Лекция 7. Диаграммы классов(class diagram)

Диаграммы классов при моделировании объектно-ориентированных систем встречаются чаще других. На таких диаграммах отображается множество классов, интерфейсов, кооперации и отношений между ними. Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Кроме того, диаграммы классов составляют основу еще двух диаграмм – компонентов и развертывания.

Диаграмма классов может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы.

Компоненты диаграммы классов

Классом называется описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Например, класс "Стена" описывает объекты с общими свойствами: высотой, длиной, толщиной, и т.д. При этом конкретные стены будут рассматриваться как отдельные экземпляры класса «стена». У каждого класса есть имя, (простое или составное, к которому спереди добавлено имя пакета, в который входит класс). Имя класса в пакете должно быть уникальным. Класс реализует один или несколько интерфейсов.

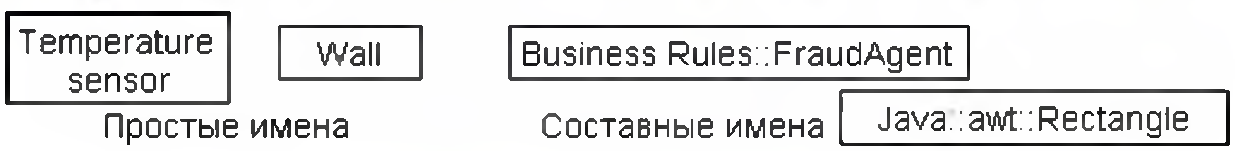

Рис. 40 Простые и составные имена

Атрибут – это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого класса. Класс может иметь любое число атрибутов или не иметь их вовсе. В языках высокого уровня, таких, как C++, Java, атрибуты соответствуют переменным, объявленным в классе. Например, у любой стены есть высота, ширина и толщина. Атрибуты представлены в разделе, расположенном под именем класса; при этом указываются их имена, и иногда начальное значение (рис. 41).

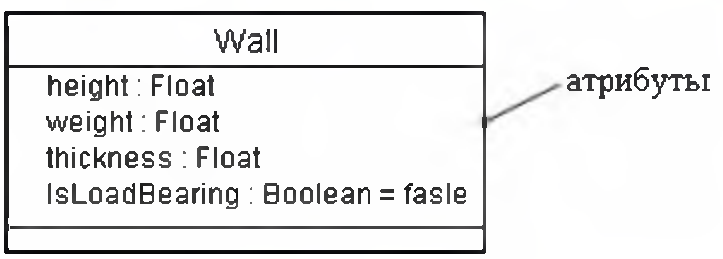

Рис. 41 Атрибуты и их класс

Операция – это некоторый сервис, который предоставляет экземпляр или объект класса по требованию своих клиентов (других объектов, в том числе и экземпляров данного класса). Класс может содержать любое число операций или не содержать их вовсе. В языках высокого уровня, таких, как C++, Java, операции соответствуют функциям, объявленным в классе. Операцию можно

систем встречаются чаще других. На таких диаграммах отображается множество классов, интерфейсов, кооперацій и отношений между ними. Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Кроме того, диаграммы классов составляют основу еще двух диаграмм – компонентов и развертывания.

Диаграмма классов может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы.

5.1 Компоненты диаграммы классов

Классом называется описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Например, класс “Стена” описывает объекты с общими свойствами: высотой, длиной, толщиной, и т.д. При этом конкретные стены будут рассматриваться как отдельные экземпляры класса «стена». У каждого класса есть имя, (простое или составное, к которому спереди добавлено имя пакета, в который входит класс). Имя класса в пакете должно быть уникальным. Класс реализует один или несколько интерфейсов.

Рис. 40 Простые и составные имена

Атрибут – это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого класса. Класс может иметь любое число атрибутов или не иметь их вовсе. В языках высокого уровня, таких, как C++, Java, атрибуты соответствуют переменным, объявленным в классе. Например, у любой стены есть высота, ширина и толщина. Атрибуты представлены в разделе, расположенном под именем класса; при этом указываются их имена, и иногда начальное значение (рис. 41).

