Лабораторная работа № 7

Тема: *разработка диаграмм классов.*
Цель: *изучить принципы построения диаграмм классов и пакетов языка UML, освоить построение диаграмм классов в CASE-средстве Enterprise Architect 8.*

Краткая теория

Диаграмма классов занимает центральное место при проектировании программной системы с использованием объектно-ориентированного подхода к разработке ПО. Большинство современных CASE-средств осуществляют автоматическую генерацию кода основываясь именно на этой диаграмме.

Диаграмма классов – диаграмма, предназначенная для представления модели статической структуры программной системы в терминологии классов объектно-ориентированного программирования. Разработка этой диаграммы преследует следующие цели:

- определить сущности предметной области и представить их в форме классов с соответствующими атрибутами и операциями;
- определить взаимосвязи между сущностями предметной области и представить их в форме типовых отношений между классами;
- разработать исходную модель программно системы для последующей реализации в форме физических моделей;
- подготовить документацию для последующей разработки программного кода.

Диаграмма классов в общем случае представляет собой граф, вершинами которого являются элементы типа «классификатор», связанные различными типами структурных отношений.

Класс – элемент модели, который описывает множество объектов, имеющих одинаковые спецификации характеристик, ограничений и семантики. Класс является специализацией классификатора. Структурные характеристики класса принято называть атрибутами, а характеристики поведения – операциями.

Графически класс на языке UML отображается в виде прямоугольника, разделенного по вертикали на три секции. В верхней секции указывается имя класса, в средней секции – атрибуты, а в нижней – операции. Если у класса отсутствуют атрибуты или операции, то соответствующие секции могут быть опущены. Примеры графического отображения классов на диаграмме классов приведены на рисунке 1.

Рисунок 1 – Нотация класса на языке UML
а) – только имя класса; б) – указываются все секции; в) – секция атрибутов отсутствует; г) – активный класс

Активный класс – класс, каждый экземпляр которого имеет свою собственную нить управления. Пассивный класс – класс, каждый экземпляр которого выполняется в контексте некоторого другого объекта.

Имя класса – строка текста, предназначенная для идентификации класса на диаграмме классов. Имя класса должно быть уникальным в пределах пакета, который может содержать одну или несколько диаграмм классов. Имя класса отображается по центру верхней секции

полужирным шрифтом. Желательно, чтобы имя класса было допустимым идентификатором целевого языка программирования, так как это существенно облегчит процесс автоматической генерации кода. В большинстве CASE-средств можно помимо имени класса задать его псевдоним (alias), который будет использоваться при отображении класса на диаграмме, а при генерации кода будет использоваться само имя класса. Класс может не иметь экземпляров или объектов. Такой класс называется абстрактным и его имя на диаграмме выделяется курсивом.

Имя класса можно задавать двумя способами: в виде простого имени или в виде квалифицированного имени. Простое имя класса – собственно имя класса. Квалифицированное имя класса включает в себя простое имя класса и имена всех вложенных пакетов, в которых описан данный класс. Имена пакетов отделяются друг от друга и от простого имени класса двойным двоеточием, что соответствует следующему синтаксису:

```
<имя пакета 1>:: … ::<имя пакета N>::<простое имя класса>
```

Атрибут класса служит для представления отдельной структурной характеристики или свойства, которое является общим для всех объектов данного класса. Каждому атрибуту класса в языке UML соответствует отдельная строка, имеющая в нотации БНФ следующий формат:

```
<атрибут> ::= [<видимость>] ['/'] <имя> [':' <тип>]
              [ '[' <кратность> ']' ] [ '=' <значение>]
              [ '{' <модификатор> [ ',' <модификатор>]* '}' ]
```

Видимость определяет доступность атрибута для остальных элементов. Всего в языке UML выделяют четыре типа видимости:

- открытый (public, знак ‘+’),
- закрытый (private, знак ‘-’),
- защищенный (protected, знак ‘#’),
- пакетный (package, знак ‘~’).

Первые три типа рассматривались в предыдущей главе. В языке UML введен еще один тип видимости – пакетный. Элемент с такой видимостью виден всем элементам в ближайшем охватывающем пакете.

Символ ‘/’ в описании атрибута указывает на то, что этот атрабут является производным (вычислимым). Значение такого атрибута может быть вычислено на основе значений других атрибутов этого или других классов.

Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута. Имя должно быть уникальным в пределах данного класса и, желательно, быть корректным идентификатором целевого языка программирования.

Тип атрибута – имя классификатора, который является типом данного атрибута. Тип атрибута представляет собой имя некоторого типа данных, определенный в языке UML, разработчиком или заложенным в целевой язык программирования.

Кратность атрибута характеризует общее количество конкретных значений для атрибута, которые могут быть заданы для объектов данного класса. В общем случае синтаксис описания кратности должен удовлетворять следующему формату в нотации БНФ:

```
<кратность> ::= [ <нижняя граница> '..' ] <верхняя граница>
```

Нижняя и верхняя границы задают интервал, в пределах которого должно находиться количество элементов данного атрибута. При этом значение нижней границы должно быть положительным целочисленным литералом, а значение верхней границы – произвольным натуральным числом, большим, чем значение нижней границы. Для обозначения бесконечности верхней границы используется символ ‘*’.

Значение по умолчанию – некоторое выражение, которое служит для задания начального значения или значений данного атрибута в момент создания отдельного объекта данного класса.

Модификатор атрибута представляет собой текстовое выражение, которое придает дополнительную семантику данному атрибуту. В языке UML изначально заложено несколько модификаторов: readOnly – атрибут только для чтения, ordered – множество значений атрибута

упорядочено, sequence – множество значений атрибута представляет собой последовательность.

Операция класса служит для представления отдельной характеристики поведения, которая является общей для всех объектов данного класса. Операция класса специфицирует некоторый сервис, который предоставляет каждый объект класса по требованию своих клиентов. Совокупность всех операций характеризует функциональный аспект поведения всех объектов данного класса. Описание отдельной операции класса имеет следующий синтаксис (БНФ):

```
<операция>::= [<видимость>] <имя> '('[<список параметров>]')'
              [ ':' <тип возвращаемого значения>]
              ['{' <свойство> [',' <свойство>] '}']
```

Видимость определяет доступность операции для окружения. Значения видимости операции совпадают со значениями видимости атрибутов.

