Лабораторная работа №5

МОДЕЛИРОВАНИЕ НА ЭТАПЕ ПРОЕКТИРОВАНИЯ ИС:
МОДЕЛИРОВАНИЕ СТРУКТУРЫ СИСТЕМЫ

Цель работы:

- изучение объектно-ориентированного моделирования и исследование процесса построения диаграммы классов в заданной предметной области.

- изучение основных элементов диаграммы классов, создание диаграммы классов, а также получение навыков работы с инструментальными средствами создания диаграмм классов.

Задание.

1. Используя требования к ИС и модели ИС (диаграммы), разработанные в лабораторных работах №3-4, выделить классы и отношения между ними.

2. Используя выбранное CASE-средство для построения диаграмм UML, разработать диаграмму классов и сгенерировать программный код на языке C++ или Java.

ПОРЯДОК ВЫПОЛНЕНИЯ

1. Ознакомиться с краткими теоретическими сведениями.
2. Выделить классы на основе требований к ИС и ранее разработанных диаграмм. Выявить активные и абстрактные классы.
3. Построить диаграмму классов (минимум 8–10 классов), моделирующую структуру реализуемой системы на этапе проектирования.
4. Использовать различные типы отношений между классами (ассоциация, агрегация, композиция, обобщение, зависимости). Отобразить роли и кратности ассоциаций, абстрактный класс, операции, области видимости, вычислимые атрибуты, направления навигации, зависимости.
5. Выбрать язык объектно-ориентированного программирования для прямого проектирования (C++ или Java). Использовать в диаграмме классов типы данных выбранного языка.
6. Дать комментарии к диаграмме классов в виде краткого описания классов, их атрибутов и операций. Аргументировать выбор структур данных.
7. Сгенерировать программный код на языке C++ или Java, описать последовательность его получения. Приложить листинг, полученного кода в отчет.
8. Оформить отчет
9. Продемонстрировать работу преподавателю.
10. Ответить на контрольные вопросы.

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Диаграммы классов

Диаграмма классов – это графическое представление статической модели, в которой собраны декларативные (статические) элементы, такие как классы, типы, а также их содержимое и отношения. Содержит некоторые конкретные элементы поведения (например, операции), однако их динамика в этой диаграмме не отображается.

Обычно для описания системы создают несколько диаграмм классов. На одних показывают некоторое подмножество классов и отношения между классами подмножества. На других отображают то же подмножество, но вместе с атрибутами и операциями классов. Третьи соответствуют только пакетам классов и отношениям между ними. Для представления полной картины системы можно разработать столько диаграмм классов, сколько требуется.

По умолчанию существует одна диаграмма классов, называемая Главной (Main). На этой диаграмме показывают пакеты классов модели. Внутри каждого пакета также имеется Главная диаграмма, включающая в себя все классы этого пакета. На этой диаграмме показывают пакеты классов модели. Внутри каждого пакета также имеется Главная диаграмма, включающая в себя все классы этого пакета.

Выявление классов можно начать с изучения потока событий сценария. Имена существительные в описании этого потока дадут понять, что может являться классом.

Каждый объект диаграмм последовательности и кооперативных диаграмм должен быть соотнесен с соответствующим классом.

Поток событий и диаграммы взаимодействия являются прекрасным местом, где можно начать поиск классов.

Атрибут - это значение, характеризующее объект в его классе. Примеры атрибутов: категория, баланс, кредит (атрибуты объектов класса счет); имя, возраст, вес (атрибуты объектов класса человек) и т.д.

Среди атрибутов различаются постоянные атрибуты (константы) и переменные атрибуты. Постоянные атрибуты характеризуют объект в его классе (например, номер счета, категория, имя человека и т.п.). Текущие значения переменных атрибутов характеризуют текущее состояние объекта (например, баланс счета, возраст человека и т.п.); изменяя значения этих атрибутов, мы изменяем состояние объекта.

Атрибуты перечисляются во второй части прямоугольника, изображающего класс (см. рис.1).

Иногда указывается тип атрибутов (ведь каждый атрибут - это некоторое значение) и начальное значение переменных атрибутов (совокупность начальных значений этих атрибутов задает начальное состояние объекта).

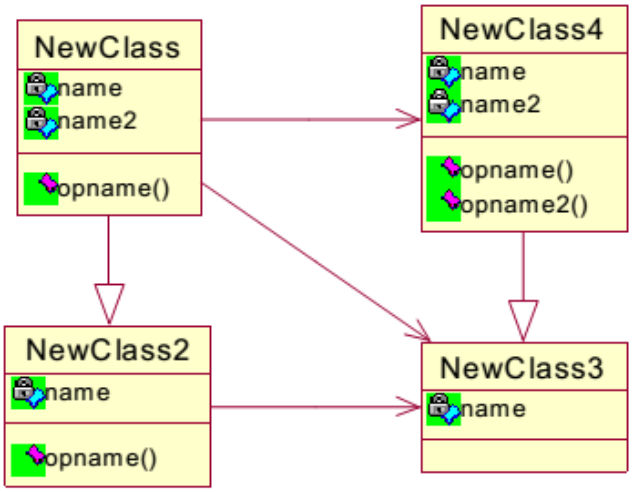

Рис.1 - Диаграмма классов

Операция - это функция (или преобразование), которую можно применять к объектам данного класса. Одна и та же операция может, вообще говоря, применяться к объектам разных классов: такая операция называется полиморфной, так как она может иметь разные формы для разных классов.

Каждой операции соответствует метод - реализация этой операции для объектов данного класса. Таким образом, операция- это спецификация метода, метод - реализация операции. Например, в классе файл может быть определена операция печать (print). Эта операция может быть реализована разными методами: (а) печать двоичного файла; (б) печать текстового файла и др. Логически эти методы выполняют одну и ту же операцию, хотя реализуются они разными фрагментами кода.

Каждая операция имеет один неявный аргумент - объект к которому она применяется. Кроме того, операция может иметь и другие аргументы (параметры). Эти дополнительные аргументы параметризуют операцию, но не связаны с выбором метода. Метод связан только с классом и объектом (некоторые объектно-ориентированные языки, например C++, допускают одну и ту же операцию с разным числом аргументов, причем используя то или иное число аргументов, мы практически выбираем один из методов, связанных с такой операцией; на этапе предварительного проектирования системы удобнее считать эти операции различными, давая им разные имена, чтобы не усложнять проектирование).

Операция (и реализующие ее методы) определяется своей сигнатурой, которая включает, помимо имени операции, типы (классы) всех ее аргументов и тип (класс) результата (возвращаемого значения). Все методы, реализующие операцию должны иметь такую же сигнатуру, что и реализуемая ими операция.

Операции перечисляются в третьей части прямоугольника (рис.1), описывающего класс. Каждая операция должна быть представлена своей сигнатурой, однако на ранних стадиях проектирования можно ограничиваться указанием имени операции, отложив полное определение сигнатуры на конец рассматриваемой фазы жизненного цикла (либо даже на последующие фазы). В

графическом языке технологии UML тип любого объекта данных указывается вслед за именем этого объекта после двоеточия.

