Министерство науки и высшего образования Российской Федерации

Московский государственный университет геодезии и картографии (МИИГАиК)

Факультет геоинформатики и информационной безопасности

**Отчёт**

по дисциплине «Моделирование процессов и систем»

**Тема:**

«Составление диаграмм классов для геопортала Крыма»

Выполнил: студент

data

data

Проверил:

data

**Москва –** data

**Назначение и состав диаграммы классов:**

Состав **диаграммы классов** аналогичен составу диаграммы классов анализа. В то же время классы анализа должны пройти процедуру строгой экспертизы на предмет их возможной декомпозиции на более мелкие и специализированные классы. При построении диаграммы окончательно должны быть определены атрибуты и операции классов.

**Графически класс** отображается в виде прямоугольника, который может быть разделен горизонтальными линиями на секции. В этих секциях указывается имя, атрибуты (свойства) и операции (методы).

|  |  |
| --- | --- |
|  | **Графически класс** |
|  | **Графически класс** |
|  | **Графически класс** |

**Секция атрибутов** выделяется горизонтальной линией, даже если у класса отсутствуют атрибуты (характерно для классов-интерфейсов).

С точки зрения структурного подхода, **атрибуты** – это переменные, а **методы** – это функции, описанные в теле класса. Они могут быть доступны или не доступны для изменения (атрибуты) или выполнения (методы) внешними объектами.

Обязательным элементом обозначения класса на диаграмме является его имя. Оно должно быть уникальным в пределах пакета. Если класс является абстрактным, то его имя пишется курсивом. **Абстрактный класс** – это класс, на основе которого нельзя создать объекты. Такие классы используются в качестве шаблона для дочерних классов при наследовании.

В секции имени класса может быть указан стереотип (например, "entity", "boundary", "interface" и т. п.).

Во второй секции каждому **атрибуту** соответствует отдельная строка со следующей спецификацией:

[видимость] [/] имя [: тип [‘[‘кратность‘]‘] [ = исходное значение]] [‘{‘модификаторы’}’].

Квадратные скобки означают, что соответствующий элемент спецификации может отсутствовать. Таким образом, при описании обязательным является только имя атрибута.

**Видимость** (англ. visibility) характеризует возможность чтения и модификации значения атрибута объекта описываемого класса, из объектов других классов. Модификация значения возможна лишь при условии, что атрибут не является константой. Видимость отображается с помощью следующих символов:

* "+" – общедоступный атрибут (англ. public) – доступен для чтения и модификации из объектов любого класса;
* "#" – защищенный атрибут (англ. protected) – доступен только объектам описываемого класса и его потомкам при наследовании;
* "–" – закрытый атрибут (англ. private) – доступен только объектам описываемого класса;
* "~" – пакетный атрибут (англ. package) – доступен только объектам классов, входящих в тот же пакет.

**Символ "/"** перед именем атрибута указывает на то, что он является производным (т.е. его значение вычисляется из значений других атрибутов или ассоциаций).

**Имя** (англ. name) атрибута представляет собой строку текста, которая используется для его идентификации. Оно должно быть уникальным в пределах класса.

**Тип** (англ. type) атрибута выбирается исходя из семантики значений, которые должны храниться в атрибуте, и, как правило, возможностей целевого языка программирования по представлению этих значений. Он соответствует одному из стандартных типов, определенных в этом языке (например, String, Boolean, Integer, Color и т. д.) или имени класса, на объект которого в этом атрибуте будет храниться ссылка. Во втором случае класс, имя которого указано в качестве типа, должен быть определен на диаграмме или в модели.

**Кратность** (англ. multiplicity) атрибута характеризует количество значений, которые можно хранить в атрибуте. Если кратность атрибута не указана, то по умолчанию принимается ее значение, равное 1, т. е. атрибут является атомарным. Такой вариант допускает и отсутствие значения в атрибуте (null). Для атрибута, представляющего собой массив, множество, список и т. п., требуется указание кратности, которая записывается после типа в квадратных скобках. Варианты указания кратности, имеющие смысл, могут быть следующие:

* [0..\*] или [\*] – количество хранимых значений может принимать любое положительное целое число, большее или равное 0. Такой вариант задания кратности характерен для множеств, списков и других атрибутов, допускающих добавление или удаление элементов;
* [0..<число>] – количество хранимых значений, может быть не более указанного числа. Данный вариант применяется при описании массивов фиксированного размера. При этом не обязательно, чтобы все элементы массива имели конкретные значения;
* [0..<число>] [0..<число>] – применяется при описании двумерных массивов. Аналогичным образом можно описать трехмерные, четырехмерные и т.д. массивы.

**Исходное значение** (англ. default value) служит для задания некоторого начального значения атрибута в момент создания отдельного экземпляра класса (объекта).

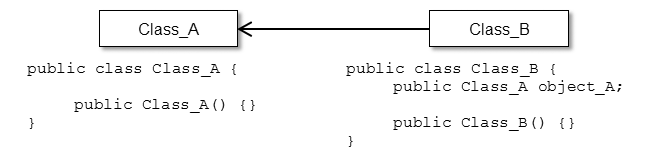
**Mодификатор** (англ. modifier) описывает особенности реализации атрибута, например:

* {final} / {readOnly} – атрибут является константой, т.е. доступен только для чтения;
* {static} – атрибут при выполнении программы в конкретный момент времени будет иметь одно и то же значение для всех объектов класса;
* {transient} – атрибут и его значение при записи объекта в БД или файл (сериализации объекта) не должны запоминаться;
* {redefines <имя атрибута родительского класса>} – атрибут переопределяет (заменяет) атрибут родительского класса;
* {id} – значение атрибута используется в качестве идентификатора объекта класса;
* {unique} или {nonunique} – значения неатомарного атрибута должны быть уникальны или допускаются повторы значений;
* {ordered} или {unordered} – значения неатомарного атрибута должны быть отсортированы или могут содержаться в произвольном порядке;
* {seq} / {sequence} – значения неатомарного атрибута хранятся упорядочено (к ним можно обращаться по индексу или выполнять перебор в соответствии с порядком их добавления в список/массив/множество) и могут повторяться.

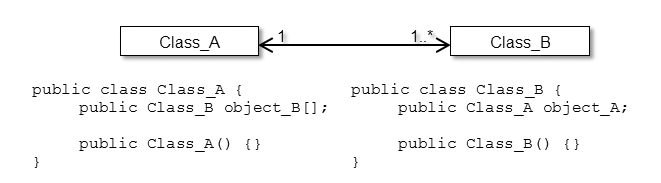
Допускается указывать несколько модификаторов через запятую. Например, {unique, ordered} означает, что элементы массива должны быть уникальны и следовать в строго определенном порядке (например, по возрастанию значений).

**Отношения**, которые можно устанавливать между классами, и их смысловая нагрузка (семантика) были рассмотрены в подразделе по диаграммам классов анализа. Далее иллюстрируется связь между графическим отображением классов и отношений на диаграммах и исходными текстами программ. Современные Case-средства при разработке классов, как правило, работают в режиме синхронизации диаграмм и исходных текстов. Т. е. если меняется диаграмма классов, то это приводит к автоматической корректировке текста программы и наоборот.

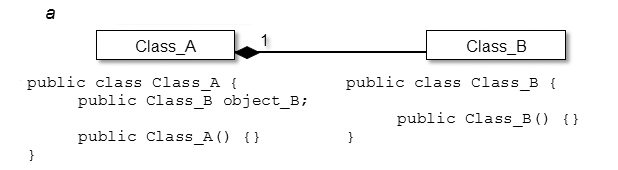
**Отношение ассоциации** означает наличие атрибута, в котором будет храниться ссылка (ссылки) на объект (объекты) класса, в сторону которого направлена стрелка ассоциации.

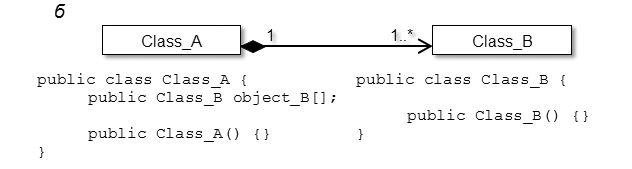


Графический символ класса Class\_A преобразуется в строки определения самого класса "public class Class\_A" и его конструктора "public Class\_A() {}". Аналогично для Class\_B. Ассоциация от Class\_B в сторону Class\_A преобразуется в строку "public Class\_A object\_A;", описывающую атрибут object\_A, в котором будет храниться ссылка на объект класса Class\_A. Ввиду отсутствия указания кратности отношения, она по умолчанию принимается равной 1. На следуюшей рисунке приведен пример двунаправленной ассоциации кратностью более 1.

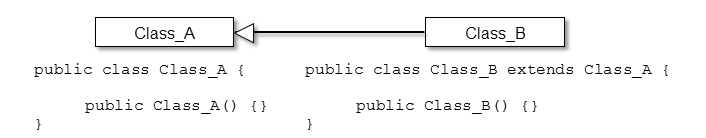


**Отношения агрегации** и **композиции** являются частными случаями ассоциации. В связи с этим интерпретация этих отношений с точки зрения текста программы совпадает с рассмотренной выше. В "объекте–целом", даже при отсутствии стрелки на стороне "объекта–части", будет храниться ссылка на него.