Имя операции – строка текста, которая используется в качестве идентификатора этой операции. Имя операции должно быть уникальным в пределах класса и, желательно, корректным с точки зрения целевого языка программирования.

Список параметров представляет собой перечень разделенных запятыми формальных параметров операции, записанный в следующем формате БНФ:

```
<список параметров> ::= <параметр> [',' <параметр>]*
```

Параметр является спецификацией аргумента и используется при выполнении операции. Каждый параметр должен иметь следующий формат записи в нотации БНФ:

```
<параметр> ::= [<направление>] <имя> ':' <тип>
               [ '=' <значение по умолчанию>]
```

Направление указывает режим передачи параметра. Выделяют четыре вида направления:

- in – значения этого параметра передаются в операцию вызывающим объектом;
- inout – значения этого параметра передаются в операцию вызывающим объектом и затем обратно вызывающему объекту по окончании операции;
- out – значения этого параметра передаются вызывающему объекту по окончании выполнения операции;
- return – значения этого параметра передаются в качестве возвращаемых значений вызывающему объекту по окончании выполнения операции.

Имя параметра – строка текста, идентифицирующая параметр, которая должна быть уникальна в пределах данной операции. В остальном к имени параметра предъявляются те же требования, что и к именам атрибутов и операций. К типу параметра и типу возвращаемого операцией значения предъявляются те же требования, что и к типу атрибута. Для параметра возможно указать значение по умолчанию, которое будет передано в операцию, если значение параметра не будет указано.

Свойство операции служит для указания дополнительных свойств, которые могут быть применены к данной операции. В языке UML изначально заложены следующие свойства операций:

- query – операция является запросом (не модифицирует атрибуты класса);
- sequential – в каждый момент времени в объект поступает только один вызов операции;
- guarded – допускается одновременное поступление в объект нескольких вызовов, но в каждый момент времени обрабатывается только один вызов охраняемой операции;
- concurrent – в объект поступает несколько потоков вызовов операций и при этом разрешается параллельное (и множественное) выполнение операции.

Свойства sequential, guarded и concurrent не могут указываться для одной и той же операции одновременно.

В языке UML статические атрибуты и операции выделяются подчеркиванием надписи. Также для атрибутов и операций могут быть указаны стереотипы, в таком случае CASE-средства, как правило, осуществляют группировку характеристик классов по стереотипам.

Кроме внутреннего устройства классов важную роль при построении моделей программного обеспечения играют различные отношения между классами, которые также

изображаются на диаграмме классов. В языке UML определены следующие виды отношений между классами: ассоциация, обобщение, агрегация, композиция, зависимость, реализация.

Ассоциация – произвольное семантическое отношение или взаимосвязь между классами. В языке UML выделяют несколько видов ассоциаций:

- бинарная ассоциация,
- исключающая ассоциация,
- n-арная ассоциация,
- ассоциация-класс.

Бинарная ассоциация обозначается сплошной линией, соединяющей два класса, со стрелками на концах или без них, а также с некоторыми необязательными дополнительными символами, которые характеризуют специальные свойства ассоциации. К таким символам относят: имя ассоциации, а также роли, видимость и кратность концов ассоциации.

Ассоциация объявляет, что между экземплярами ассоциированных классов могут существовать связи, представленные кортежем с одним значением для каждого конца ассоциации. Кратность конца ассоциации ограничивает размер этой коллекции.

Рассмотрим пример отношения ассоциации между классами Студент и Университет, приведенный на рисунке 2. Здесь указано имя ассоциации – Учится, а также роли экземпляра класса Студент (учащийся) и экземпляра класса Университет (УО – Учреждение образования). Также здесь указано, что все стороны в ассоциации имеют открытую видимость (символ ‘+’ перед именами ролей). С точки зрения значений кратности здесь указано, что в одном университете может учиться один и более студент, а один студент может учиться только в одном университете.

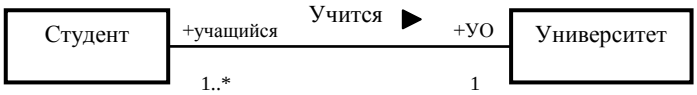

Рисунок 2 – Пример отношения ассоциации

Как правило, на диаграммах одновременно имя ассоциации и роли не указываются, так как это делает отношение перегруженным текстовой информацией. Для конца ассоциации можно указать наличие или отсутствие навигации. Наличие навигации обозначается в виде V-образной стрелки, а ее отсутствие – символом X на соответствующем конце ассоциации. Наличие навигация означает доступность элемента на этом конце навигации с другого конца, а ее отсутствие – недоступность. Например, в отношении ассоциации между классом Треугольник и классом Точка (рисунок 3) наличие навигации со стороны класса Точка означает, что объект класса Треугольник «знает» свои вершины, а отсутствие навигации со стороны класса Треугольник означает, что объект класса Точка «не знает», какому треугольнику он принадлежит.

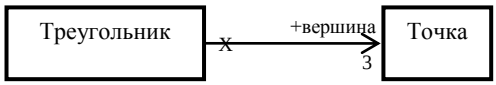

Рисунок 3 – Обозначение наличия и отсутствия навигации

Частным случаем отношения ассоциации является исключающая ассоциация. Семантика данной ассоциации указывает на тот факт, что из нескольких потенциально возможных вариантов данной ассоциации в данный момент времени может использоваться только один ее экземпляр. На диаграмме классов исключающая ассоциация изображается с помощью дополнительной пунктирной линии, соединяющей две и более ассоциаций, рядом с которой записывается ограничение в форме специального ключевого слова {xor}. На рисунке 4 приведен пример исключающей ассоциации, на которой показано, что объект класса Счет может принадлежать либо объекту класса Физическое лицо, либо объекту класса Юридическое лицо.

