**Лабораторная работа №2.1 “Введение в UML. Диаграмма классов”**

**Цель. Получить практические навыки по созданию диаграмм классов.**

**Основное содержание работы.**

**Краткие теоретические сведения.**

Диаграмма в UML - это графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы рисуют для визуализации. Основная цель диаграмм - визуализация разрабатываемой системы с разных точек зрения. Диаграмма - в самом общем смысле некоторый срез системы. Обычно, за исключением самых простых моделей, диаграммы дают свернутое представление элементов, из которых состоит разрабатываемая система. Один и тот же элемент может присутствовать во всех диаграммах, или только в нескольких (самый часто встречающийся вариант), или не присутствовать ни в одной (очень редко). Согласно теории диаграммы могут содержать любые комбинации сущностей, однако в практике моделирования применяется сравнительно небольшое количество типовых комбинаций, каждая из которых соответствует одному из пяти наиболее необходимых видов, составляющих архитектуру программной системы.

**Диаграммы классов** (class diagram), на которых показывают классы, интерфейсы, объекты и кооперации, а также их отношения. При моделировании объектно-ориентированных систем этот тип диаграмм использует наиболее часто. Диаграммы классов соответствуют статическому виду системы с точки зрения проектирования. Диаграммы классов, которые включают активные классы, соответствуют статическому виду системы с точки зрения процессов.

**Диаграммы объекто*в*** (object diagram), на которых представляются объекты и отношения между ними. Это статические снимки экземпляров сущностей, показанных на диаграммах классов. Диаграммы объектов, как и диаграммы классов, относятся к статическому виду системы сточки зрения проектирования или процессов, но с расчетом на настоящую или макетную реализацию.

**Диаграммы прецедентов** (use case diagram), на которых представлены прецеденты и актеры (частный случай классов), а также отношения между ними. Диаграммы прецедентов относятся к статическому виду системы с точки зрения возможностей ее использования. Это вид диаграмм особенно важен при организации и моделирования поведения системы.

***Диаграммы взаимодействия***, на которых представлены связи между объектами, показаны, в частности, сообщения, которыми объекты могут обмениваться. Обычно рассматриваются два частных случая это вида диаграмм:диаграммы последовательностей (sequence diagram), которые отражают временную упорядоченность сообщений, и диаграммы кооперации (collaboration diagram), на которых показана структурная организация обменивающихся сообщениями объектов. Эти виды диаграмм являются изоморфными, то есть свободно могут быть трансформированы друг в друга.

**Диаграммы состояний** (statechart diagram) представляют автомат, включающий в себя состояния, переходы, события и виды действий. Эти диаграммы относятся к динамическому виду системы, особенно важна их роль при моделировании поведения интерфейса, класса или кооперации. Они заостряют внимание на поведении объекта, которое в свою очередь зависят от последовательности событий, что очень полезно при моделировании реактивных систем.

***Диаграммы деятельности*** (activity diagram) - это частный случай диаграмм состояний. На диаграмме этого типа представляются переходы потока управления от одной деятельности к другой внутри системы. Этот вид диаграмм относится к динамическим представлениям системы, и является наиболее полезным при моделировании ее функционирования, так как отражает передачу потока управления между объектами.

**Диаграммы компонентов** (component diagram), на которых представлена организация совокупности компонентов и существующие между ними зависимости. Диаграммы компонентов относятся к статистическому виду системы с точки зрения реализации. Они могут быть связаны с диаграммами классов в силу очень простой причины: один компонент обычно отображается на один или несколько классов, интерфейсов или коопераций.

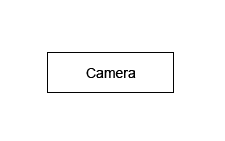
**Диаграммы развертывания** (deployment diagram), на которых представлена конфигурация обрабатывающих узлов системы и размещенных в них компонентов. Диаграммы развертывания относятся к статическому виду системы с точки зрения развертывания. Они связаны с диаграммами компонентов, поскольку в узле обычно размещаются один или несколько компонентов.

**Диаграммы классов UML (Class diagram)**

В UML можно создавать несколько типов диаграмм. В большинстве случаев мы будем пользоваться диаграммами классов (Class diagram). В данном типе диаграмм показывается взаимодействие классов программы.