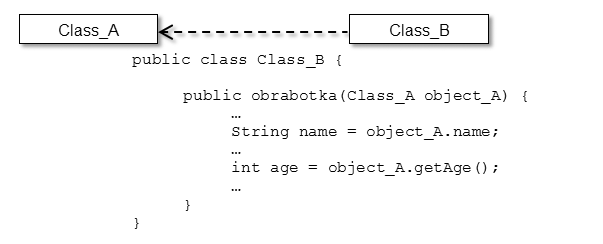




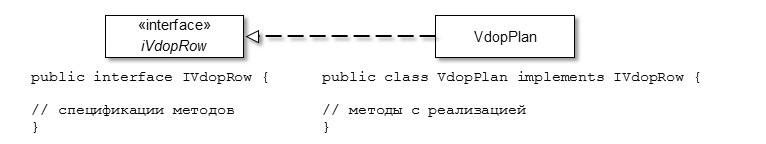
**Отношение обобщения** в тексте программы на языке Java показывается ключевым словом "extends" (англ. – расширяет) в дочернем классе.



Отношение зависимости не приводит к автоматической генерации кода программы, но свидетельствует об обращении из объекта зависимого класса к атрибутам, методам или непосредственно к объектам независимого класса. Данное отношение в Case-средстве может автоматически отображаться на диаграмме при обратном проектировании или при синхронизации диаграммы и текста программы.

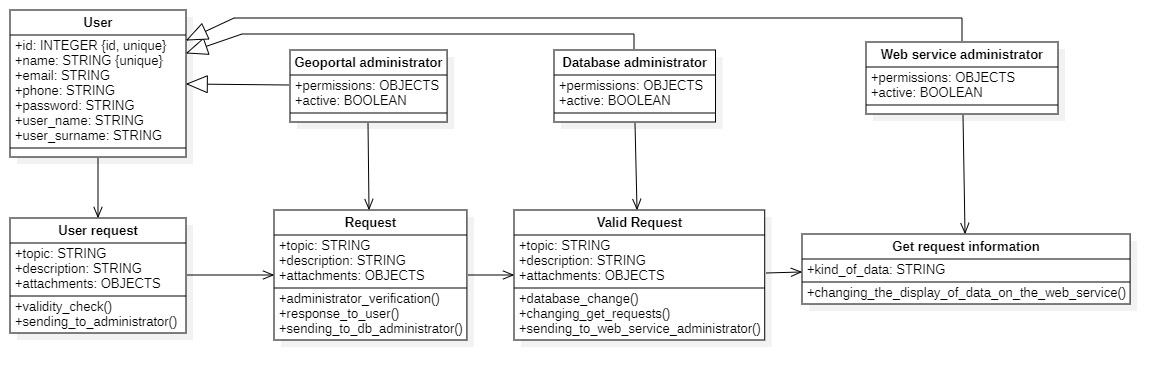


**Отношение реализации** - дополнительное отношение на диаграмме классов по сравнению с диаграммой классов анализа, которое отображается только между классами и интерфейсами. В тексте на языке Java данное отношение обозначается ключевым словом "implements" (англ. – реализует).



**Основная часть:**

На следующем рисунке показан фрагмент диаграммы классов, демонстрирующий процесс добавления заявки пользователем на обновление/добавление новой информации на геопортале.



Всё начинается с модели пользователя (User). Она имеет следующие параметры:

* id – уникальное ид для пользователя;
* name – уникальное имя для пользователя;
* email – email-почта пользователя;
* phone – телефон пользователя;
* password – зашифрованный пароль пользователя;
* user\_name – имя пользователя;
* user\_surname – фамилия пользователя.

Классы Geoportal administrator, Database administrator, Web service administrator расширяют встроенную модель пользователя и добавляют в свои модели следующие параметры:

* permissions – права для администраторов;
* active – флаг активации аккаунта администратора.

Сам пользователь можем создать заявку на обновление/добавление какой-либо новой информации для веб сервиса. Класс User request описывает это действие:

* topic – тема заявки;
* description – описание заявки;
* attachments – прикрепленный файл.

Функция validity\_check() отвечает за валидацию заявки, а функция sending\_to\_administrator() – за отправку действительной заявки администратору геопортала.

Далее переходим к следующему классу – Request. В нём есть всё те же параметры, что и в предыдущей модели, однако функции разные: administrator\_verification() – проверка администратором информации из заявки, response\_to\_user() – ответ пользователю о статусе заявки (успешно/неуспешно), sending\_to\_db\_administrator() – в случае успешной валидации, отправка информации администратору БД.

Valid Request – аналогично с прошлой моделью. Функции database\_change(), changing\_get\_requests(), sending\_to\_web\_service\_administrator(), отвечают за изменение БД, изменение гет-запросов, отправка информации об изменениях администратору веб сервиса, соответственно.

И последний класс - Get request information. В нем есть параметр kind\_of\_data, который описывает изменения, сделанные администратором базы данных и функция changing\_the\_display\_of\_data\_on\_the\_web\_service(), которая отвечает за изменение вида отображаемой информации веб сервиса.

**Источники:**

**1)** Учебная и научная деятельность Анисимова Владимира Викторовича –Диаграмма классов // URL: <https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema14/tema14_2> (Дата обращения: data)