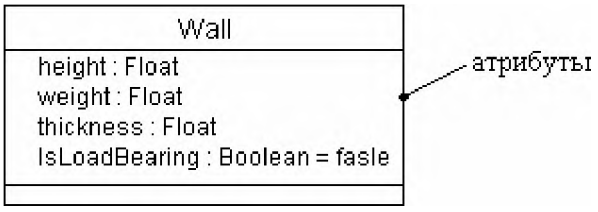

Рис. 41 Атрибуты и их класс

Операция – это некоторый сервис, который предоставляет экземпляр или объект класса по требованию своих клиентов (других объектов, в том числе и экземпляров данного класса). Класс может содержать любое число операций или не содержать их вовсе. В языках высокого уровня, таких, как C++, Java, операции соответствуют функциям, объявленным в классе. Операцию можно

описать более подробно, указав имена и типы параметров, их значения, принятые по умолчанию, а также тип возвращаемого значения (рис. 42).

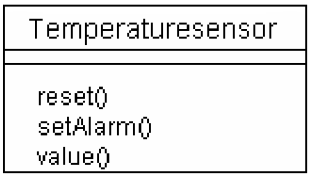

Рис. 42 Операции

При изображении класса необязательно сразу показывать все его атрибуты и операции. Их может быть много, однако для данного представления системы лишь небольшое подмножество атрибутов и операций имеет значение. В это случае класс сворачивают, и изображают только некоторые из атрибутов и операций, а дополнительные атрибуты или операции обозначают многоточием. Для лучшей организации списков атрибутов и операций можно снабдить каждую группу дополнительным описанием (стереотипами).

Обязанности класса – это своего рода контракт, которому он должен подчиняться. Атрибуты и операции являются свойствами, посредством которых выполняются обязанности класса. Например, класс FraudAgent (агент по предотвращению мошенничества), который встречается в приложениях по обработке кредитных карточек, отвечает за оценку платежных требований – законные, подозрительные или подложные (рис. 43). Моделирование классов лучше всего начинать с определения обязанностей сущностей, которые входят в словарь системы. Число обязанностей класса может быть произвольным, но на практике хорошо структурированный класс имеет по меньшей мере одну обязанность; с другой стороны, их не должно быть и слишком много. При уточнении модели обязанности класса преобразуются в совокупность атрибутов и операций, которые должны наилучшим образом обеспечить их выполнение.

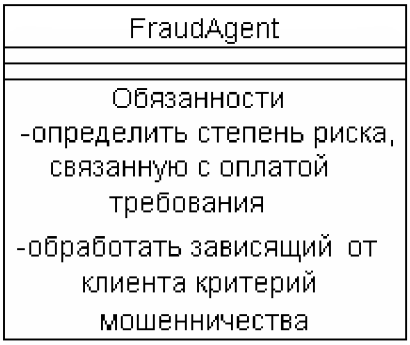

Рис. 43 Обязанности

Классы редко существуют автономно, они взаимодействуют между собой. Это значит, что, при моделировании системы, необходимо идентифицировать не только сущности, составляющие ее словарь, но и описать, как они соотносятся друг с другом. Существует основные три вида отношений между классами: *зависимости*, *обобщения, ассоциации* (рис. 44).

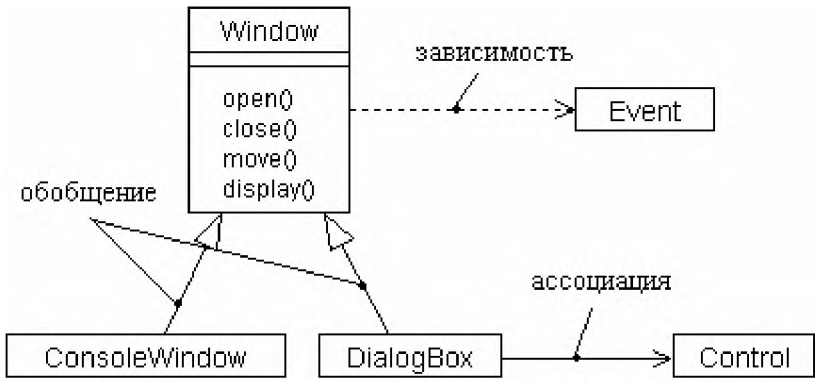

Рис. 44 Отношения

Кроме перечисленных отношений на диаграммах классов также применяются отношения агрегации и композиции.

Отношение агрегации применяется для представления системных взаимосвязей типа "часть-целое". Например, деление персонального компьютера на составные части: системный блок, монитор, клавиатуру и мышь (рис. 45).

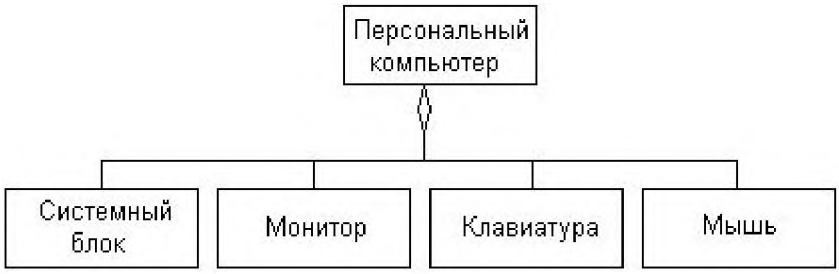

Рис. 45 Отношения агрегации

Отношение композиции является частным случаем агрегации. Оно служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т.е. с уничтожением целого уничтожаются и все его составные части. Например – окно интерфейса программы, состоящее из строки заголовка, кнопок управления размером, полос прокрутки, главного меню, рабочей области и строки состояния (рис. 46).

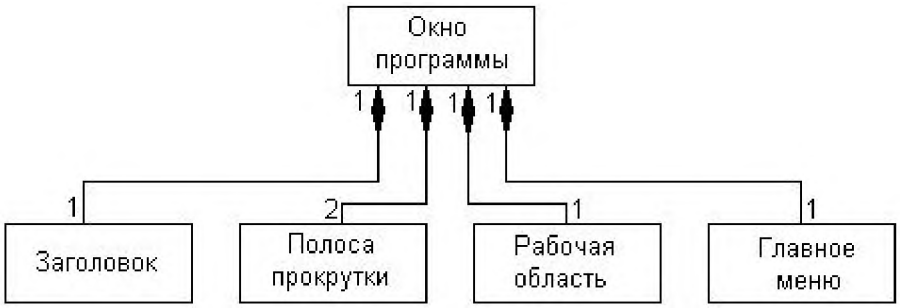

Рис. 46 Отношения композиции

Кроме описанных выше элементов, на диаграмме классов также могут присутствовать примечания, дополнения, стереотипы, помеченные значения,

ограничения.

Стереотипом называют расширение словаря UML, позволяющее создавать новые виды строительных блоков, аналогичные существующим, но специфичные для данной задачи. Стереотип представлен в виде имени, заключенного в кавычки и расположенного над именем другого элемента. С помощью стереотипа создаётся новый строительный блок, который напоминает существующий, но обладает новыми свойствами (рис. 47).

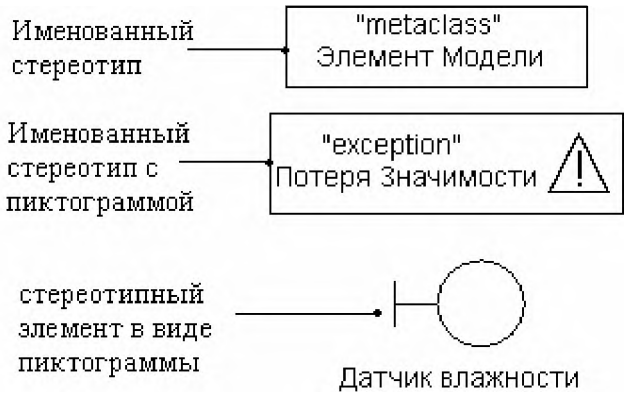

Рис. 47 Стереотипы

Помеченное значение позволяет включать новую информацию в спецификацию элемента. Например, при создании нескольких версий ПО необходимо отслеживать версию и автора какой-нибудь важной абстракции. Ни версия, ни автор не являются первичными концепциями UML, но их можно добавить к любому блоку, например, к классу, задавая для него новые помеченные значения (рис. 48).