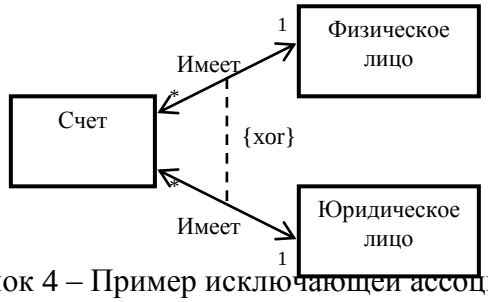

Рисунок 4 – Пример исключающей ассоциации

N-арная ассоциация представляет собой более общий случай, когда отношением ассоциации связывается более двух классов. Каждый экземпляр n-арной ассоциации представляет собой n-арный кортеж, состоящий в точности из n объектов соответствующих классов. N-арная ассоциация графически обозначается ромбом, от которого ведут сплошные линии к символам классов данной ассоциации. Имя n-арной ассоциации записывается рядом с ромбом соответствующей ассоциации. В качестве примера (рисунок 5) рассмотрим ассоциацию Игра между классами Команда, Год и Дата, символизирующую проведение игр футбольных команд в различные сезоны.

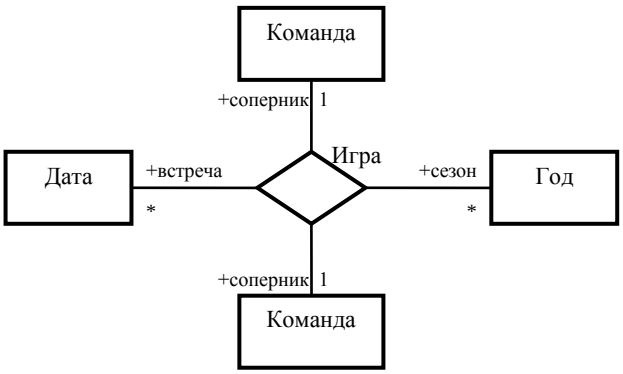

Рисунок 5 – Пример 4-арной ассоциации

Еще одной разновидностью отношения ассоциации является ассоциация-класс – элемент модели, который имеет как свойства ассоциации, так и класса, и предназначенный для спецификации дополнительный свойств ассоциации в форме атрибутов и, возможно, операций класса. Ассоциация-класс изображается форме символа класса, присоединенного к линии ассоциации посредством пунктирной линии. В качестве примера используем рассмотренную ранее n-арную ассоциацию Игра, выделив ее параметры (сезон, дата, место и результат) в качестве ассоциации-класса (рисунок 6). В приведенном на рисунке примере используется рефлексивная бинарная ассоциация.

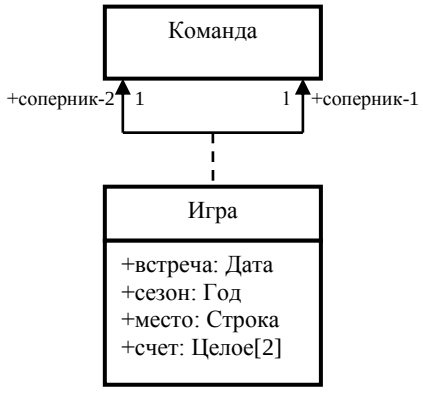

Рисунок 6 – Пример ассоциации-класса

На одном из концов ассоциации может быть указан квалификатор, который объявляет разбиение множества ассоциированных экземпляров относительно экземпляра на

квалифицированном конце ассоциации. Экземпляр квалификатора состоит из одного значения для атрибута каждого квалификатора. Если задан некоторый квалифицированный объект и экземпляр квалификатора, число объектов на другом конце ассоциации ограничено объявленной кратностью. Квалификатор изображается в форме маленького прямоугольника, присоединенного к концу линии ассоциации между ее окончанием и символом класса. На рисунке 7 приведен пример использования квалификатора, в котором показано, что класс Клиент соединяется отношением ассоциации с классом Банк посредством квалификатора номерСчета.

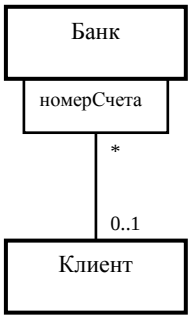

Рисунок 7 – Пример использования квалификатора

Отношение ассоциации является наиболее общей формой отношения в языке UML. Практически все другие типы отношений являются специальными случаями данного отношения. Особенно это справедливо для отношений агрегации и композиции, которые частными случаями ассоциации.

Агрегация – направленное отношение между двумя классами, предназначенное для представления ситуации, когда один из классов представляет собой некоторую сущность, которая включает в себя в качестве составных частей другие сущности. Как уже отмечалось в главе 7 отношение агрегации используется в объектно-ориентированном подходе для построения иерархий целое-часть: деление системы на составные части, которые при необходимости также могут быть подвергнуты декомпозиции. Агрегация наследует от ассоциации такие свойства как: имена концов ассоциации, кратность.

Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой не закрашенный (пустой) ромб для обозначения агрегированного конца линии ассоциации (рисунок 8).

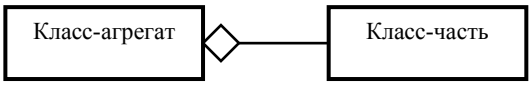

Рисунок 8 – Обозначение агрегации

Если существует две или более агрегации для одного агрегата, то они могут быть изображены в форме дерева посредством слияния концов агрегации в отдельный сегмент. В качестве примера рассмотрим отношение агрегации между составными частями персонального компьютера: системного блока, монитора, клавиатуры и мыши (рисунок 9).

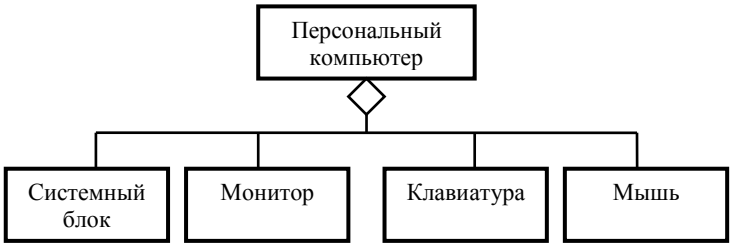

Рисунок 9 – Отношение агрегации в форме дерева

Композиция (композитная агрегация) – это отношение, предназначенное для спецификации более сильной формы агрегации, при которой с уничтожением объекта класса-агрегата, уничтожаются и все объекты, являющиеся его составными частями.

Графически отношение композиции изображается сплошной линией, один из концов которой (со стороны класса-агрегата) представляет собой ромб, который имеет заливку (рисунок 10).

