Анализ и моделирование на UML

Направления подготовки
«Информационные системы и технологии»

Максим Валерьевич Хлопотов,
старший преподаватель кафедры ИС

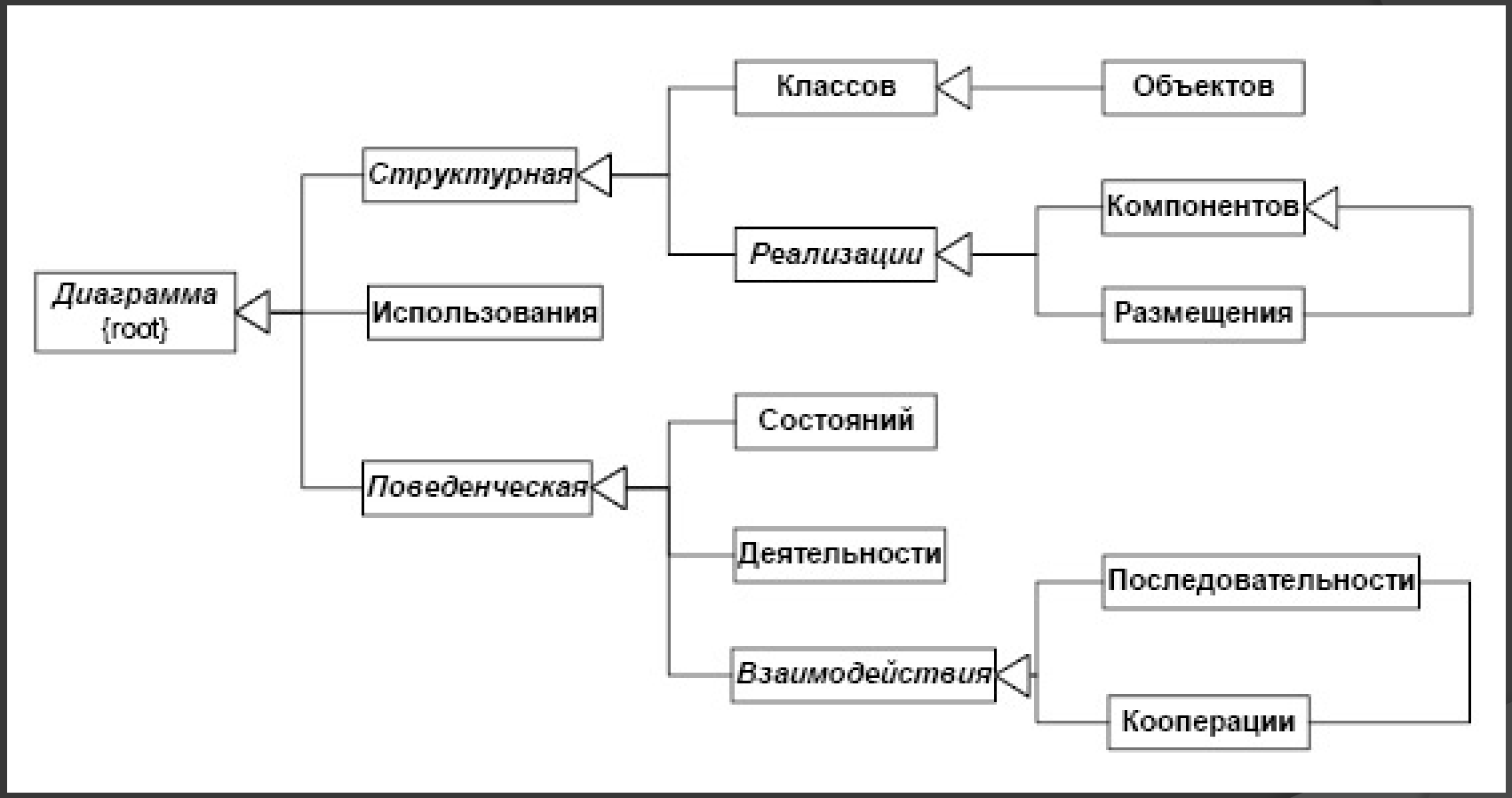
Темы лекционных занятий

1. Введение в UML
2. Моделирование использования
3. **Моделирование структуры**
4. Моделирование поведения
5. Дисциплина моделирования

Назначение UML

UML — это графический язык моделирования общего назначения, предназначенный для спецификации, визуализации, проектирования и документирования всех артефактов, создаваемых при разработке программных систем.