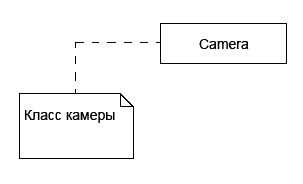
**Элементы диаграмм UML**

Диаграммы UML состоят из элементов. Элементы представляются прямоугольниками, в которых пишется имя элемента. Например, изобразим в UML-диаграмме какой-нибудь класс (для примера я взял, написанный нами ранее, класс Camera):



**Комментарии в UML**

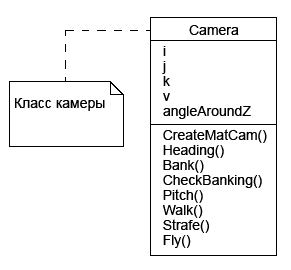
Для комментариев в UML используется прямоугольник с "загнутым" правым верхним уголком. Пунктирной линией показывается, какому элементу принадлежит комментарий:



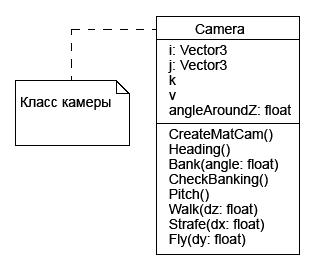
**Атрибуты (attribute) и операции (operation) в UML-диаграммах**

Если в C++ переменная, принадлежащая классу, называется полем класса или переменной-членом, то в UML такая переменная называется атрибутом. Также и с функцией/методом класса - в UML это операция.

Для атрибутов и операций в элементах отводится отдельный блок. Каждый блок разделяется горизонтальной чертой. Например, для класса Camera элемент с атрибутами и операциями будет выглядеть вот так:



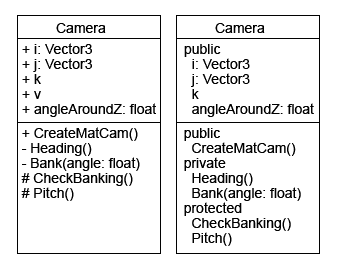
Тип атрибута (как и тип аргумента операции) задаётся через двоеточие:



Здесь можно увидеть все достоинства UML. В Unified Modeling Language необязательно расписывать **все**детали классов. Это будет сделано при написании кода на конкретном языке (в нашем случае - C++). В UML-диаграмме можно опускать ненужные детали. Например, в диаграмму элемента можно добавить только те операции/атрибуты, которые важны для данной диаграммы, неважные особенности класса в UML можно опускать.

**Видимость атрибутов и операций в UML: +, -, # (спецификаторы доступа)**

Спецификатора доступа языка C++ (public, private, protected) в UML отображаются символами + (public), - (private), # (protected), которые ставятся перед именем атрибута/операции. Также возможен вариант с ключевыми словами public, private, protected. На картинке ниже показаны оба варианта:



Напомню значение спецификаторов доступа: public - поля/методы класса видны снаружи класса. Т.е. к ним могут получать доступ объекты класса. private - поля/методы класса видны только внутри определения класса. protected - поля/методы класса видны в определении самого класса и в определениях производных классов.

**Отношения между классами в ООП (UML, С++)**

В программах между классами существуют различные виды взаимодействия (или связи): один класс может быть производным другого, третий может содержать объект четвёртого в виде поля. Для различных видов взаимодействия в UML есть специальные умные названия.

**Ассоциация/объединение/связь (association)**

Первый вид связи - association. На русский можно перевести по-разному: ассоциация, связь, объединение. На мой взгляд, наилучший вариант - связь, но я это слово использую для всех видов взаимодействия классов. Поэтому для обозначения вида связи association мы воспользуемся словом ассоциация.

Ассоциация - самый слабый вид связи. Обычно ассоциация возникает, когда один класс вызывает метод другого или если при вызове метода в качестве аргумента передаётся объект другого класса.

На диаграммах ассоциация обозначается сплошной линией.

Для примера напишем простой класс:

class MonstAr

{

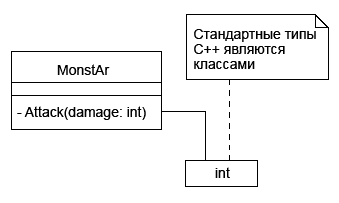
private:

attack(int damage) // damage - урон

{}

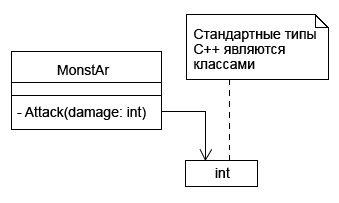
};

Напоминаю, что стандартные типы C++ являются классами. Вот как будет выглядеть взаимодействие классов MonstAr и int на диаграмме UML:



Обратите внимание на то, как в этой диаграмме показано отсутствие атрибутов у элемента.