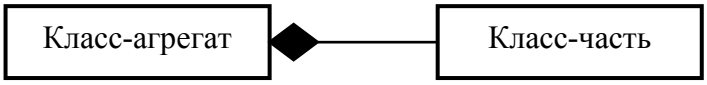

Рисунок 10 – Обозначение композиции

Также как и агрегация, композиция может быть представлена в форме дерева. Для примера рассмотрим отношение композиции: окно программы состоит из заголовка, двух полос прокрутки, рабочей области и главного меню (рисунок 11).

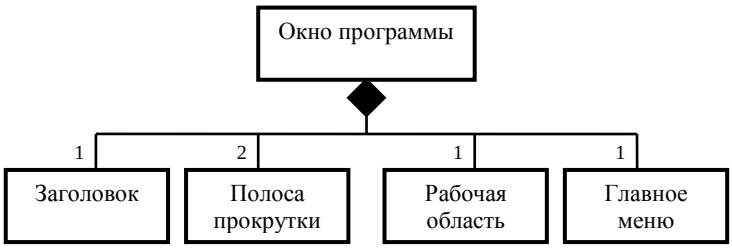

Рисунок 11 – Отношение агрегации в форме дерева

Отношение обобщения – это таксономическое отношение между более общим классификатором (родителем или предком) и более специальным классификатором (дочерним или потомком). Семантика отношения обобщения означает, что каждый экземпляр специального классификатора также является экземпляром общего классификатора. Отношение обобщения является направленным и может использоваться для представления иерархических взаимосвязей между классами, вариантами использования и другими элементами модели, которые являются классификаторами в нотации языка UML.

Согласно одному из главных принципов методологии ООАП – наследованию, класс-потомок обладает всеми свойствами и поведением класса-предка, а также может иметь дополнительные свойства и поведение, которые отсутствуют у класса предка. На диаграмме классов обобщение обеспечивает наследование атрибутов и операций классов и обозначается сплошной линией со стрелкой в форме не закрашенного треугольника на одном из своих концов. Стрелка указывает на общий класс или класс-предок, а ее противоположный конец – на специальный класс или класс-потомок (рисунок 12).

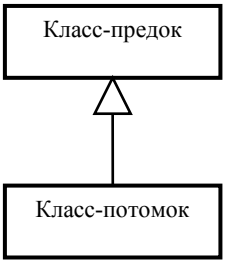

Рисунок 12 – Обозначение обобщения

Отношение обобщения допускает, чтобы от одного класса предка одновременно наследовали несколько классов-потомков. Это может быть отображено в виде отдельных отношений для каждого класса-потомка или в форме дерева, как это показано на рисунке 13. В языке UML допускается также, чтобы класс-потомок наследовал от нескольких классов предков (множественное наследование).

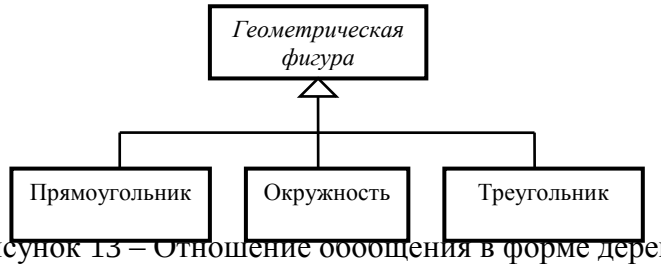

Рисунок 13 – Отношение обобщения в форме дерева

Множество обобщения – это элемент модели, экземпляры которого определяют коллекции подмножеств отношения обобщения. Каждое множество обобщения определяет отдельное множество отношений обобщения, которые описывают способ, которым общий классификатор (предок) может быть разделен на несколько специальных подтипов. Множество обобщения может содержать строку текста, указывающую на не которые специальные свойства этого отношения, оформленную в виде ограничения. В языке UML заложены следующие ограничения:

- {complete, disjoint} – означает, что данное множество обобщения является покрывающим и его специальные классы не имеют общих экземпляров;
- {incomplete, disjoint} – означает, что данное множество обобщения является покрывающим и его специальные классы не имеют общих экземпляров;
- {complete, overlapping} – означает, что данное множество обобщения является покрывающим и его специальные классы имеют общие экземпляры;
- {incomplete, overlapping} – означает, что данное множество обобщения не является покрывающим и его специальные классы имеют общие экземпляры.

Графически ограничения на множество обобщения изображаются с использованием нотации общей стрелки обобщения или пунктирной линии для отдельного множества обобщения, как это показано на рисунке 14.

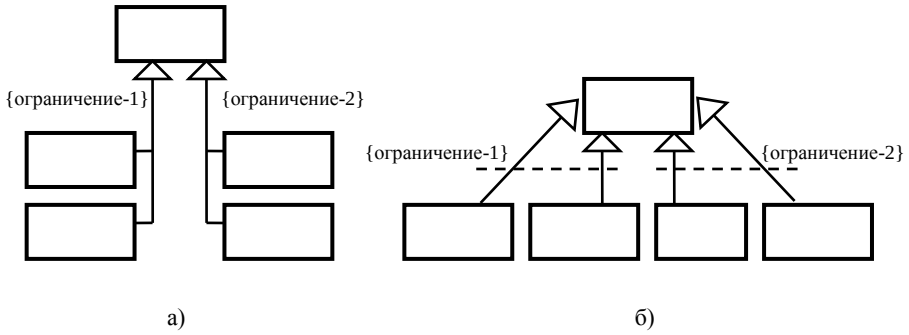

Рисунок 14 – Варианты нотации для ограничения на множество обобщения:
а) – с помощью стрелки обобщения; б) – с помощью пунктирной линии

На рисунке 15 приведен пример использования множеств обобщения с ограничениями. На этом примере показано, что множество классов-потомков Мужчина и Женщина является покрывающим и не имеющим общих экземпляров, а множество классов-потомков Кодировщик, Тестировщик, Проектировщик не является покрывающим и может содержать общие экземпляры.