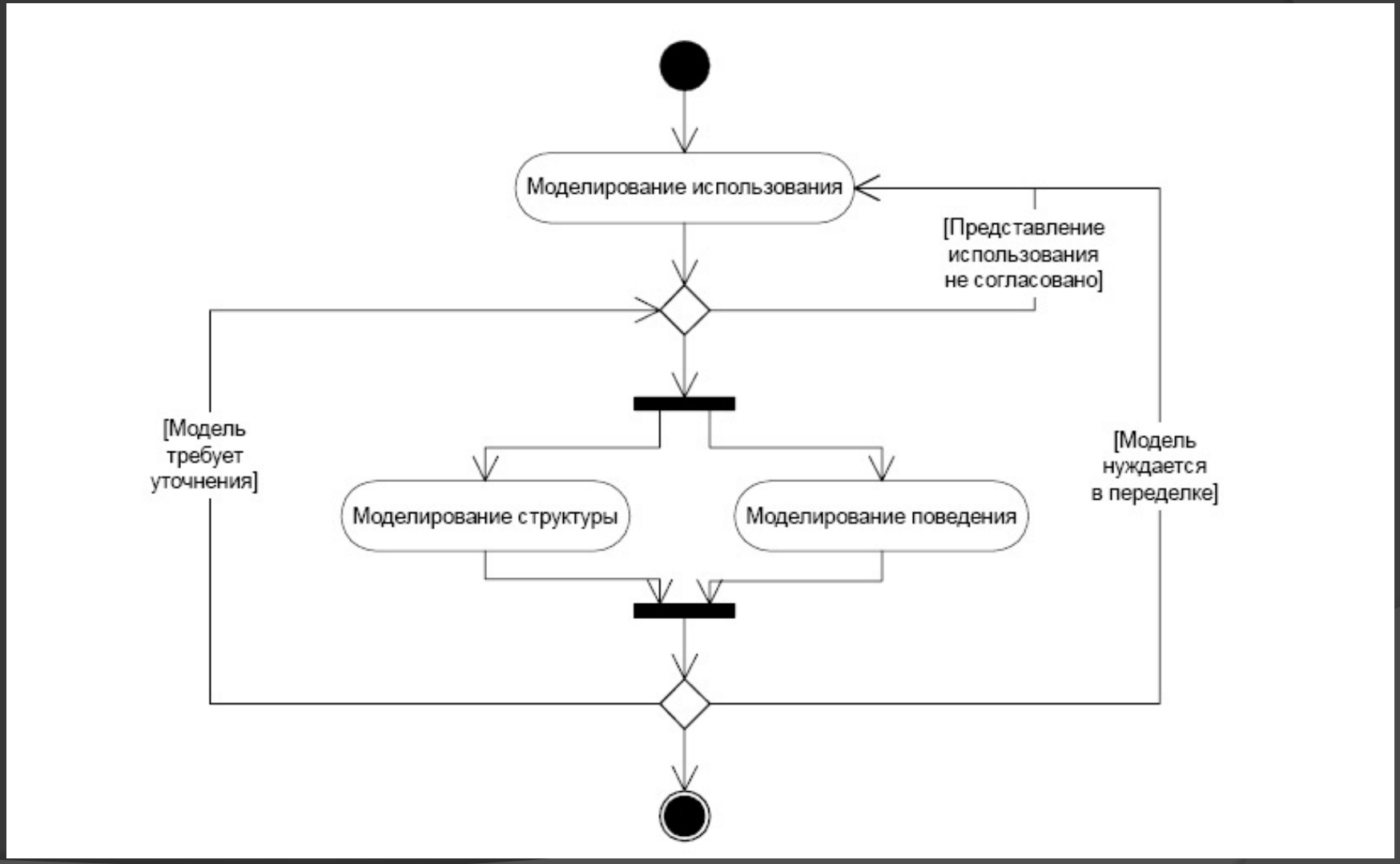
Иерархия диаграмм UML



Представления

Выделим три представления:
- представление использования (что делает система полезного?);
- представление структуры (из чего состоит система?);
- представление поведения (как работает система?).