При моделировании системы полезно различать операции, имеющие побочные эффекты (эти эффекты выражаются в изменении значений атрибутов объекта, т.е. в изменении его состояния), и операции, которые выдают требуемое значение, не меняя состояния объекта. Эти последние операции называются запросами.

Таким образом, для задания класса необходимо указать имя этого класса, а затем перечислить его атрибуты и операции (или методы). Полное описание класса на графическом языке UML имеет вид, изображенный на рис. 2. Однако иногда удобно бывает пользоваться сокращенным описанием класса, когда в прямоугольнике, изображающем этот класс, указывается только имя класса.

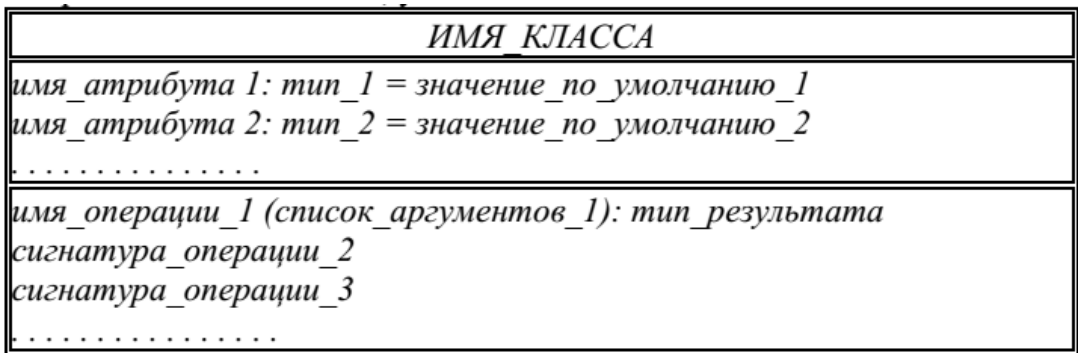

Рис. 2 - Полное описание класса в нотации UML

Параметризованный класс (parameterized class) — один из специальных типов классов. Он применяется для создания семейства других классов. Обычно параметризованный класс является разновидностью контейнера, его еще называют шаблоном.

На языке UML параметризованный класс изображается с помощью следующей нотации (рис.3):

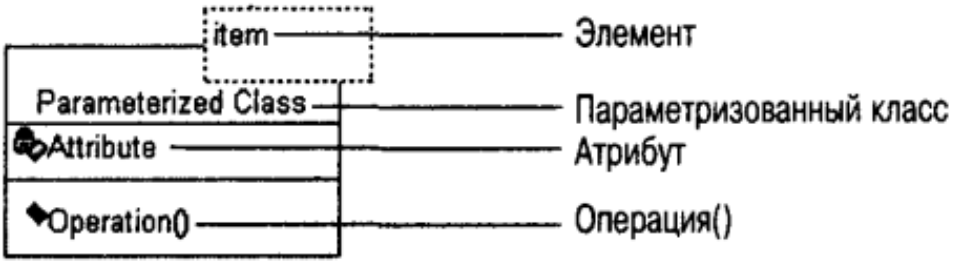

Рис. 3 – Нотация UML для изображения параметризованного класса

Класс-наполнитель (instantiated class) является параметризованным классом, аргументы которого имеют фактические значения. В соответствии с нотацией UML, имя аргумента класса-наполнителя заключается в угловые скобки (< >).

Утилита класса (class utility) - это совокупность операций. Например, это может быть совокупность математических функций, которые используются всей

системой и не слишком хорошо подходят для какого-либо конкретного класса. Эти функции можно собрать вместе и объединить в утилиту класса, которая будет использоваться другими классами системы.

Утилиты классов часто применяют для расширения функциональных возможностей языка программирования или для хранения общих элементов функциональности многократного использования, необходимых в нескольких системах.

Утилита класса выглядит на диаграмме как класс "с тенью" (рис.4):

Рис. 4 - Утилита класса в нотации UML

Утилитой параметризованного класса (parameterized class utility) является параметризованный класс, содержащий только набор операций. Это шаблон для создания утилит класса. На диаграмме классов она выглядит следующим образом (рис.5):

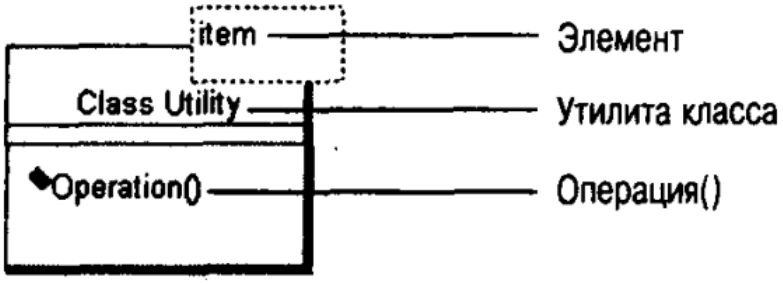

Рис. 5 - Утилита параметризованного класса в нотации UML

Метакласс (metaclass) - это класс, экземпляры которого являются классами, а не объектами. К числу метаклассов относятся параметризованные классы и утилиты параметризованных классов. На языке UML метаклассы изображают следующим образом (рис.6):

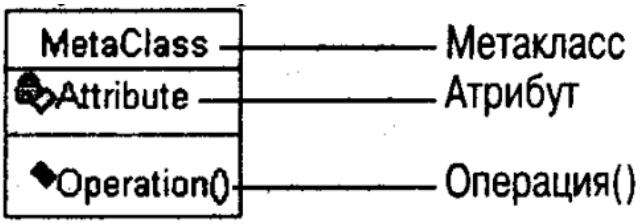

Рис. 6 - Метакласс в нотации UML

Каждому классу модели необходимо дать уникальное имя. В общем случае используются существительные в единственном числе.

Обычно имена классов не содержат пробелов. Это делается по практическим причинам и из соображений удобочитаемости - языки программирования, как правило, не поддерживают пробелы именах классов.

Стереотип - это механизм, позволяющий категоризировать классы. Допустим, вы хотите найти все формы в вашей модели. Для этого можно создать стереотип Form (Форма) и назначить его всем окнам вашего приложения. В дальнейшем при поиске форм нужно только искать классы с этим стереотипом. На языке UML определены три основных стереотипа: Boundary (Граница), Entity (Объект) и Control (Управление).

Пограничными классами (boundary classes) называются такие классы, которые расположены на границе системы со всем остальным миром. Они включают в себя формы, отчеты, интерфейсы с аппаратурой такой, как принтеры или сканеры) и интерфейсы с другими системами (рис.7).

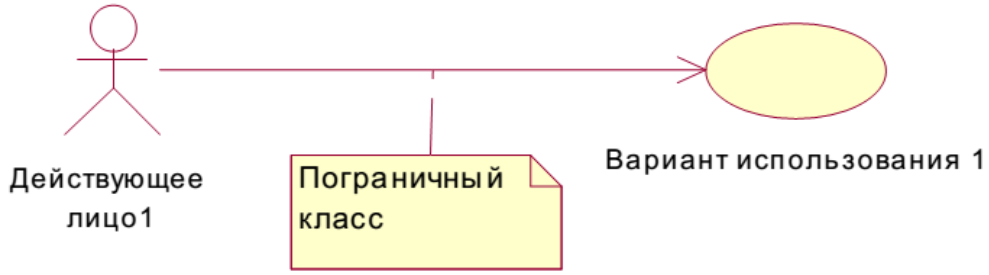

Рис. 7 - Пограничный класс

Для выявления пограничных классов необходимо исследовать диаграммы Вариантов Использования (диаграммы прецедентов).

Для каждого взаимодействия между действующим лицом и вариантом использования должен существовать хотя бы один пограничный класс. Именно он позволяет действующему лицу взаимодействовать с системой.

Классы-сущности (entity classes) содержат информацию, хранимую постоянно. Классы-сущности можно обнаружить в потоке событий и на диаграммах Взаимодействия. Они имеют наибольшее значение для пользователя, и потому в их названиях часто применяют термины из предметной области.

Часто для каждого класса-сущности создают таблицу в базе данных. В этом заключается еще одно отличие от типичного подхода к конструированию системы. Вместо того чтобы с самого начала задавать структуру базы данных, у нас теперь есть возможность разрабатывать ее на основе информации получаемой из объектной модели. Это позволяет отслеживать соответствие всех полей базы данных и требований к системе.

Активный класс – это класс, объекты которого являются владельцами одного или нескольких процессов или потоков и могут инициировать управляющие воздействия. Поведение объектов такого класса осуществляется параллельно с поведением других элементов ИС.

Изображается активный класс прямоугольником с двойными боковыми линиями.

Прямое проектирование – это создание программного кода из модели. При создании диаграммы классов учитывается синтаксис целевого языка ООП, в который может быть выполнена с помощью CASE-средств генерация программы.

Требования определяют поток событий. Поток событий определяет классы и атрибуты классов. Каждый атрибут класса-сущности становится полем в базе данных. Применяя такой подход, легко отслеживать соответствие между полями базы данных и требованиями к системе, что уменьшает вероятность сбора ненужной информации.

Управляющие классы (control classes) отвечают за координацию действий других классов. Обычно у каждого варианта использования имеется один управляющий класс, контролирующий последовательность событий этого варианта использования.

Сам управляющий класс не несет в себе никакой функциональности - остальные классы посылают ему мало сообщений. Но сам он посылает множество сообщений. Управляющий класс делегирует ответственность другим классам. По этой причине управляющий класс часто называют классом-менеджером.

Множественность управляющего класса обычно равна 1.

Доступны следующие значения множественности:

− N – много (по умолчанию)

− 0..0 – нуль

− 0..1 – нуль или один

− 0..n – нуль или больше

− 1..1 – ровно один

− 1..n – один или больше

Можно вводить множественность самим:

− <число> - ровно число

− <число1>..<число2> - между числом1 и числом 2

− <число>..n – число или больше

− <число1>,<число2> - число1 или число 2

− <число1>,<число2>..<число3> - ровно число 1 или между числом 2 и 3

− <число1>..<число2>,<число3>..<число4> - между 1 и 2 или между 3 и 4

Абстрактным называется класс, который не наполняется конкретным содержанием. Иными словами, если класс А абстрактный, в памяти никогда не будет объектов типа А.

Обычно абстрактные классы применяют при работе с наследованием. В них содержатся данные и поведение, общие для нескольких других классов.

Во вложенные (nested) классы можно вкладывать другие классы, организуя столько уровней вложения, сколько необходимо.

Пакеты (packages) применяются для группирования классов, обладающих некоторой общностью.

Объединять классы можно, как угодно, однако существует несколько наиболее распространенных подходов. Во-первых, можно группировать классы по стереотипу. В таком случае получается один пакет с классами-сущностями, один с пограничными классами, один с управляющими классами и т.д.

Второй подход заключается в объединении классов по их функциональности.

Наконец, применяют комбинацию двух указанных подходов.

Поведение системы обеспечивается взаимодействием объектов. Два типа отношений, которые можно выделить на этапе анализа – это ассоциация и агрегация.

Ассоциация (association) – это двунаправленная семантическая связь между классами (рис.8).

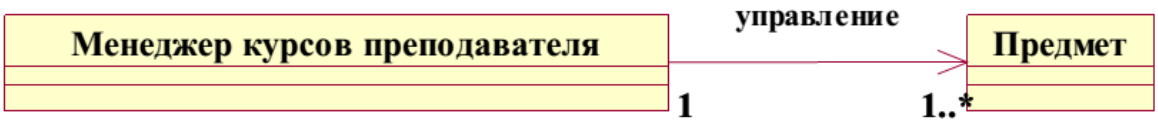

Рис.8 - Связь «Ассоциация»

Агрегационное отношение – это специальная форма ассоциации между целым и его частью или частями, то есть отношение типа «часть от» или «содержит» (рис.9).

Рис.9 - Связь «Агрегация»

Мощность отношений (multiplicity) указывается для классов и определяет допустимое количество объектов, участвующих в отношении с каждой стороны.

Несколько объектов, принадлежащих одному классу, могут взаимодействовать друг с другом. Такое взаимодействие показывается на диаграмме классов как возвратное.

Наследованием называется такое отношение между классами, когда один класс использует часть структуры и/ или поведения другого или нескольких классов. При наследовании создается иерархия абстракций, в которой подкласс (subclass) наследуется от одного или нескольких суперклассов (superclass).

Подкласс наследует все атрибуты, операции и отношения, определенные в каждом его суперклассе. Такой тип связи называется обобщением. (рис.10).

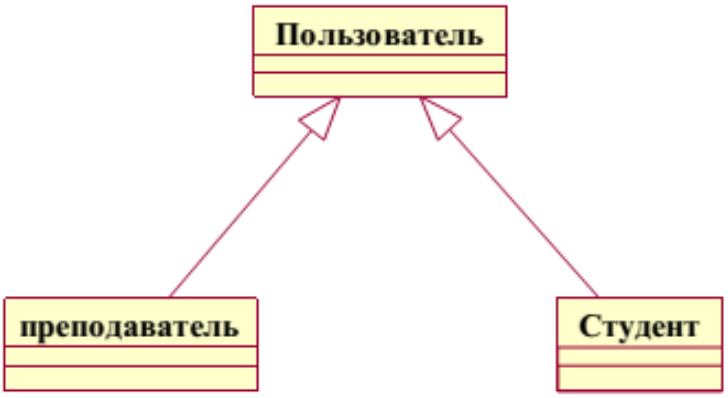

Рис.10 - Связь «Обобщение»

Диаграммы Классов являются хорошим инструментом проектирования. С их помощью разработчики могут видеть и планировать структуру системы еще до фактического написания кода, благодаря чему они с самого начала могут понять, хорошо ли спроектирована система.

Работа с CASE – средствами кодирования программного обеспечения

Одно из свойств **StarUML** – возможность генерации программного кода, представляющего модель. Пакет способен выполнять кодогенерацию на языках C++, C#, Java и др.

Подготовка к генерации программного кода

Чтобы получить возможность генерации программного кода в нужном языке программирования, необходимо установить соответствующий плагин:

1. Во вкладке *Tools* выбрать *Extension Manager*… (рис.11)

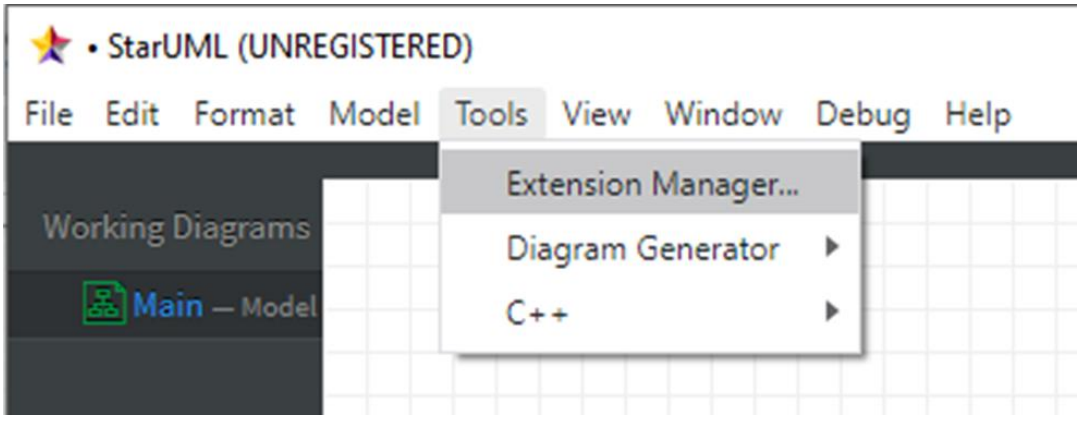

Рис.11 – Меню установки плагина

2. Из списка предложенных плагинов выбрать, например, Java или C++ , и нажать кнопу *Install.* (рис. 12)

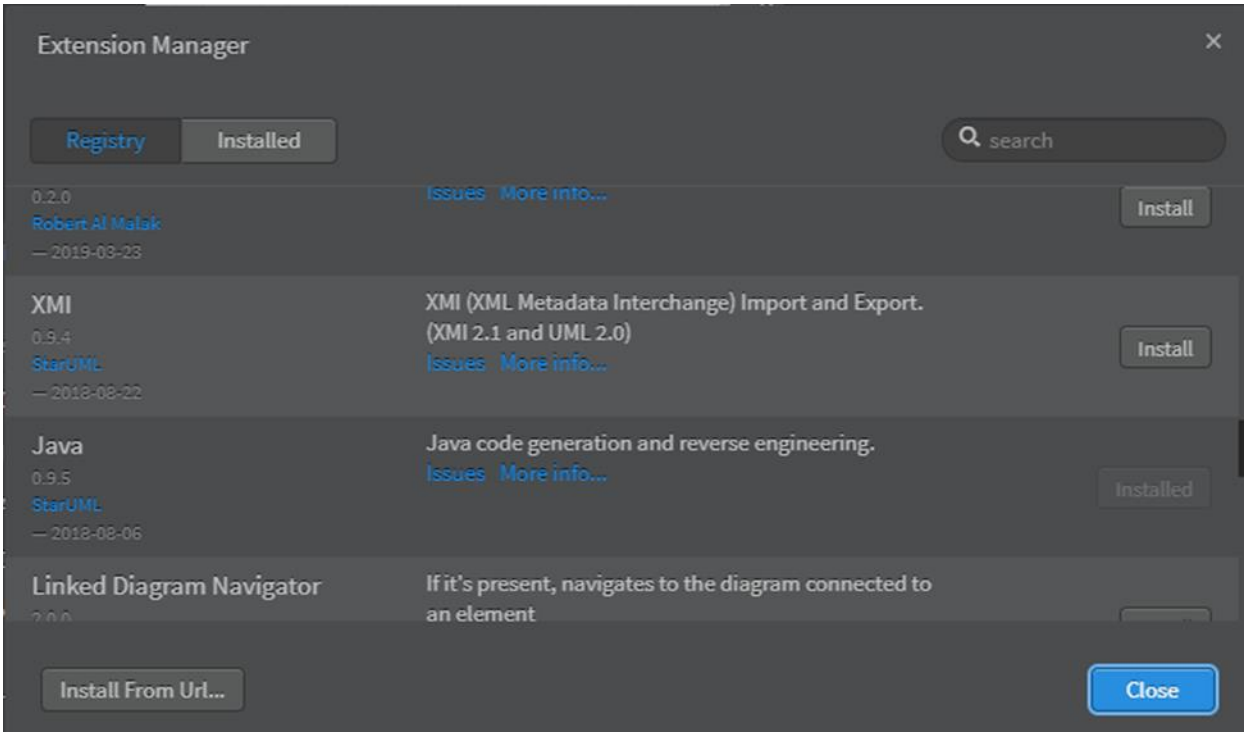

Рис. 12 – Выбор плагина

3. После установки плагин появится во вкладке Installed (рис.13):

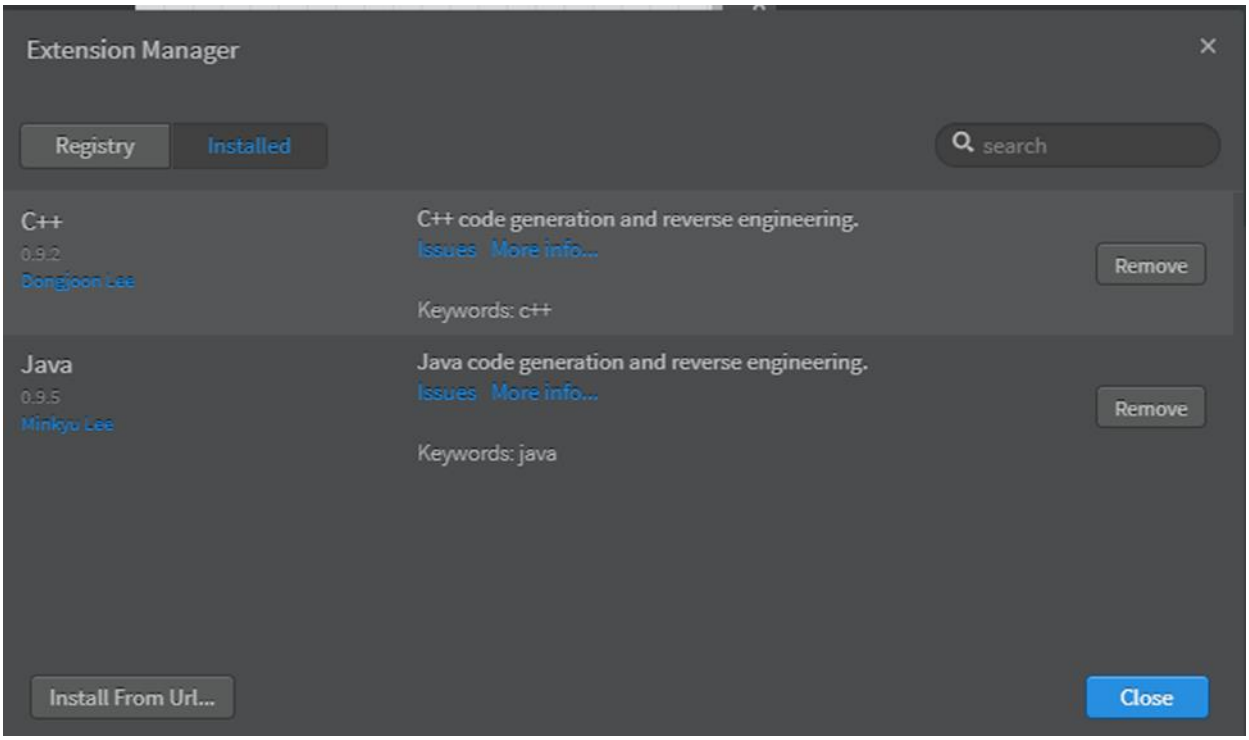

Рис. 13 – Список установленных плагинов

4. Далее необходимо перегрузить StarUML.