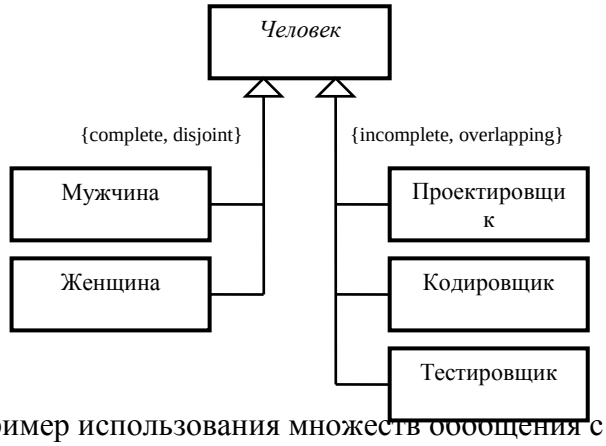

Рисунок 15 – Пример использования множеств обобщения с ограничениями

Зависимость – отношение, предназначенное для описания ситуации, когда отдельному элементу или множеству элементов модели требуются другие элементы модели для своей спецификации или реализации. Зависимость означает отношения типа «поставщик – клиент» между элементами модели, когда модификация поставщика может оказывать влияние на одного или несколько клиентов.

Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со V-образной стрелкой на одном из концов, направленной на независимый элемент (поставщик). Графическое изображение отношения зависимости приведено на рисунке 16.

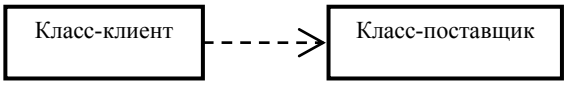

Рисунок 16 – Обозначение зависимости

Линия зависимости может быть помечена необязательным стереотипом и необязательным именем. Возможно наличие нескольких элементов для клиента или поставщика. В этом случае одна или более стрелок от клиентов соединяются с одной или более стрелок, указывающих на поставщиков.

Реализация – специализированное отношение зависимости между двумя элементами модели, один из которых представляет некоторую спецификацию (поставщик), а другой представляет его реализацию (клиент). Другими словами отношение реализации означает, что один классификатор (поставщик) определяет контракт, который другой классификатор (клиент) обязуется выполнять. Отношение реализации применяют в двух случаях:

- между интерфейсами и классами (или компонентами), реализующими их;
- между элементами варианты использования и кооперациями, которые реализуют их.

На диаграмме отношение реализации обозначается пунктирной линией с треугольной не закрашенной стрелкой на одном конце, направленной на поставщика контракта (рисунок 17).

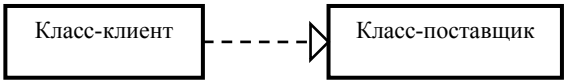

Рисунок 17 – Обозначение реализации

Чаще всего отношения зависимости и реализации используются между классами и интерфейсами. Интерфейс – вид класса, который представляет собой множество общедоступных характеристик и обязанностей. В языке UML интерфейс является специальным случаем класса, у которого, как правило, имеются операции и отсутствуют атрибуты. Интерфейсы предназначены для спецификации таких операций класса, объявления которых видимы извне, но особенности их реализации скрыты от пользователя. В этом смысле интерфейс не может существовать отдельно от класса, реализующего объявленные в нем операции.

Графически интерфейс в языке UML обозначается двояко: в виде класса со стереотипом <<interface>> или в виде кружка с именем. При этом имя интерфейса всегда должно начинаться с заглавной буквы ‘I’. На рисунке 18 приведен пример обозначения интерфейса.

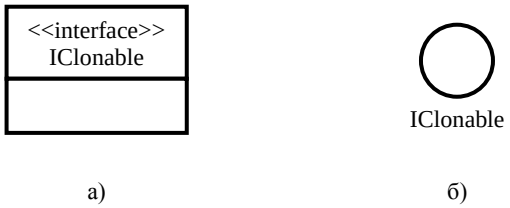

Рисунок 18 – Обозначение интерфейса
а) – в виде класса, б) – в виде круга

Любой класс может реализовывать (или как еще говорят – предоставлять) интерфейс или его использовать (или как еще говорят – требовать). В первом случае класс связывается с интерфейсом отношением реализации, а во втором случае – отношением зависимости. При этом в обоих случаях класс выступает в качестве клиента, а интерфейс – в качестве поставщика. На рисунке 19 приведен пример обозначения отношений реализации и зависимости между классами и интерфейсом. На этом примере класс Элемент реализует (предоставляет) интрефейс ICравнение, который содержит, например, операции сравнения двух объектов, а класс Сортировщик использует (требует) этот интерфейс для сравнения двух объектов класса Элемент, например, в процессе сортировки списка объектов этого класса.

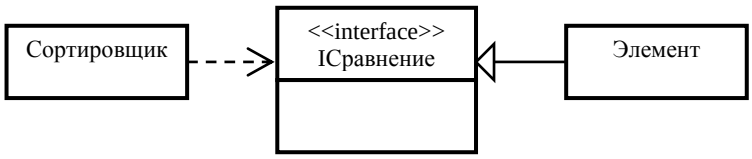

Рисунок 19 – Пример отношений между классами и интерфейсом

Если используется нотация круга для обозначения интерфейса, то отношение реализации на диаграмме заменяется отношением ассоциации (рисунок 20).

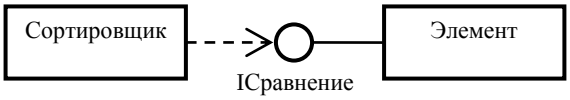

Рисунок 20 – Отношения между классами и интерфейсом, обозначенным в виде круга

Еще одним средством, используемым в языке UML, является шаблон. Шаблон (родовой класс) – классификатор, который в своем описании имеет несколько формальных параметров. Шаблон является параметризируемым элементом, который может быть использован для получения других элементов модели с помощью отношения связывания шаблона. Параметры шаблона специфицируют формальные параметры, которые должны быть заменены действительными параметрами (или по умолчанию) при связывании.

Графически шаблон класса изображается прямоугольником класса, на верхний правый угол, которого наложен маленький прямоугольник из пунктирных линий (рисунок 21). Основной прямоугольник может быть разделен на секции, аналогично обозначению обычных классов. В пунктирном прямоугольнике указывается список формальных параметров шаблона в форме списка, разделенного запятыми, или представлен по одному параметру в каждой строке.