Процесс моделирования



Моделирование использования

Наш язык и мышление устроены так, что самой простой, понятной и четкой формой изложения мыслей являются так называемые простые утверждения.

Простое утверждение имеет следующую грамматическую форму: подлежащее — сказуемое — прямое дополнение. В логических терминах: субъект — предикат — объект.

Например: начальник увольняет сотрудника, директор создает отдел.

Моделирование использования

По сути, именно простые утверждения и записаны на диаграмме использования.

Действующее лицо — это субъект, а вариант использования — предикат (вместе с объектом).

Моделирование использования предполагает явное формулирование требований к системе на самом начальном этапе разработки.

Реализация вариантов использования

- текстовые описания;
- псевдокод;
- **диаграмма деятельности;**
- *диаграммы взаимодействия.*

- Вариант использования должен доставлять значимый результат, значит, если результата нет, то что-то спроектировано не так, как нужно.

Диаграмма деятельности

- Реализация варианта использования диаграммой деятельности является компромиссным способом ведения разработки — в сущности, это проектирование сверху вниз в терминах и обозначениях UML.
- Эту диаграмму можно показать заказчику, чтобы проверить, действительно ли проектируемая нами логика работы системы соответствует тому бизнес-процессу, который существует в реальности.

Диаграмма деятельности

- Применение диаграмм деятельности для реализации вариантов использования не слишком приближает к появлению целевого артефакта — программного кода, однако может привести к более глубокому пониманию существа задачи и даже открыть неожиданные возможности улучшения приложения, которые было трудно усмотреть в первоначальной постановке задачи.

Моделирование структуры

- Моделируя структуру, мы описываем составные части системы и отношения между ними.
- UML является объектно-ориентированным языком моделирования, поэтому не удивительно, что основным видом составных частей, из которых состоит система, являются объекты.

Моделирование структуры

- В каждый конкретный момент функционирования системы можно указать конечный набор конкретных объектов и связей между ними, образующих систему. Однако в процессе работы этот набор не остается неизменным: объекты создаются и уничтожаются, связи устанавливаются и теряются.
- Число возможных вариантов наборов объектов и связей, которые могут иметь место в процессе функционирования системы, если и не бесконечно, то может быть необозримо велико. Представить их все в модели практически невозможно, а главное бессмысленно, поскольку такая модель из-за своего объема будет недоступна для понимания человеком, а значит бесполезна при разработке системы.

ООП

- Центральной идеей парадигмы объектно-ориентированного программирования является *инкапсуляция*, т. е. структурирование программы на структуры особого вида, объединяющего данные и процедуры их обработки, причем внутренние данные структуры не могут быть обработаны иначе, кроме как предусмотренными для этого процедурами.
- В настоящее время структуру из данных и процедур их обработки, существующую в памяти компьютера во время выполнения программы, чаще всего называют *объектом*, а описание множества однотипных объектов к тексте программы называют *классом*.

ООП

- Объект – структура из данных и процедур их обработки, существующая в памяти компьютера во время выполнения программы.
- Класс – описание множества однотипных объектов к тексте программы.

ООП

- Кроме основной идеи инкапсуляции, с объектно-ориентированным программированием принято ассоциировать также понятия наследования и полиморфизма.

ООП

- Наследование — это способ структуризации описаний многих классов, который позволяет сократить текст программы, сделав его тем самым более обозримым, а значит более надежным и удобным. Если класс A наследует классу B, то говорят, что класс A является суперклассом класса B, а класс B — подклассом класса A. Подкласс содержит все составляющие своего суперкласса и удовлетворяет принципу подстановочности.