Установка свойств генерации программного кода

Для каждого языка в StarUML предусмотрен ряд определенных свойств генерации программного кода (рис. 14).

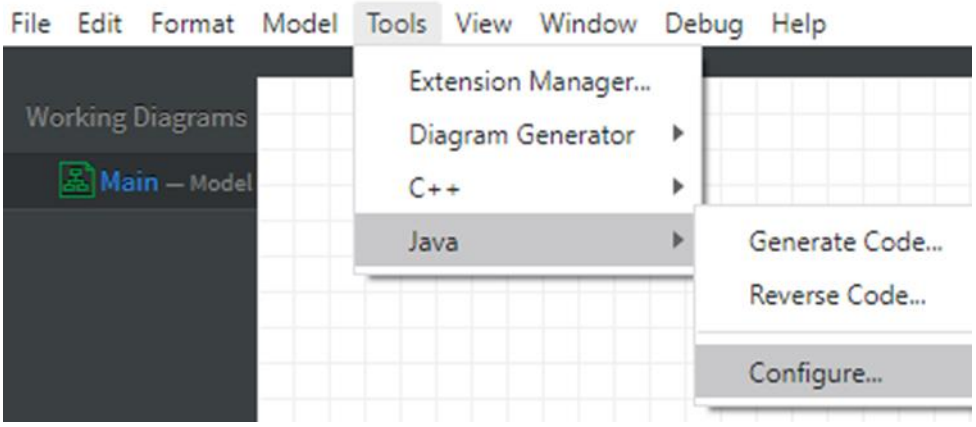

Рис. 14 – Меню для выбора настройки конфигурации свойств генерации кода на Java

Можно установить несколько параметров генерации программного кода для классов, атрибутов, компонентов и других элементов модели. Этими свойствами определяется способ генерации программ (рис. 15). В StarUML предлагаются общепринятые параметры по умолчанию.

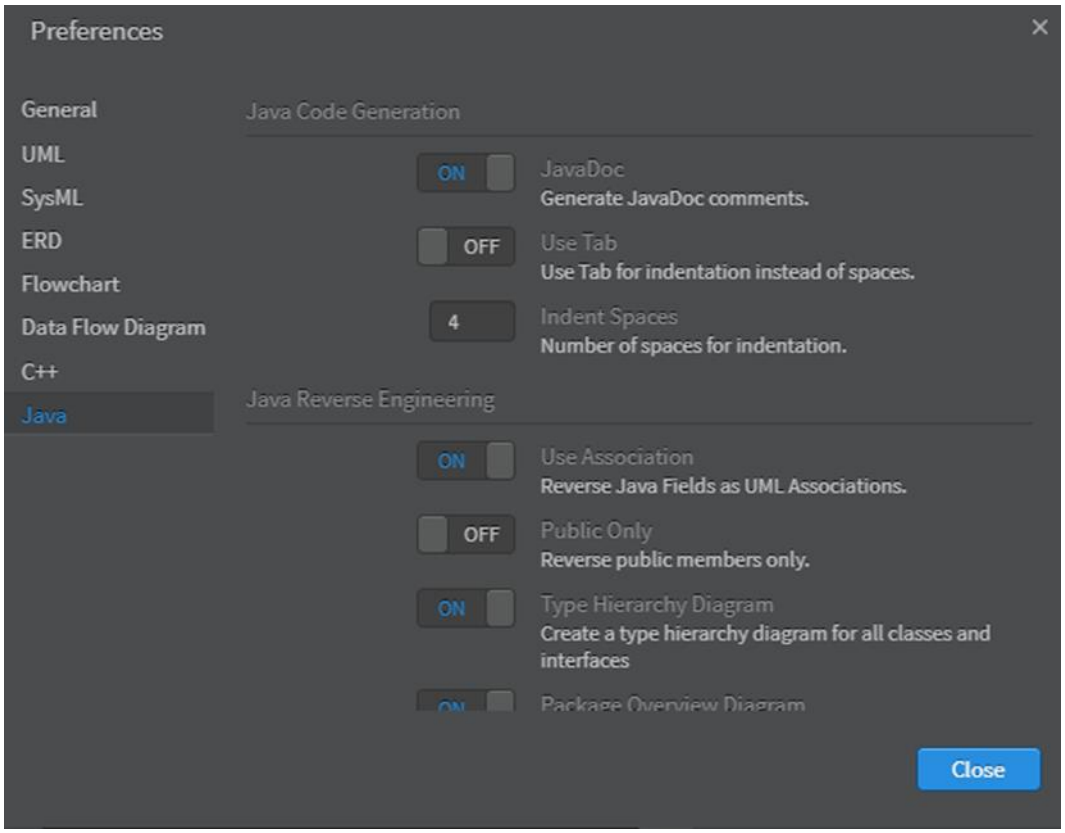

Рис. 15 – Окно свойств генерации кода на Java

Генерация программного кода

Генерация программного кода в среде StarUML возможна для отдельного класса или компонента.

1. Нужный элемент модели предварительно следует выделить в браузере проекта и выполнить операцию контекстного меню, например, *Tools → Java → Generate Code...* В результате этого будет открыто диалоговое окно с

предложением выбора классов для генерации программного кода на выбранном языке программирования (рис. 16).

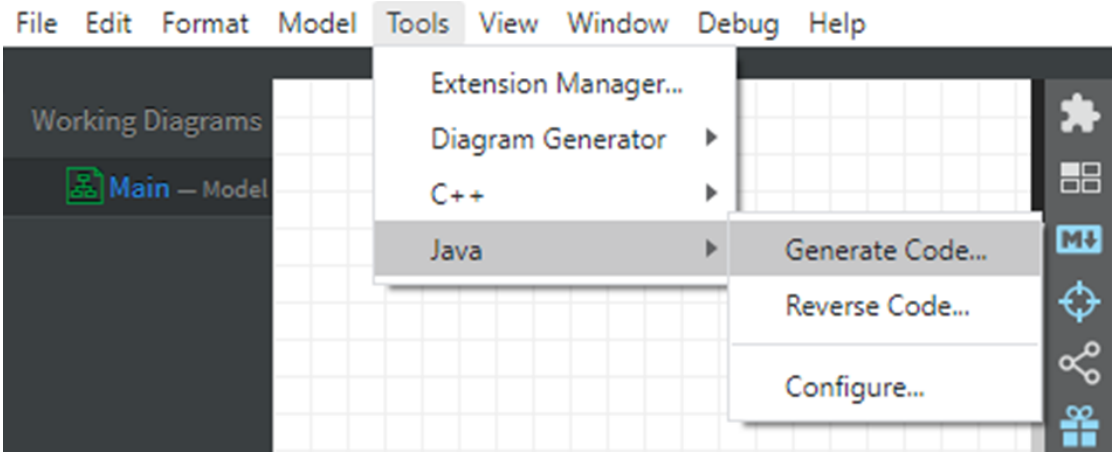

Рис. 16 - Окно выбора языка для генерации программного кода

2. Выбрать соответствующие классы и нажать кнопку OK (рис. 17).

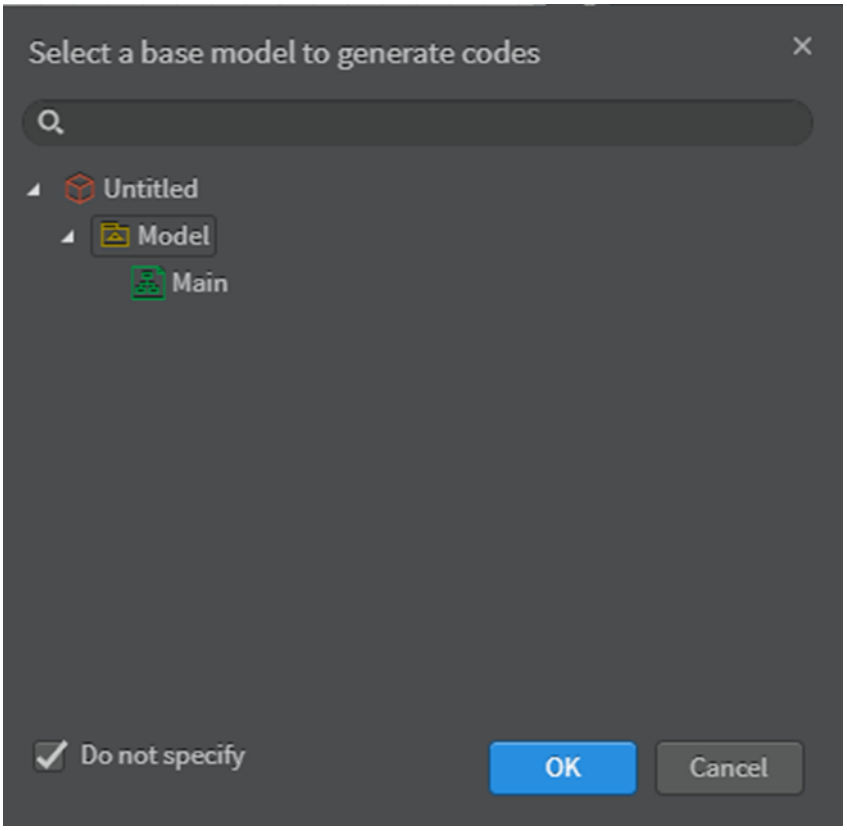

Рис. 17 - Окно выбора моделей классов для генерации программного кода

3. Выбрать место для записи сгенерированного программного кода (рис. 18).

Не забудьте запомнить место, где в дальнейшем искать созданный программный код!

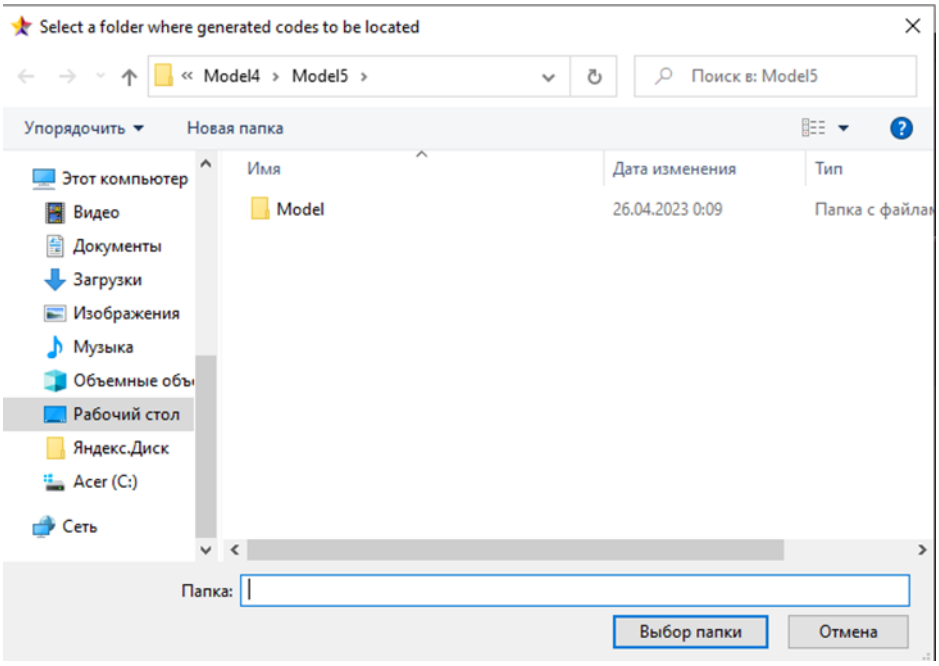

Рис. 18 - Окно выбора месторасположения сгенерированного программного кода

Результаты генерации

В результате кодогенерации StarUML создает файла выбранного кода. Эти файлы открываются с помощью блокнота и теперь легко можно увидеть скелет класса, с различными комментариями, которые писали вы на диаграммах, и комментарии, которые вставляет сама StarUML. Теперь можно открыть файл и доработать класс, описать работу функций, добавить необходимый расчет и т.п.

Сгенерированные программой StarUML файлы с текстом программного кода содержат минимум информации. Для включения дополнительных элементов в программный код следует изменить свойства генерации программного кода, установленные по умолчанию.

В заключение следует отметить, что эффект от использования средства StarUML проявляется при разработке *масштабных проектов* в составе команды или проектной группы. Однако ситуация покажется не столь тривиальной, когда станет необходимо выполнить проект с несколькими десятками вариантов использования и сотней *классов*. Именно для подобных проектов использование дополнительных возможностей StarUML и нотации языка UML для документирования и реализации соответствующих моделей будет очень полено и актуально.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое объект в объектном моделировании?
2. Что понимается под классом в объектном моделировании?
3. Какие типы отношений имеются в нотации UML?
4. Назовите основные стереотипы класса.

5. Что такое диаграмма классов?

6. Назовите основные элементы диаграммы классов.

7. Назовите основную характеристику класса.

8. Что понимается под операцией в диаграмме классов?

9. Назовите основные типы классов.

10. Какие типы связей поддерживаются в диаграммах классов?

11. Какие диаграммы необходимо предварительно разработать, чтобы выполнить кодогенерацию?

12. Что необходимо добавить в шаблоны классов для получения работоспособного приложения?

13. Какие основные этапы кодогенерации в StarUML вы знаете? Расскажите кратко о каждом из них?

ПРИМЕР ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Примером являются требования к ИС и модели ИС «PAsystem», представленные в примере лабораторных работ №3-4.

1. Производим выделение классов:

а)

1) PASystem (Система);

2) КритерииОценивания;

3) Рубрика;

4) Уровень;

5) Работа;

6) СписокidНазначений;

б)

1) СписокРубрик;

2) СписокУровней;

3) Назначение;

4) Оценка;

5) ИтоговаяОценка;

в)

1) СервисСозданияКрОцен;

2) СервисРабот;

3) СервисВычИтогОценки.

Активного класса в диаграмме классов нет.

Абстрактного класса в диаграмме классов нет.

2. Диаграмма классов PASystem представлена на рисунке 19.

3. Распределение обязанностей между некоторыми классами:

а) классам СервисВычИтогОценки, Итоговая Оценка, Оценка вменяется обязанность знать итоговую оценку обучающегося,

б) классам СписокУровней, Рубрика, СписокРубрик, КритерииОценивания, СервисРабот, СервисВычИтогОценки вменяется обязанность создавать объекты классов Уровень, СписокУровней, Рубрика, СписокРубрик, СписокidНазначений, Оценка, ИтоговаяОценка соответственно, так как между ними установлено отношение агрегации,

в) классы СервисСозданияКрОцен, СервисРабот, СервисВычИтогОценки были дополнительно введены, чтобы разгрузить класс PASystem, распределив между этими сервисами обязанности по функциональному принципу,

г) классам PASystem, СервисСозданияКрОцен, СервисРабот, СервисВычИтогОценки вменяется обязанность отвечать за обработку системных сообщений.

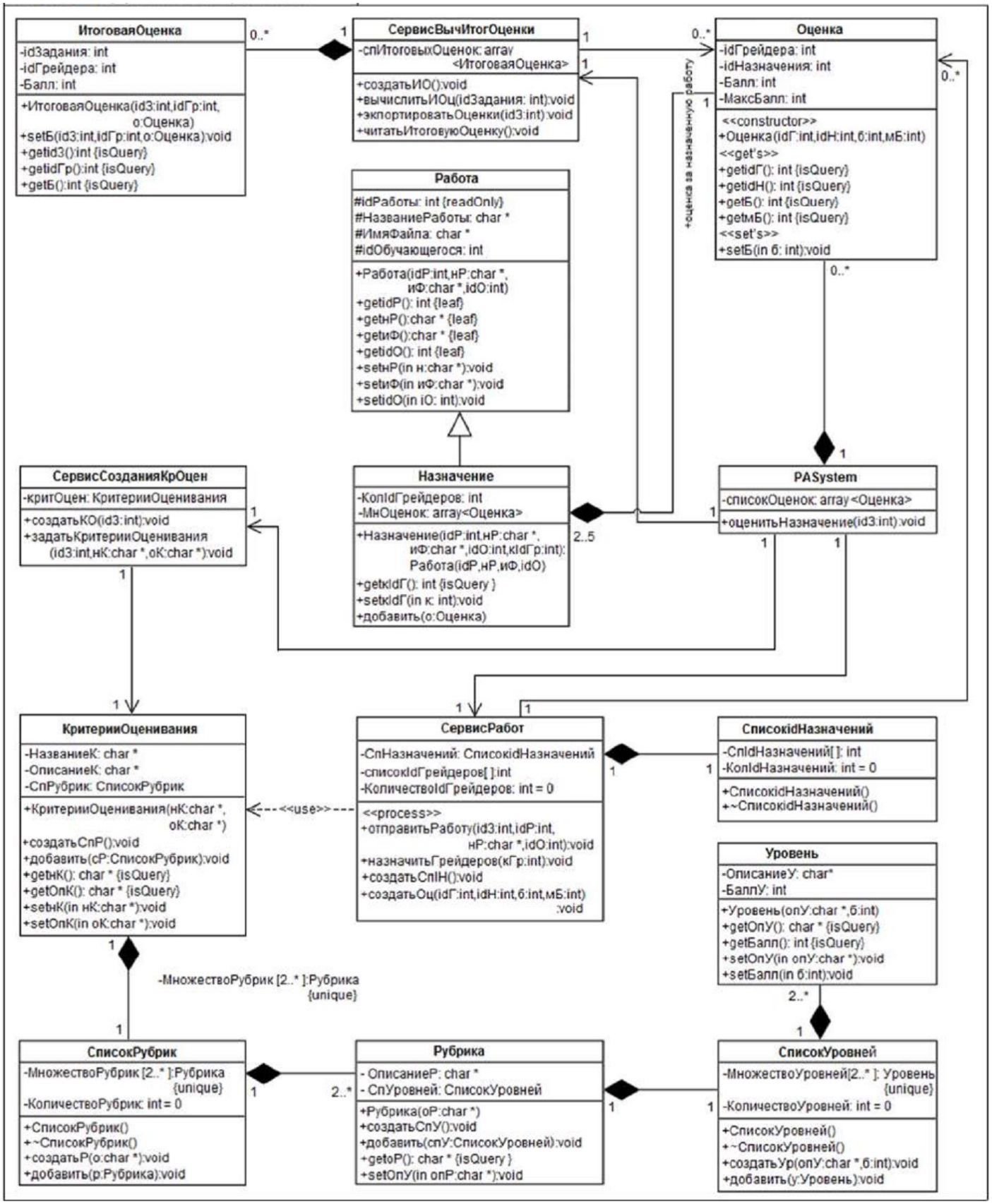

Рис. 19 - Диаграмма классов PASystem

4. Атрибуты классов, информация о типах данных и об области видимости.

Необходимо предварительно выбрать целевой язык ООП для прямого проектирования. Система PASystem будет реализована на языке C++. Поэтому для целочисленных данных в диаграмме классов использован тип «int», для строковых – «char *», для обозначения пустого значения – «void».

Атрибуты классов п. 1 а) и б) совпадают с атрибутами одноименных понятий концептуальной модели. Атрибутами классов п. 1 в) являются:

а) критОцен для класса СервисСозданияКрОцен;

б) спНазначений и списокIdГрейдеров для класса СервисРабот;

в) спИтоговыхОценок для класса СервисВычИтогОценки.

Атрибуты классов-контейнеров, предназначенные для хранения элементов других классов, моделируются двумя способами:

1) массивами языка С++ (классы СписокУровней, СписокРубрик, СервисРабот, СписокidНазначений,

2) последовательными контейнерами С++ (array <тип>) (классы PASystem, Назначение, СервисВычИтогОценки.

В первом случае кроме атрибута-массива классы имеют целочисленный атрибут для хранения текущего значения количества элементов в нем.

В UML по умолчанию атрибут имеет область видимости «+» (public).

Атрибуты всех классов, кроме класса Работа, размещены в части класса private (область видимости « -»). Атрибуты класса Работа, который является родительским по отношению к классу Назначение, размещены в части класса protected (область видимости «#»). Это позволяет реализовать отношение обобщения средствами механизма наследования классов в целевом языке ООП.

Вычислимых атрибутов в диаграмме классов нет.

5. Дополнительная информация об атрибутах классов.

Множественность не является обязательной и записывается после типа данных в виде диапазона целочисленных значений, напр., «ФИО: char * [0..1]».

У атрибута может быть указано начальное значение, которое записывается после типа данных в стиле инициализации значения переменных языка С при их объявлении, напр., «левыйУгол: Координаты=(0,0)».

У атрибутов могут быть указано свойство, которое записывается после типа данных в фигурных скобках :

– readOnly (значение атрибута не может быть изменено),

– ordered (у атрибута с множественностью, большей единицы, значения упорядочены),

– unique (у атрибута с множественностью, большей единицы, дубликаты значений запрещены).