Рисунок 21 – Графическое обозначение шаблона

Для обозначения того, что некоторый класс-клиент использует некоторый шаблон для своей последующей параметризации, используется отношение реализации со стереотипом <<bind>> и списком подстановок параметров шаблона, разделенных запятыми (рисунок 22). Каждый элемент списка указывается в следующей нотации БНФ:

```
<подстановка-параметра> ::=
            <имя-параметра> '->' <действительный-параметр>
```

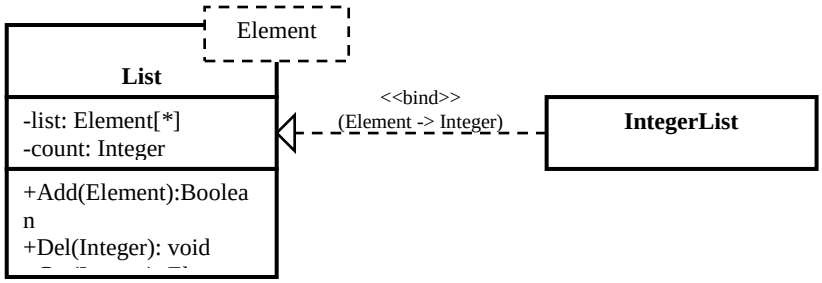

Рисунок 22 – Пример замещения шаблона

Диаграмма пакетов

Диаграмма пакетов в языке UML относится к дополнительным диаграммам для представления информации на логическом уровне и служит основным средством для представления архитектуры программных систем, а также повсеместно используется для спецификации всех основных конструкций метамодели языка UML. В виде самостоятельной канонической диаграммы диаграмма пакетов была введена только в языке UML стандарта 2.0, а до этого она рассматривалась как разновидность диаграммы классов.

Диаграмма пакетов предназначена для представления размещения элементов модели в пакетах и спецификации зависимостей между пакетами и их элементами. Как правило, основными предметами языка UML, изображаемыми на этой диаграмме, являются классы и пакеты.

Пакет – элемент модели, используемый для группировки других элементов модели. Элементы модели, входящие в состав некоторого пакета, называются его членами. Члены принадлежат пакту (или пакет владеет своими членами), поэтому при удалении пакета из модели все его члены также удаляются. Элементами пакета называются элементы модели, которые входят в пространство имен этого пакета. К элементам пакета относят как члены этого пакета, так и члены других импортируемых пакетов.

Графически пакет обозначается в форме прямоугольника, у которого слева к верхней стороне примыкает еще один меньший прямоугольник (ярлык пакета) (рисунок 23). Имя пакета, как правило, указывается внутри большого прямоугольника, но в некоторых CASE-средствах может указываться и на ярлыке пакета.

Рисунок 23 – Графическое изображение пакета

На диаграммах пакетов одни пакеты могут быть вложены в другие пакеты. В этом случае вложенный пакет называется подпакетом, а все его элементы также принадлежат любому пакету, для которого рассматриваемый пакет является вложенным. Вложенность пакетов в языке UML может обозначаться двумя способами. В первом способе вложенный пакет изображается внутри пакета, в который он вложен. Во втором случае используется специальное отношение вложения (введено только в стандарте UML 2.0). На рисунке 24 приведены примеры изображения вложенности пакетов.

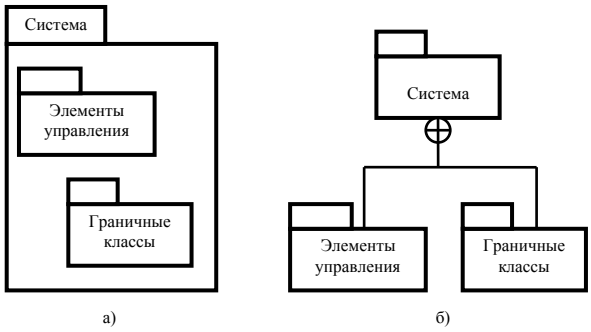

Рисунок 24 – Изображение вложенности пакетов
а) – внутри пакета, б) – с помощью отношения вложенности

Нотация обозначения вложенности пакетов в различных CASE-средствах может быть изменена. На элементы пакета можно ссылаться, используя квалифицированные (с указанием имени содержащего этот элементы пакета) и неквалифицированные (содержит только имя самого элемента) имена. Дополнительно каждый элемент пакета может иметь видимость, которая в общем случае может быть общедоступной (public), частной (private) или пакетной (packaged).

Помимо отношения вложенности между пакетами может существовать отношение зависимости и две его разновидности: импорт и слияние. В общем случае отношение зависимости между двумя пакетами показывает, что элементы одного пакета (зависимого) зависят от элементов другого пакета (независимого). На рисунке 25 с помощью этого отношения продемонстрирована зависимость пакетов в модели трехуровневой архитектуры, где каждый пакет содержит элементы, реализующие тот или иной уровень.

Рисунок 25 – Изображение зависимости пакетов

Импорт пакета – направленное отношение между пакетами, при котором члены одного пакета могут быть добавлены в пространство имен другого пакета. Импорт пакета позволяет ссылаться только на общедоступные члены импортируемого пакета в другом пространстве имен, используя при этом неквалифицированные имена. Импорт пакета обозначается с помощью отношения зависимости, направленного от импортирующего пакета к импортируемому пакету, со стереотипом <<import>> – для общедоступного импорта пакета или <<access>> – для закрытого импорта пакета. При общедоступном импорте пакета импортируемые элементы могут быть видимыми за пределами импортирующего пакета, а при закрытом импорте – нет. На рисунке 26 приведен пример общедоступного и закрытого импорта пакетов, где элементы пакета «Алгоритмы» импортируются в пакет «Обработка данных» в открытом режиме, а элементы пакета «ADO» – в закрытом. Если пакет «Обработка данных» будет импортирован в какой-либо другой пакет, то импортирующий пакет получит доступ только к элементам пакета «Алгоритмы».