Иногда при ассоциации показывают направленность (если это имеет значение). В спецификации UML используется слово navigable. На мой взгляд, на русском здесь нужно использовать *направленность*, так как это слово правильно отражает суть. Направленность показывается с помощью стрелочки (обратите внимание, как рисуется стрелочка, это имеет значение):



Заметьте, что стрелочка указывает на int. В данном случае направленность ассоциации говорит нам, что в методе MonstAr::Attack используется объект типа int.

**Обобщение (generalization)**

Для представления наследования в UML используется обобщение (generalization, напоминаю, что все термины берутся из спецификации UML). Пример:

MonstAr

{

private:

attack(int damage) // damage - урон

{}

};

BigMonstAr : public MonstAr // большой (big) MonstAr

{

// определение класса

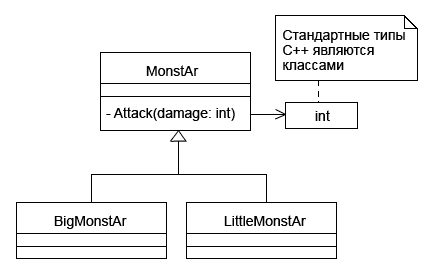
};

SmallMonstAr : public MonstAr // маленький (small) MonstAr

{

// определение класса

};



При обобщении рисуется сплошная линия. Обратите внимание как рисуется стрелочка - пустой треугольник.

Теперь насчёт слова *обобщение* (generalization). В UML используется именно оно, а не *наследование*, так как в данном виде связи один из классов (базовый) является общим, а остальные классы (производные) - более специализированными.

**Aggregation - агрегация, агрегирование, включение в UML**

Следующий тип связи между классами - aggregation (слово происходит от латинского aggregatio - присоединение). По-русски это будет агрегация, агрегирование или соединение частей. Мы будем использовать слово *агрегация*.

Итак, в UML агрегация отражает связь классов, когда объект одного класса является атрибутом другого. Пример:

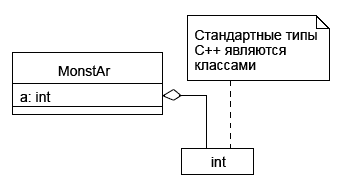
class MonstAr

{

public:

int a;

};



На диаграммах агрегация показывается незакрашенным ромбом.

**Композиция классов - composition в UML**

Композиция классов - более сильная связь между классами, чем агрегация. Между агрегацией и композицией довольно тонкая грань. Особенностью композиции является то, что объекты, из которых создаётся композиция, могут принадлежать только классу, с которым они образуют композицию. При этом время жизни объекта и класса, в который встраивается объект, совпадает.

Для начала рассмотрим два примера из жизни. Например, dvd-привод и диски, которые он читает, образуют агрегацию. Диски можно свободно менять. Примером композиции может служить хлеб и мука. Извлечь муку уже невозможно. На этих двух примерах хорошо видна разница между композицией и агрегацией: компоненты собранные агрегацией можно разъединить, а с композицией этого сделать не получится. Но вернёмся к программированию.

Одним из признаков агрегации является использование указателей. И наоборот, если при связи классов указатели не используются, то существует большая вероятность, что перед нами композиция классов.

class Claws; // claws - когти

class MonstAr

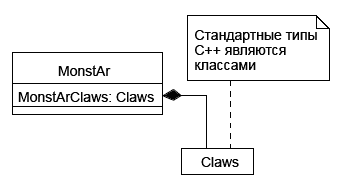
{

public:

Claws MonstArClaws;

};

В данном случае у монстра "есть когти" (определённые в отдельном классе). Возможно, пример не слишком удачный, но здесь хорошо видна композиция классов: нельзя от монстра отделить его когти (он будет сильно недоволен). В UML композиция выглядит вот так:



На диаграммах композиция показывается закрашенным ромбом.

Впереди у нас много примеров, в которых можно будет потренироваться в определении типа связи между классами.

**Реализация - realization в UML**

Последнее отношение, которое мы рассмотрим, будет realization - реализация. Данная связь показывает отношение: класс - объект.

На диаграмме реализация показывается пунктирной линией и незакрашенной стрелочкой:

