Практическое занятие

Построение логической модели АСОИУ. Диаграмма классов

Цель занятия: получить навыки построения диаграмм классов.

Теоретические сведения к практическому занятию

Диаграмма классов - это тип диаграммы UML, который описывает систему путем визуализации различных типов объектов внутри системы и типов статических отношений, существующих между ними. Он также иллюстрирует операции и атрибуты классов.

Обычно диаграммы классов используются для изучения концепций предметной области, понимания требований к программному обеспечению и описания подробных проектов.

Назначение диаграммы классов - моделировать статическое представление приложения. Диаграммы классов - единственные диаграммы, которые могут быть напрямую сопоставлены с объектно-ориентированными языками и, следовательно, широко использоваться во время создания.

Элементами диаграммы классов являются набор классов, интерфейсов, ассоциаций, взаимодействий и ограничений. Она также известна как структурная схема.

Рис. 1. Графические примитивы диаграммы классов UML

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Описание класса состоит в определении атрибутов (свойств) и методов (операций или сервисов).

Интерфейс в языке UML – это семантическая и синтаксическая конструкция, используемая для специфицирования методов класса.

Отношения изображают связь между несколькими вещами, такими как структурные, поведенческие или группирующие вещи на едином языке моделирования. Поскольку он называется связью, он демонстрирует, как вещи взаимосвязаны друг с другом во время выполнения системы. Он представляет собой четыре типа отношений, т. е. Зависимость, ассоциацию, обобщение и реализацию.

Зависимость. Всякий раз, когда происходит изменение в структуре или

поведении класса, которое влияет на другой класс, такое отношение называется зависимостью. Или, проще говоря, мы можем сказать, что класс, содержащийся в другом классе, известен как зависимость. Это однонаправленные отношения.

Ассоциация - это структурное отношение, которое представляет, как два объекта связаны друг с другом в системе. Он может образовывать несколько типов ассоциаций, таких как «один-к-одному», «один-ко-многим», «многие-к-одному» и «многие-ко-многим». Тернарная ассоциация - это ассоциация, состоящая из трех звеньев. Он отображает статические отношения между сущностями двух классов.

Ассоциацию можно разделить на четыре типа ассоциаций, т. е. Двунаправленную, однонаправленную, агрегацию (агрегацию композиции) и рефлексивную, так что агрегация представляет собой особую форму ассоциации, а композиция представляет собой особую форму агрегации. Чаще всего используются однонаправленные и двунаправленные ассоциации.

Агрегация - это особая форма ассоциации. Он изображает частичные отношения. Он образует бинарные отношения, что означает, что он не может включать более двух классов. Это также известно как «Имеющие отношения». Он определяет направление объекта, содержащегося в другом объекте. В совокупности дочерний элемент может существовать независимо от родителя.

В композиционных отношениях ребенок зависит от родителя. Это формирует двусторонние отношения. Это частный случай агрегирования. Это известно как частичные отношения.

Обобщение отношений реализует объектно-ориентированный концепция называется наследование или является разновидностью отношений. Он существует между двумя объектами (вещами или сущностями), так что одна сущность является родительской (суперкласс или базовый класс), а другая - дочерним (подклассом или производным классом). Они представлены с точки зрения наследования. Любой дочерний элемент может получить доступ, обновить или унаследовать функциональность, структуру и поведение родительского элемента.

Реализация - это своего рода отношения, в которых одна вещь определяет поведение или ответственность, которую необходимо выполнять, а другая вещь выполняет это поведение. Он может быть представлен на диаграмме классов или диаграммах компонентов. Отношения реализации устанавливаются между интерфейсами, классами, пакетами и компонентами, чтобы связать клиентский элемент с элементом поставщика.

Пространство имен (namespace) – это именованный элемент модели, который может содержать другие именованные элементы. Принадлежность пространству имен показывается отношением **включения в пространство имен (membership)**. **Полностью квалифицированное имя (fully-qualified name)** элемента в модели состоит из последовательности имен всех вложенных пространств имен, в которые включен элемент.

Классификатор (classifier) – это пространство имен в модели, указывает

на общие некоторому множеству объектов черты. Черты классификатора могут быть поведенческими, структурными или соединительными».

Типом данных (data type) называется классификатор, экземпляры которого не обладают индивидуальностью и, при совпадении значений свойств, взаимозаменяемы (табл. 1).

Таблица 1

Типы данных

Тип данных	Пример	Описание
Примитивный (простой)	«primitive» Простой тип	Примитивный тип служит для указания кратности той или иной сущности. Примитивные типы: • целочисленный тип Integer, • булевский тип Boolean, • строковый тип String, • множество натуральных чисел UnlimitedNatural.
Определенные в языке программирования		
Определенные в модели пользователем	«enumeration» Перечисление «datatype» Тип	В типе данных «enumeration» всем возможным значениям присваиваются имена Тип данных «datatype» определяет тип

Перечисление можно определить так, как показано на рисунке 2.

Рис. 2. Перечислимый тип данных Марка авто

Пример использования стереотипа «dataType» приведен на рисунке 3.

Рис.3. Тип данных Real – действительное число

Основная нотация диаграммы классов представлена на рисунке 4.

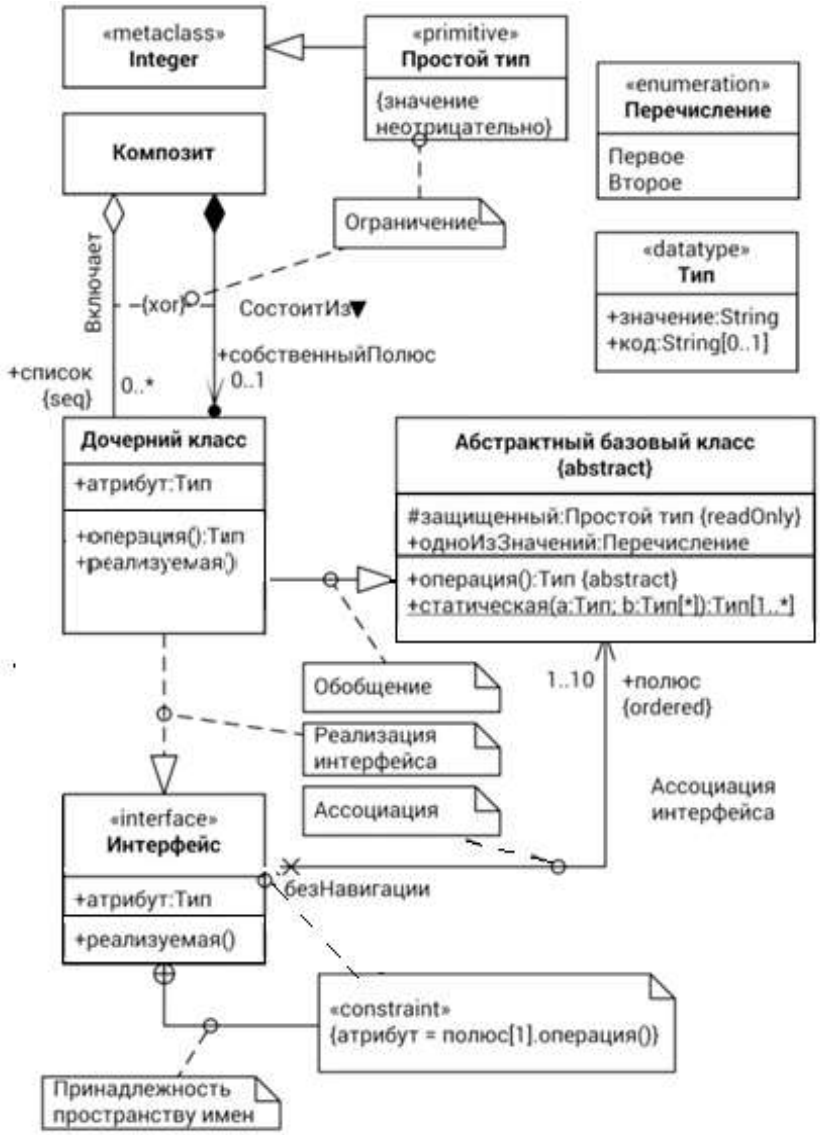

Рис. 4. Основная нотация диаграммы классов

Задачи для самостоятельной работы

6.1. Абстрактный класс *Account* имеет два дочерних класса: счет физического лица *PersonalAccount* и юридического *CompanyAccount*. При решении задачи используйте диаграммы классов.

а. Добавьте класс *Person* с общедоступным атрибутом *FullName* строкового типа и свяжите его с классом *PersonalAccount* ассоциацией *Owns* с полюсом *owner* у *Person* и навигируемым полюсом *account* у *PersonalAccount*.

б. Аналогично для счета юридического лица добавьте владельца *Company*, свяжите анонимной ассоциацией с *CompanyAccount* и укажите подходящие названия полюсов.

в. С учетом компании *Company* и номера социального страхования *SSN* можно найти не более одного сотрудника

г. Добавьте классу *CompanyAccount* общедоступные атрибуты владелец *owner* строкового типа и баланс *balance* типа *Number*. Укажите, что

юридический счет может иметь непустой владелец и баланс должен быть положительным

д. Добавьте класс адреса *Address* с атрибутами строкового типа *street, city* и целочисленным положительным *building*. Укажите с помощью новых анонимных ассоциаций, что *Person* может иметь адрес регистрации *registeredAt*, фактический адрес *actual*, в то время как компания связана с одним юридическим адресом *legalAddress* и может иметь почтовый адрес *postAddress*.

6.2. В пространстве имен *Time* расположены перечисления *Month*, *DayOfWeek* , а также классы *Date* и *Period*. При решении задачи используйте диаграммы классов.

а. Укажите, что перечисление *Month* может принимать значения: *Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec*.

б. Укажите, что перечисление *DayOfWeek* может принимать значения: *Mon, Tue, Wed, Thu, Fri, Sat, Sun*.

в. Добавьте классу *Date* частные атрибуты *year, month, dayOfMonth* типа *Integer*, а также общедоступные операции: получения года *getYear* типа *Integer*; получения месяца *getMonth* типа *Month*; получения дня *getDayOfMonth* типа *Integer*; получения дня недели *getDayOfWeek* типа *DayOfWeek*.

г. Добавьте классу *Date* общедоступную статическую операцию *now ()* типа *Date*.

д. Добавьте классу *Period* общедоступную статическую операцию *between*. У операции два аргумента: *from* и *to*. Оба аргумента имеют тип *Date*. Операция возвращает значение типа *Period*.

е. Добавьте классу *Date* операцию *plus* с аргументом *delta* типа *Period*. Результат операции – значение типа *Date*.

6.3. Класс *MyWindow* уточняет абстрактный базовый класс *Window*. *MyWindow* состоит (композиция) из кнопки класса *Button* и надписи класса *Label*. Отобразите на диаграмме классов.

а. Класс *Label* имеет закрытый атрибут *text* типа *String* и общедоступную операцию *setText* с параметром *text* типа *String* .

б. Композиция между *MyWindow* и *Button* называется *HoldsButton*. Полюс со стороны кнопки имеет имя *okButton*, защищенную видимость, кратность 1. Композиция между *MyWindow* и *Label* называется *HoldsLabel*. Украшения полюса со стороны *Label*: название *textLabel*, частная видимость, кратность 1.

в. Класс *MyWindow* реализует интерфейс *IClickListener* для реакции на нажатие кнопки. Отобразите на диаграмме, что между классом *Button* и *MyWindow* есть ассоциация с именем *NotifyListener* с направлением от кнопки к окну. Укажите, что полюс со стороны окна называется *listener*, имеет тип *IClickListener*, множественную кратность и закрытую видимость.

6.4. Интерфейс доступа к коллекции элементов *Collection* обобщает интерфейс работы со списками *List*. Абстрактный класс *BaseCollection* реализует интерфейс *Collection*, абстрактный класс *BaseList* является потомком *BaseCollection* и реализует интерфейс *List*, оставляя операции по

хранению данных дочерним классам.

а. Используя наследование, добавьте в модель класс *ArrayList*, реализующий операции со списками с помощью массива.

б. Пусть интерфейс *List* содержит операцию *get* получения элемента списка по заданной позиции *k*. Укажите, в каких классах должна быть объявлена данная операция, чтобы модель была согласованной.

в. Пусть интерфейс *Collection* содержит операцию *add* добавления элемента *obj*. Укажите, в пространстве имен каких классов может присутствовать поведение, реализующее операцию *add*. Ответ поясните.

г. Добавьте в модель класс *BaseDisk*, который уточняет класс *BaseList*. Диск *BaseDisk* содержит несколько папок *BaseFolder*, которые могут содержать файлы *File* и папки. Произведения *Composition* хранятся на дисках в виде файлов. Произведение может быть картинкой *Picture*, либо музыкой *Music*, либо фильмом *Movie.*

6.5. Создайте абстрактный класс *AbstractSpace*, реализующий интерфейс *Space*. Классы *RentedSpace* и *OwnedSpace* должны наследоваться от класса *AbstractSpace*.

Добавьте классу *AbstractSpace* общедоступную статическую операцию *period*. У операции два аргумента: *from* и *to*. Оба аргумента имеют тип *Date*. Операция возвращает значение типа *Period*. Добавьте классу частные атрибуты *person* типа *Person, vehicle* типа *Vehicle*, статический атрибут *empty* типа *LocalDate*, предназначенный только для чтения.

Классы *RentedSpace* и *OwnedSpace* состоят из классов *Person* – владелец ТС (транспортное средство) и *Vehicle* – ТС.

Класс *OwnedSpace* включает в себя класс *Position*, который можно определить как упорядоченное множество точек (класс *Point*), который не может меняться.

Отобразите на диаграмме, что *Vehicle* исполняет *Order* с классом ассоциации *FormEntryOrder*.

Ассоциация между классами *Order* и *OrderLine* использует квалификатор. Квалификатор указывает, что в соответствии с заказом для каждого экземпляра продукта (*Product*) может существовать только одна строка заказа. Полюс со стороны *OrderLine* называется *LineItems*.

Класс *Person* характеризуется частным атрибутом *name* (строка) для хранения 3 составляющих. Имеет производный атрибут *age*. Общедоступная константа класса: *unknow* типа *Person* с начальным значением пустая строка. У *Person* нет много *Purchase*. *Purchase* делаются в определенном порядке, и каждая из них уникальна. Для работы класса *Person* нужен интерфейс *Space*.

Создайте перечисление *VehicleTypes* – типа ТС. Константы перечисления: *NONE CAR CROSSOVER MOTOR_BIKE SUV TRUCK OTHER*.