системы с точки зрения пользователей.
- *Диаграмма состояний* (statechart diagram)- представляет собой конечный автомат, показывающий функционирование системы.
- *Диаграмма деятельности* (activity diagram) - показывает потоки информации в системе.

Диаграммы взаимодействия

- *Диаграмма кооперации* (collaboration diagram) - показывает структурную организацию участвующих во взаимодействии объектов.
- *Диаграмма последовательности* (sequence diagram) - показывает временную упорядоченность событий.

Диаграммы языка UML

Каждая из диаграмм детализирует и конкретизирует различные представления о модели сложной системы в терминах языка UML.

Диаграмма вариантов использования.

- представляет наиболее общую концептуальную модель системы
- разрабатывается на начальной стадии проекта
- Составляет основу для разработки функциональных требований к системе и является исходной для построения всех остальных UML-диаграмм

Диаграмма классов

- представляет логическую модель системы
- отражает статические аспекты структуры системы

Диаграммы поведения

- представляют логическую модель системы
- отражают динамические аспекты функционирования системы.

Диаграмма компонентов и *диаграмма развертывания*

- служат для представления компонентов физической модели системы
- используются для моделирования системы на стадии ее реализации.

Диаграммы языка UML

Интегрированная модель сложной системы в нотации UML представляется в виде совокупности диаграмм:

Элементы и общие правила построения UML-диаграмм

UML-диаграмма - это граф специального вида с вершинами в форме геометрических фигур, которые связаны между собой дугами.

На рисунке в качестве примера показана диаграмма вариантов использования, выполненная в рамках проекта системы продаж товаров по каталогу

Поскольку информация, которую содержит в себе граф, имеет топологический характер, ни геометрические размеры, ни расположение элементов диаграмм, как правило, не имеют принципиального значения.

UML-диаграммы используют конструкции четырех видов:

Графические символы - изображаются с помощью геометрических фигур, внутри которых могут размещаться другие конструкции языка. Чаще всего внутри таких фигур помещаются другие графические символы или текст, уточняющий семантику или фиксирующий отдельные свойства элементов языка.

Пути – изображаются отрезками линий, соединяющих отдельные графические символы. Концы отрезков линий (терминаторы) обязательно должны соприкасаться с соответствующими графическими символами, как это принято в теории графов. Пути могут иметь в качестве терминаторов специальные *значки* (*пиктограммы*).

Значки (***пиктограммы***) - графические фигуры фиксированного размера и формы, внутри значка не допускается размещать дополнительные символы. Значок может размещаться как внутри графического символа, так и вне его на свободном поле диаграммы. Примерами значков могут служить стрелки, указывающие направления отрезков путей, или некоторые другие дополнительные обозначения (например, значок ⊕ обозначает отношение вложенности на диаграммах пакетов).

Текст - используется для представления информации в некоторой грамматической форме. Например, в различных секциях обозначения класса текст может описывать атрибуты этого класса или его операции. При этом предполагается, что использование текста должно соответствовать определенному синтаксису в нотации языка UML, посредством которого может быть реализован грамматический разбор этого текста для получения полной информации о модели. На использование текста накладывается важное условие – семантика всех допустимых символов должна быть заранее определена в языке UML (или в его расширении в конкретной модели).

Рекомендации по построению UML-диаграмм :

1. Каждая диаграмма должна служить законченным представлением соответствующего фрагмента моделируемой предметной области. Отсутствие существенных элементов на диаграмме служит признаком неполноты модели и может потребовать ее последующей доработки.
2. Все сущности одной диаграммы должны принадлежать одному концептуальному уровню представления модели (слайд №8). Диаграммы или их отдельные фрагменты могут детализироваться на других (подчиненных) диаграммах этого же типа, образуя пакет иерархически вложенных диаграмм. Степень детализации диаграмм нижнего уровня должна быть достаточной для последующей генерации программного кода.
3. Необходимо стремиться к явному указанию свойств всех элементов диаграмм, несмотря на то, что язык UML допускает использование значений по умолчанию при отсутствии некоторых символов на диаграмме (например, в случае неявного указания видимости атрибутов и операций классов).
4. Диаграммы не должны содержать противоречивой информации. Противоречивость модели приводит к неоднозначной ее интерпретации и может служить источником проблем при реализации (например, наличие замкнутых путей при изображении отношений агрегирования или композиции, наличие элементов с одинаковыми именами и различными атрибутами свойств в одном пространстве имен).
5. Не следует перегружать диаграммы текстовой информацией - визуализация модели является наиболее эффективной, если она содержит минимум пояснительного текста. Как правило, наличие больших фрагментов текста на диаграмме служит признаком неоднородности модели, когда в рамках одной модели представляется различная по характеру информация.
6. Состав диаграмм, используемых в конкретном программном проекте, не является строго фиксированным и зависит от специфики проекта: для простых приложений нет необходимости строить все без исключения типы диаграмм. Например, модель системы может не содержать диаграмму развертывания для приложения, выполняемого локально на компьютере пользователя.

Заключение

Процесс построения отдельных типов диаграмм имеет свои особенности, которые тесно связаны с семантикой элементов этих диаграмм. Сам процесс ООАП в контексте языка UML получил специальное название – рациональный унифицированный процесс (Rational Unified Process, RUP). Концепция RUP и основные его элементы разработаны А. Джекобсоном в ходе его работы над языком UML.

Контрольные вопросы и задания

1. Перечислите базовые принципы моделирования сложных систем, прокомментируйте эти принципы с привлечением схем, приведенных на рисунках 1 и 9.
2. Расшифруйте следующие сокращения: UML, OMG, RUP.
3. Попытайтесь дать общую характеристику языка UML. Какова область применения этого языка? Чем принципиально отличается язык UML от языка программирования высокого уровня?
4. Определите понятие "*пакет*" языка UML. Для чего могут быть использованы *пакеты* и как они изображаются на UML-диаграммах ?
5. Перечислите основные элементы UML-диаграмм.
6. Перечислите основные типы UML-диаграмм.

Начиная со следующей лекции мы приступим к обзору базовых концепций ООАП и одновременно изучим технику построения UML-диаграмм, используемых на соответствующих стадиях программного проекта.