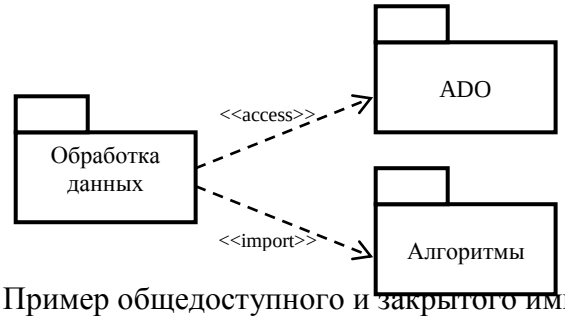

Рисунок 26 – Пример общедоступного и закрытого импорта пакетов

Помимо импорта пакетов в языке UML присутствует импорт элемента, который означает направленное отношение между импортирующим пространством имен и отдельным элементом пакета, которое позволяет ссылаться на этот элемент с использованием неквалифицированного имени. Также как и в случае импорта пакетов возможен общедоступный (стереотип <<import>>) и закрытый (стереотип <<access>>) импорт элемента. На рисунке 27 приведен пример импорта элементов, где пакет «Программа» импортирует элемент «Целое» из пакета «Типы» в общедоступном режиме, а элемент «Строка» – в закрытом.

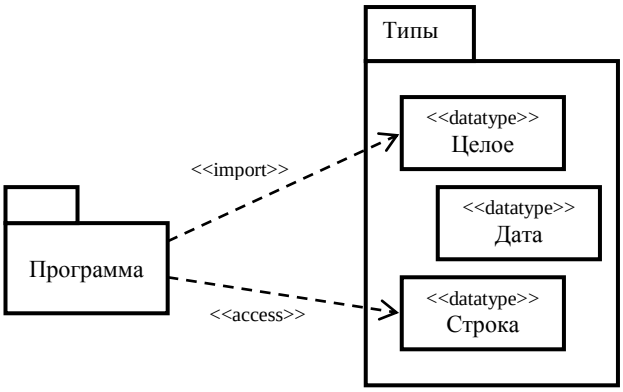

Рисунок 27 – Пример импорта элемента

Если неквалифицированное имя импортируемого элемента совпадает с именем любого другого элемента, находящегося в импортирующем пакете, то необходимо использовать квалифицированное имя элемента. Чтобы избежать этого неудобства можно вместо имени импортируемого элемента специфицировать его псевдоним, который в свою очередь не должен совпадать с именами других членов импортирующего пакета. Пример импорта элемента с псевдонимом приведен на рисунке 28.

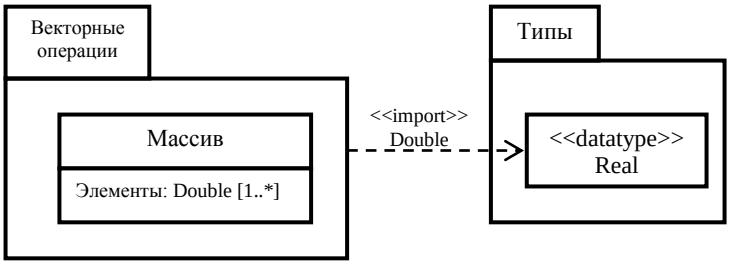

Рисунок 28 – Пример импорта элемента с псевдонимом

Слияние пакетов – направленное отношение между двумя пакетами, один из которых расширяет свое содержание посредством добавления содержимого другого пакета. Концептуально слияние пакетов можно рассматривать как операцию, которая на основе содержимого двух пакетов создает новый пакет, объединяющий содержимое используемых при слиянии пакетов. Отношение слияния пакетов является специализацией отношения зависимости, указываемой со стереотипом <<merge>>. Графическое обозначение отношения слияния приведено на рисунке 29.

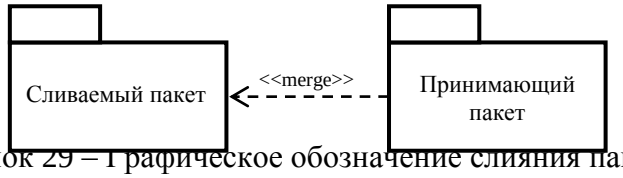

Рисунок 29 – Графическое обозначение слияния пакетов

При слиянии пакетов выделяют следующие понятия:

- сливаемый пакет – первый операнд слияния, т.е пакет, который сливается в принимающий пакет и который является целью стрелки слияния на диаграммах;
- принимающий пакет – второй операнд слияния, т.е. пакет концептуально содержит результаты слияния и который является источником стрелки слияния на диаграммах (этот термин используется, чтобы ссылаться на пакет и его содержимое до того, как будут выполнены преобразования слияния);
- результирующий пакет – пакет, который концептуально содержит результаты слияния (этот термин используется, чтобы ссылаться на пакет и его содержимое после того, как слияние будет выполнено).

Данная терминология предназначена для иллюстрации концептуального представления семантики слияния пакетов посредством схемы, приведенной на рисунке 30. На схеме показано, что собственные элементы пакетов A и B объединяются в пространство имен пакета B'.

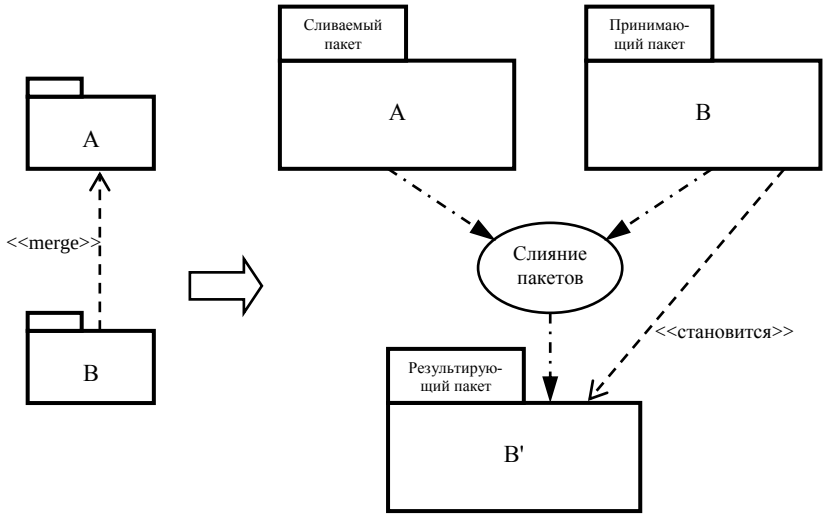

Рисунок 30 – Концептуальное представление семантики слияния пакетов

Общий принцип выполнения слияния состоит в следующем: любой элемент после слияния не должен обладать меньшими возможностями, чем до выполнения слияния. При выполнении слияния пакетов действуют следующие правила и ограничения:

- закрытые элементы пакета не сливаются;
- классы в принимающем пакете, имеющие такие же имена как и классы сливаемого пакета, становятся обобщением сливаемых классов;
- можно обращаться к исходным классам, используя квалифицированное имя с указанием оригинального пакета (имя пакета до слияния);
- классы, которые существуют только в принимающем или только в сливаемом пакете, остаются неизменными и добавляются в результирующий пакет;
- подпакеты внутри сливаемого пакета добавляются в результирующий пакет, если они не существуют в принимающем пакете. В противном случае, для этих пакетов выполняется операция слияния;
- импорты пакетов в сливаемом пакете становятся импортами пакетов в результирующем пакете (отношение обобщения для них не применяется);

– если импортируемые элементы конфликтуют с элементами в принимающем пакете, то элементы принимающего пакета имеют преимущество, а к импортируемым элементам необходимо обращаться, указывая квалифицированное имя.