ООП

- Полиморфизм — это способ идентификации процедур при вызове.
- В классических процедурных системах программирования процедуры идентифицируются просто по именам. В объектно-ориентированном программировании конкретный метод идентифицируется не только по имени, но и по объекту, которому метод принадлежит, типам и количеству аргументов (т. е. по полной сигнатуре). Таким образом, сходные по назначению и смыслу, но различающиеся в деталях реализации методы разных классов могут быть названы одинаково.

Назначение структурного моделирования

Следующие структуры нужно моделировать на UML:

- структура связей между объектами во время выполнения программы;
- структура хранения данных;
- структура программного кода;
- структура компонентов в приложении;
- структура используемых вычислительных ресурсов;
- структура сложных объектов, состоящих из взаимодействующих частей
- структура артефактов в проекте.

Структурное моделирование

Структура связей между объектами во время выполнения программы

В парадигме ООП процесс выполнения программы состоит в том, что программные объекты взаимодействуют друг с другом, обмениваясь сообщениями. Наиболее распространенным типом сообщения является вызов метода объекта одного класса из метода объекта другого класса.

Для того чтобы вызвать метод объекта, нужно иметь доступ к этому объекту. Один объект "знает" другие объекты и, значит, может вызвать открытые методы, использовать и изменять значения открытых свойств и т.д.

В этом случае, мы говорим что объекты связаны. Для моделирования структуры связей в UML используются отношения ассоциации на диаграмме классов.

Структурное моделирование

Структура хранения данных

Программы обрабатывают данные, которые хранятся в памяти компьютера. В парадигме объектно-ориентированного программирования для хранения данных во время выполнения программы предназначены свойства объектов, которые моделируются в UML атрибутами классов.

Объекты, которые сохраняют (по меньшей мере) значения своих свойств даже после того, как завершился породивший их процесс, мы будем называть хранимыми.

В настоящее время самой распространенным способом хранения объектов является использование системы управления базами данных (СУБД). При этом хранимому классу соответствует таблица базы данных, а хранимый объект (точнее говоря, набор значений хранимых атрибутов) представляется записью в таблице.

Для моделирования структуры хранения данных в UML применяются ассоциации с указанием кратности полюсов.

Структурное моделирование

Структура программного кода

Программы существенно отличаются по величине: от сотен строк кода (и менее) до сотен миллионов строк (и более). Столь большие количественные различия не могут не проявляться и на качественном уровне.

Для маленьких программ структура кода практически не имеет значения, для больших — наоборот, имеет едва ли не решающее значение.

Поскольку UML не является языком программирования, модель не определяет структуру кода непосредственно, однако косвенным образом структура модели существенно влияет на структуру кода.

Структура классов и пакетов в модели UML фактически полностью моделирует структуру кода приложения.

Структурное моделирование

Структура сложных объектов, состоящих из взаимодействующих частей.

Для моделирования этой структуры применяется диаграмма внутренней структуры классификатора (UML 2). В курсе не рассматриваем.

Структурное моделирование

Структура компонентов в приложении

Приложение, состоящее из одной исполнимой компоненты, имеет тривиальную структуру компонентов, моделировать которую нет нужды. Большинство современных приложений состоят из многих компонентов, даже если и не являются распределенными.

Компонентная структура предполагает описание двух аспектов: во-первых, как классы распределены по компонентам, во-вторых, как (через какие интерфейсы) компоненты взаимодействуют друг с другом.

Оба эти аспекта моделируются диаграммами компонентов UML.

Структурное моделирование

Структура используемых вычислительных ресурсов

Многокомпонентное приложение, как правило, бывает распределенным, т. е. различные компоненты выполняются на разных компьютерах.

Диаграммы размещения и компонентов позволяют включить в модель описание и этой структуры.