По умолчанию для атрибута установлено свойство «changeable», свидетельствующее об отсутствии ограничений на модификацию значения.

6. Операции классов.

Операции-действия классов п.1, а) соответствуют сообщениям диаграмм сотрудничества. Операции-модификаторы, имеющие в имени префикс «get», указаны для всех атрибутов классов. Операции-селекторы, имеющие в имени префикс «set», указаны только для тех атрибутов классов, которые будут изменяться после установки начальных значений конструктором этого класса.

Операции-действия классов п.1, в) соответствуют п.2.

Большинство классов п.1, а) и б) имеют параметризованные конструкторы. Классы СписокРубрик и СписокУровней имеют конструктор по умолчанию. Так как класс Назначение является производным от класса Работа, его конструктор вызывает явно конструктор базового класса для инициализации унаследованных атрибутов. Для классов п.1, в) конструкторы не указаны.

Деструкторы в диаграмме классов указаны явно только для классов СписокidНазначений, СписокРубрик и СписокУровней, у которых такие деструкторы должны явно освобождать память элементов массива соответствующего класса-контейнера. У остальных классов деструкторы не указаны, так как предполагается наличие у каждого класса деструктора по умолчанию.

Операции всех классов размещены в части класса public (область видимости «+»). Это позволяет использовать их другим классам.

Для иллюстрации рекомендации п.5 в классах Оценка и СервисРабот использованы стереотипы для категорий операций этих классов.

7. Дополнительная информация об аргументах операций.

Направление аргумента операции записывается перед его именем и указывает, что аргумент является входным («in»), выходным («out») или обновляемым («inout»).

Значение по умолчанию записывается после типа данных в стиле инициализации значения переменных языка С при их объявлении, напр., «+setHh(iHh: int = 0)».

8. Свойства операций. У атрибутов могут быть указано свойство, которое записывается после типа данных в фигурных скобках:

– leaf (не может быть полиморфной / переопре деляться при наследовании),

– isQuery (ее выполнение не изменяет состояния объекта),

– sequential (допустим один поток вызовов операции, нет параллелизма),

– guarded (допустим параллелизм, но исполнение последовательное),

– concurrent (допустим параллелизм, разрешено параллельное и/или множественное исполнение).

9. Отношения между классами. В диаграмме классов присутствуют отношения обобщения, агрегации, ассоциации и зависимости.

10. Обобщение между понятиями Работа и Назначение сохраняется между одноименными классами в диаграмме классов.

11. Композитная агрегация между понятиями и кратности ролей сохраняются между одноименными классами в диаграмме классов. Дополнительно это отношение установлено между такими парами классов:

а) СервисРабот – СписокidНазначений (кратности ролей 1:1),

б) СервисВычИтогОценки – ИтоговаяОценка (кратности ролей 1:0..*),

в) PASystem – Оценка (кратности ролей 1:0..*).

12. Ассоциации связывают классы п. 1, в) с остальными элементами диаграммы классов (кратности ролей 1:1):

а) PASystem – СервисСозданияКрОцен,

б) PASystem – СервисРабот,

в) PASystem – СервисВычИтогОценки,

г) СервисСозданияКрОцен – КритерииОценивания,

д) СервисРабот – Оценка,

е) СервисВычИтогОценки – Оценка.

13. Зависимость Использование установлена между классами СервисРабот и КритерииОценивания, так как при оценивании работ обучающихся Грейдеры используют критерии оценивания задания.

14. Направление навигации указано явно между классами п.7.

Пример описания некоторых классов

1. Класс СервисСозданияКрОцен. Предназначен для того, чтобы инструктор мог создавать критерии оценивания задания. Является контроллером прецедента «Настройка критериев оценивания». Является создателем объектов класса КритерииОценивания.

Имеет атрибут «критОцен» (объект класса КритерииОценивания).

Имеет операции:

– «создатьКО (id3: int) : void» (создает критерии оценивания задания с идентификатором id3),

– «задатьКритерииОценивания (id3: int, нК: char *, оК: char *): void» (задает название и описание критерия оценивания задания с идентификатором id3).

2. Класс КритерииОценивания. Предназначен для создания критерия оценивания задания и его параметров. Является контейнером и создателем объектов класса СписокРубрик.

Имеет атрибуты:

– «НазваниеК» (строка названия критерия оценивания),

– «ОписаниеК» (строка описания критерия оценивания),

– «СпРубрик» (объект класса СписокРубрик).

Имеет операции:

– «КритерииОценивания (нК: char *, оК: char *)» (параметризованный конструктор, срабатывающий при создании объекта этого класса и устанавливающий начальные значения атрибутов «НазваниеК» и «ОписаниеК»),

– «создатьСпР (): void» (создает объект класса СписокРубрик),

- «добавить (сР: СписокРубрик) :void» (добавляет/обновляет значение атрибута «СпРубрик),

– «getнК (): char *» (возвращает значение атрибута «НазваниеК»),

– «getОпК (): char *» (возвращает значение атрибута «ОписаниеК»),

– «setнК (нК: char *): void» (устанавливает значение атрибута «НазваниеК»),

– «setОпК (оК: char *): void» (устанавливает значение атрибута «ОписаниеК»).

3. Класс СписокРубрик. Предназначен для создания списка рубрик критерия оценивания и его параметра. Является контейнером и создателем объектов класса Рубрика.

Имеет атрибуты:

– «МножествоРубрик» (массив объектов класса Рубрика),

– «КоличествоРубрик» (текущее количество объектов атрибута «МножествоРубрик»).

Имеет операции:

– «СписокРубрик ()» (конструктор по умолчанию, срабатывающий при создании объекта этого класса и устанавливающий пустое значение атрибута «МножествоРубрик» и нулевое значение атрибута «КоличествоРубрик»),

– «~СписокРубрик ()» (деструктор, который уничтожает все объекты, содержащиеся в атрибуте «МножествоРубрик», освобождая занятую ими амять, и удаляет сам объект класса СписокРубрик),

– «создатьР (о: char *): void» (создает объект класса Рубрика с указанным начальным значением его атрибута «ОписаниеР»),

– «добавить (Р: Рубрика): void» (добавляет объект Р класса «Рубрика» в атрибут «МножествоРубрик» и увеличивает значение атрибута «КоличествоРубрик»).

4. Класс Рубрика. Предназначен для создания рубрики критерия оценивания задания и ее параметров. Является контейнером и создателем объектов класса СписокУровней.

Имеет атрибуты:

– «ОписаниеР» (строка описания рубрики оценивания),

– «СпУровней» (объект класса СписокУровней).

Имеет операции:

– Рубрика (оР: char *) (параметризованный конструктор, срабатывающий при создании объекта этого класса и устанавливающий начальное значение атрибута «ОписаниеР»),

– создатьСпУ (): void (создает объект класса СписокУровней),

– добавитьСпУ (спУ: СписокУровней) (добавляет/обновляет значение атрибута «СпУровней),

– getоР (): char * (возвращает значение атрибута «ОписаниеР»),

– setОпУ (опР: char *): void (устанавливает значение атрибута «ОписаниеР»).