В объектно-ориентированном анализе и проектировании диаграммы пакетов часто используются для представления архитектуры разрабатываемого ПО, на логическом уровне. Каждый пакет в этом случае представляет собой некоторую подсистему и содержит описание классов, реализующих эту подсистему, и отношений между ними. Также диаграммы пакетов могут применяться при моделировании функционального назначения разрабатываемого ПО на диаграммах вариантов использования для группировки непосредственно вариантов использования, с которыми взаимодействует определенный пользователь.

Ход работы

При выполнении данной лабораторной работы необходимо придерживаться трехуровневой модели архитектуры разрабатываемого программного приложения. Для этого в разрабатываемом приложении необходимо выделить три вида классов: сущности предметной области, классы управления и граничные классы.

В проекте создать диаграмму пакетов (рисунок 31) на которой разместить три пакета: уровень интерфейса, уровень бизнес-логики, уровень данных. Связдать созданные пакеты отношениями зависимости: уровень интерфейса зависит от уровня бизнес-логики, а уровень бизнес-логики зависит от уровня данных.

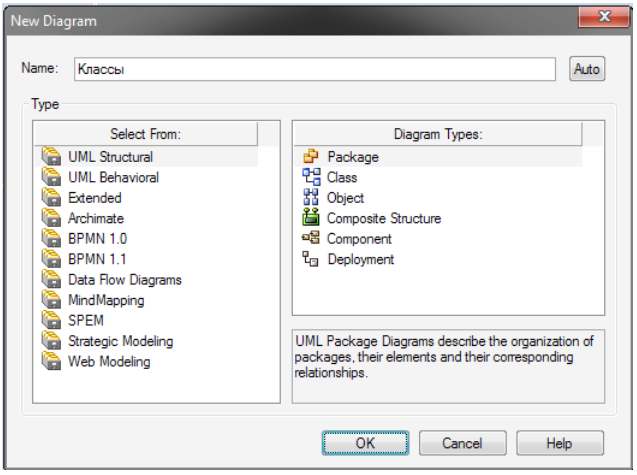

Рисунок 31 – Создание диаграммы пакетов

Для каждого соданного пакета создать диаграмму классов. Если при создании пакета был установлен флажок «Automatically add new diagram», то окно создания диаграммы для пакета появится автоматически. Если флажок не устанавливать, то создать диаграмму для пакета с помощью пункта «Add» контекстного меню выбранного пакета в Project Browser.

На диаграмме классов для пакета уровня интерфейса необходимо создать классы, реализующие интерфейс создаваемого ПО в соответствии с предыдущей лабораторной работой. При описании классов необходимо выбрать один из языков программирования, на котором будет «выполняться» релизация самого проекта. Установить язык можно в свойствах класса (рисунок 32). Выбор языка остается на усмотрение студента, хотя рекомендуется выбирать язык один из слежующих языков програмирования: C++, Java или C#.

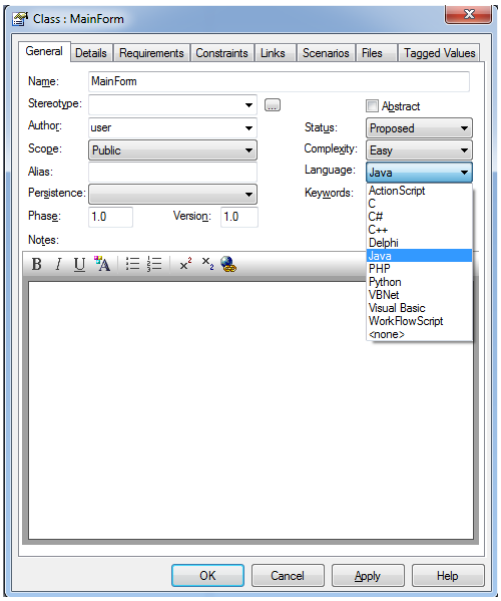

Рисунок 32 – Выбор языка программирования

На диаграмме уровня интерфейсов должны находиться только классы, реализующие графический интерфейс проектируемого приложения. Каждый класс – экранная форма, спроектированная в рамках предыдущей лабораторной работы. Атрибутами такого класса являются элементы управления (кнопки, текстовые поля и т. д.), присутствующие на экранной форме. Методами этого класса являются обработчики событий, происходящих на форме (нажатия кнопок, выбор флажков и т.д.). При моделировании классов графического интерфейса необходимо определиться с использованием какой графической библиотеки будет вестись разработка (AWT, Swing, .NET, MFC или другая) в зависимости от выбранного языка программирования.

На диаграмме классов уровня данных необходимо смоделировать классы, являющиеся предметными сущностями проектируемой информационной системы. Для классов должны быть определены атрибуты. Для каждого атрибута созданы геттеры и сеттеры. Допускается наличие и других методов. Между классами должны быть установлены стандартные для UML отношени: ассоциаации, зависимости, обобщения (желательно), агрегации и композиции (по возможности).

На диаграмме классов уровня бизнес-логики необходимо спроектировать классы управления, с помощью которых будет реализовываться функционал системы. Для идентификации классов уровня бизнес-логики обычно руководствуются диаграммой вариантов использования.

После разработки всех трех диаграмм классов необходимо выполнить автоматическую генерацию кода для спроектированных классов. Для этого необходимо выделить на открытой диаграмме все классы и вызвать команду «Code generation» в контекстном меню.