Диаграммы классов

- Диаграмма классов является основным средством моделирования структуры UML.
- Диаграммы классов наиболее информационно насыщены по сравнению с другими типами канонических диаграмм UML.
- На диаграммах классов в качестве сущностей применяются прежде всего классы, как в своей наиболее общей форме, так и в форме многочисленных стереотипов и частных случаев: интерфейсы, типы данных, процессы и др.
- Кроме того, в диаграмме классов могут использоваться (как и везде) пакеты и примечания.

Диаграммы классов

- Сущности на диаграммах классов связываются главным образом отношениями ассоциации (в том числе агрегирования и композиции) и обобщения.
- Отношения зависимости и реализации на диаграммах классов применяются реже.

Диаграммы классов

- Класс — один из самых "богатых" элементов моделирования UML.
- Описание класса может включать множество различных элементов, и чтобы они не путались, в языке предусмотрено группирование элементов описания класса по разделам.

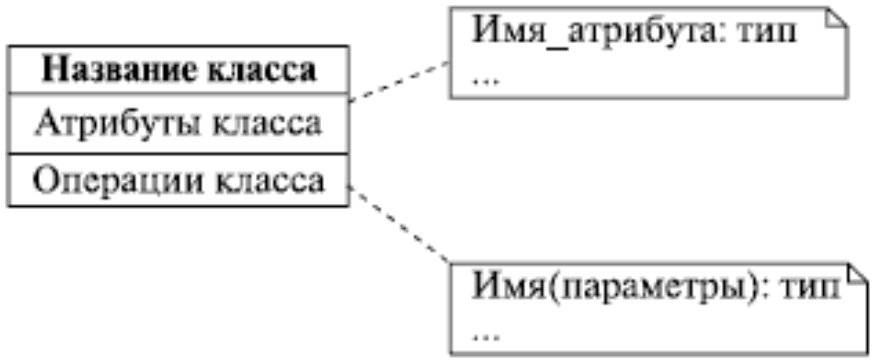
Стандартных разделов три:

- раздел имени — наряду с обязательным именем может содержать также стереотип, кратность и список свойств;
- раздел атрибутов — содержит список описаний атрибутов класса;
- раздел операций — содержит список описаний операций класса.

Класс

Атрибут - это свойство класса, которое может принимать множество значений.

Операция - реализация функции, которую можно запросить у любого объекта класса.



Атрибут

- В общем случае описание атрибута имеет следующий синтаксис:
- видимость ИМЯ кратность : тип = начальное_значение {свойства}

- Видимость, как обычно, обозначается знаками +, –, #. Если видимость не указана, то никакого значения видимости по умолчанию не подразумевается.
- «+» public (открытый доступ)
- «#» protected (только из операций этого же класса и классов, создаваемых на его основе)
- «-» private (только из операций того же класса)

Атрибут

- В обычной ситуации каждый экземпляр класса хранит свое индивидуальное значение атрибута. Если имя атрибута подчеркнуто, то это означает, что областью действия данного атрибута является класс, а не экземпляр класса, как обычно.

- Кратность, если она присутствует, определяет данный атрибут как массив (определенной или неопределенной длины).

- Тип атрибута — это либо примитивный (встроенный) тип, либо тип определенный пользователем

Атрибут

- Начальное значение имеет очевидный смысл: при создании экземпляра данного класса атрибут получает указанное значение. Заметим, что если начальное значение не указано, то никакого значения по умолчанию не подразумевается. Если нужно, чтобы атрибут обязательно имел значение, то об этом должен позаботиться конструктор класса.

- Как и любой другой элемент модели, атрибут может быть наделен дополнительными свойствами в форме ограничений и именованных значений.

Примеры описаний атрибутов

- name
- Минимальное возможное описание — указано только имя атрибута

- +name
- Указаны имя и открытая видимость — предполагается, манипуляции с именем будут производится непосредственно

- -name : String
- Указаны имя, тип и закрытая видимость — манипуляции с именем будут производится с помощью специальных операций

- -name [1..3] : String
- Указана кратность (для хранения трех составляющих; фамилии, имени и отчества)

- -name : String = "Hlopotov"
- Указано начальное значение

Операции

- Выполнение действий, определяемых операцией, инициируется вызовом операции.
- При выполнении операция может, в свою очередь, вызывать операции этого и других классов.
- Описания операций класса перечисляются в разделе операций и имеют следующий синтаксис.
- видимость ИМЯ (параметры) : тип {свойства}
- Здесь слово параметры обозначает последовательность описаний параметров операции. Описания параметров в списке разделяются запятой. Для каждого параметра обязательно указывается имя, а также могут быть указаны направление передачи параметра, его тип и значение аргумента по умолчанию.

Примеры описания операций

- move
- Минимальное возможное описание — указано только имя операции

- +move(in from, in to)
- Указаны видимость операции, направления передачи и имена параметров

- +move(in from : Dpt, in to : Dpt)
- Подробное описание сигнатуры: указаны видимость операции, направления передачи, имена и типы параметров

- +getName() : String {isQuery}
- Функция, возвращающая значение атрибута и не имеющая побочных эффектов

- +setPwd(in pwd : String = "password")
- Процедура, для которой указано значение аргумента по умолчанию

Класс

- Как и все основные сущности UML, класс обязательно имеет имя, а стало быть раздел имени не может быть опущен.
- В разделе имени класса может быть указан стереотип.

Стереотип

Стереотип — это определение нового элемента моделирования в UML на основе существующего элемента моделирования.

Определение стереотипа производится следующим образом. Взяв за основу некоторый существующий элемент модели, к нему добавляют новые помеченные значения (расширяя тем самым внутреннее представление), новые ограничения (расширяя семантику) и дополнения, то есть новые графические элементы (расширяя нотацию).

Стандартные стереотипы классов

actor - действующее лицо

enumeration - перечислимый тип данных

exception - сигнал, распространяемый по иерархии обобщений

implementation class - реализация класса

interface - нет атрибутов и все операции абстрактные

metaclass - экземпляры являются классами

powertype - метакласс, экземплярами которого являются все наследники данного класса

process, thread - активные классы

signal - класс, экземплярами которого являются сообщения

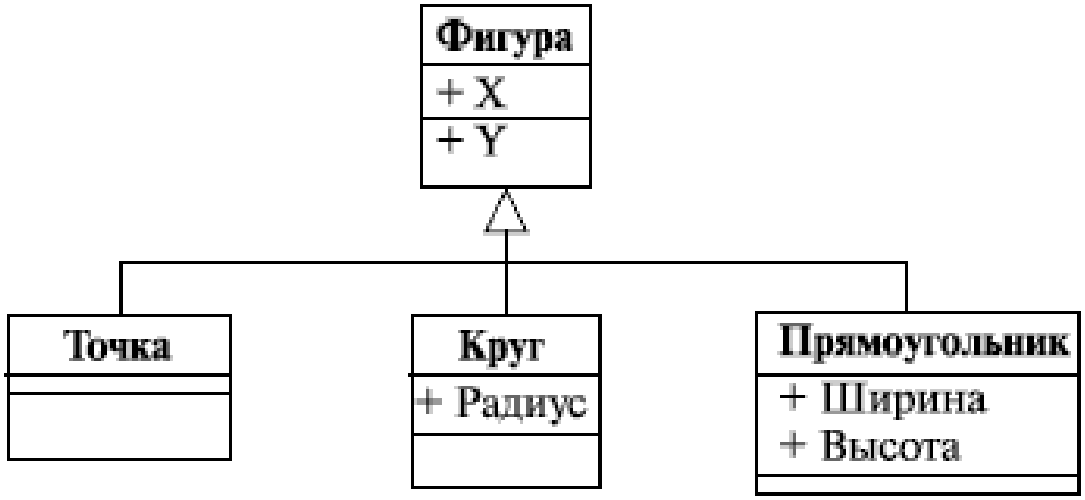
stereotype - стереотип

type (datatype) - тип данных

utility - нет экземпляров (служба)

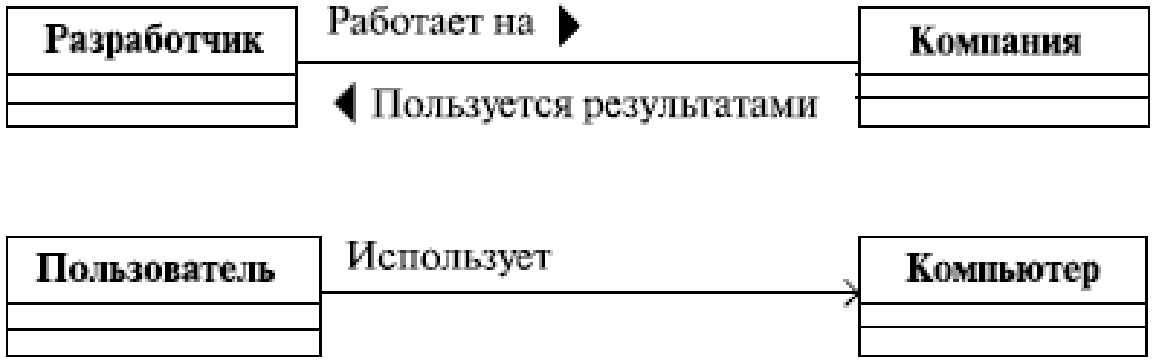
Класс

Обобщение - это отношение между более общей сущностью, называемой *суперклассом*, и ее конкретным воплощением, называемым *подклассом*.



Класс

Ассоциация – связь между объектами, по которой можно между ними перемещаться. Ассоциация может иметь имя, показывающее природу отношений между объектами, при этом в имени может указываться *направление* чтения связи при помощи треугольного маркера. Однонаправленная ассоциация изображается стрелкой.

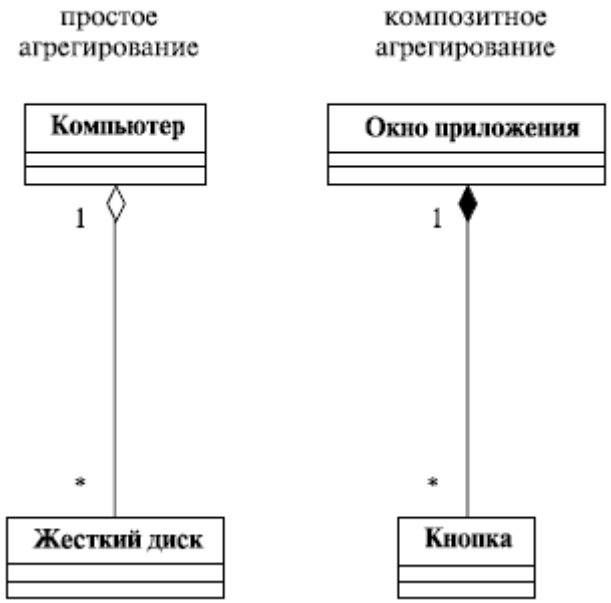


Класс

Роли и кратность

Класс

Ассоциацией с агрегированием – более сложное отношение между классами, связь типа «часть-целое». Один класс имеет более высокий статус (целое) и состоит из низших по статусу классов (частей). При этом выделяют простое и композитное агрегирование (**агрегация** и **композиция)**.
Агрегация предполагает, что части, отделенные от целого, могут продолжать свое существование независимо от него.
Композиция – целое владеет своими частями и их время жизни соответствует времени жизни целого, т. е. независимо от целого части существовать не могут.



Класс

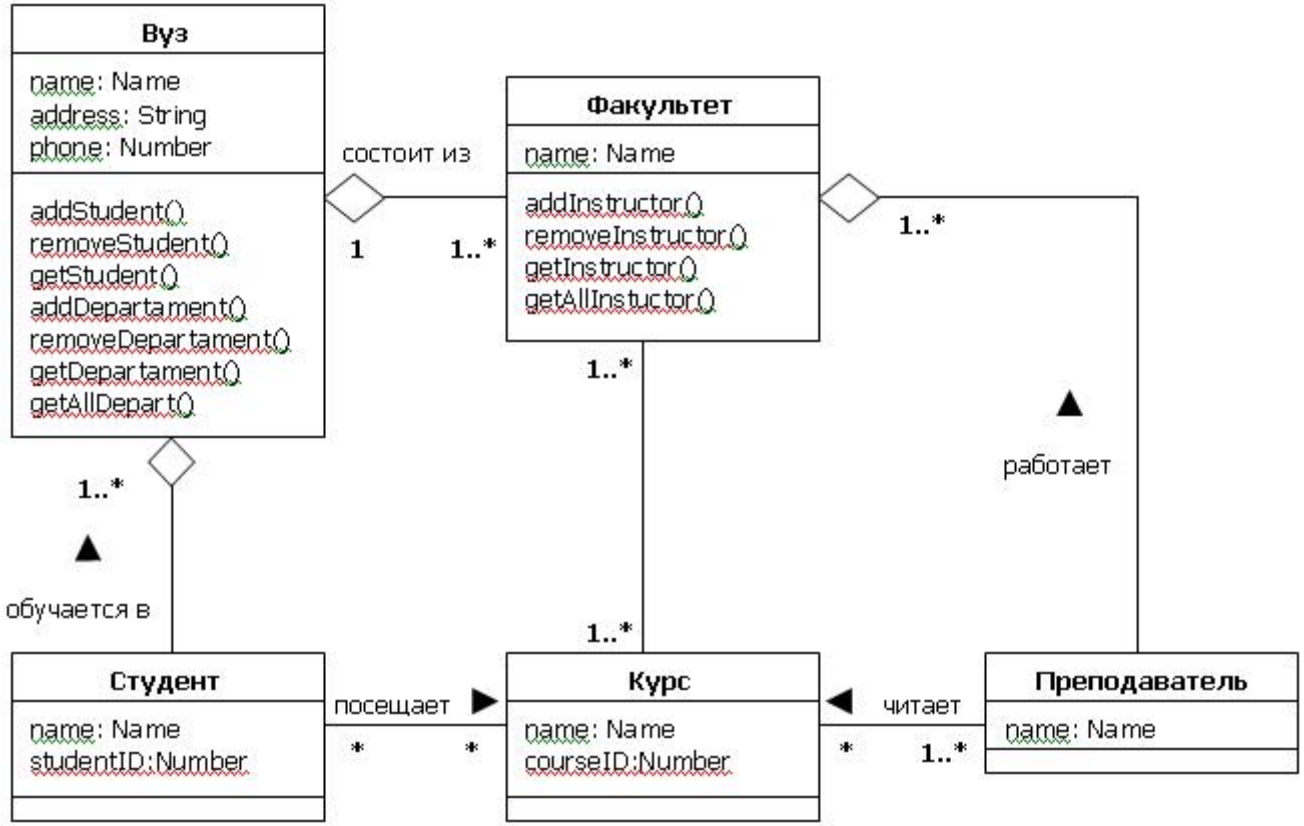

Вуз
name: Name
address: String
phone: Number
addStudent()
removeStudent()
getStudent()
addDepartament()
removeDepartament()
getDepartament()
getAllDepart()
состоит из
1
1..*
Факультет
name: Name
addInstructor()
removeInstructor()
getInstructor()
getAllInstuctor()
1..*
1..*
работает
1..*
обучается в
1..*
Студент
name: Name
studentID:Number
посещает
*
*
Курс
name: Name
courseID:Number
читает
*
1..*
Преподаватель
name: Name

РИС.3.9

Класс

- Описание классов и отношений между ними является основным средством моделирования структуры в UML.
- Как выделяются классы, подлежащие описанию.
- Нет универсального и применимого во всех случаях способа.
- Три приема выделения классов, самых простых, а потому самых действенных и широко применимых:
  - словарь предметной области (набор основных понятий данной предметной области);
  - реализация вариантов использования;
  - образцы проектирования (стандартное решение типичной задачи в конкретном контексте).

Выводы

- Диаграммы классов моделируют структуру объектов и связей между ними.
- Классы выбираются на основе анализа предметной области, взаимного согласования элементов модели и общих теоретических соображений.