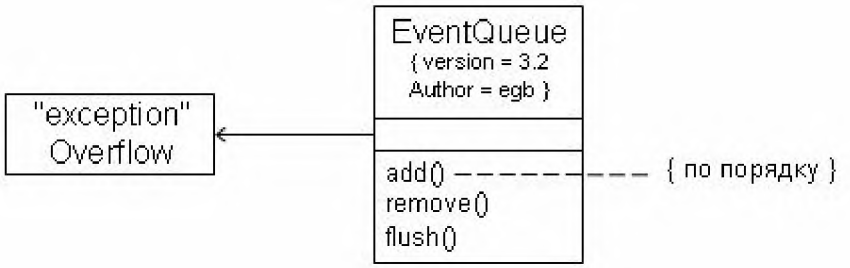

Рис. 48 Помеченное значение

Ограничение – это расширение семантики элемента UML, позволяющее создавать новые или изменять существующие правила (рис. 49).

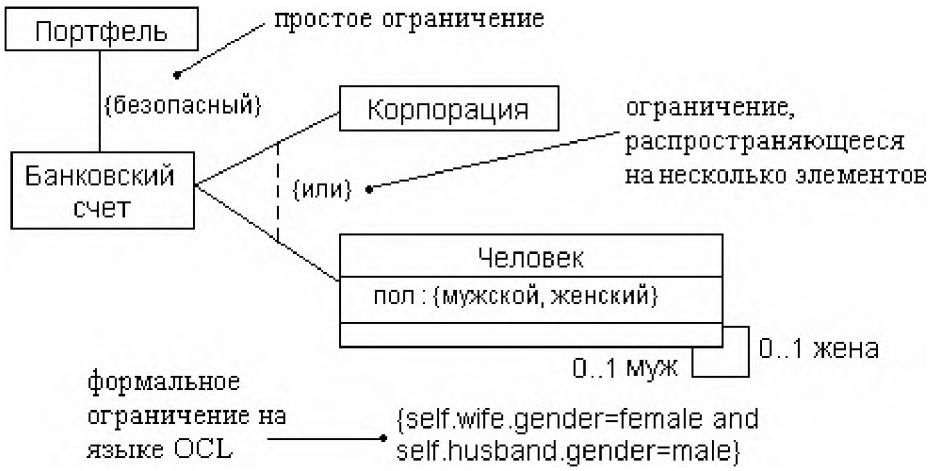

Рис. 49 Ограничения

Каждый элемент языка UML имеет свою семантику. С помощью ограничений можно создавать новую семантику или изменять существующие семантические правила. Например, на рис. 49 показано, что информация, передаваемая между классами “Портфель” и “Банковский счёт”, зашифрована. При моделировании систем реального времени часто используются временные и пространственные ограничения.

Лекция 7. Прямое и обратное проектирование

Основным результатом деятельности группы разработчиков являются не диаграммы, а программное обеспечение, поэтому модели и основанные на них реализации должны соответствовать друг другу с минимальными затратами по поддержанию синхронизации между ними. Чаще всего разработанные модели преобразуются в программный код. Хотя UML не определяет конкретного способа отображения на какой-либо объектно-ориентированный язык, он проектировался с учетом этого требования. В наибольшей степени это относится к диаграммам классов, содержание которых без труда отображается на такие известные объектно-ориентированные языки программирования, как Java, C++, ObjectPascal, Visual Basic и др.

Прямым проектированием (Forward engineering) называется процесс преобразования модели в код путем отображения на некоторый язык реализации. Процесс прямого проектирования приводит к потере информации, поскольку написанные на языке UML модели семантически богаче любого из существующих объектно-ориентированных языков. Фактически именно это различие и является основной причиной необходимости моделей. Некоторые структурные свойства системы, такие как кооперации, или ее поведенческие особенности, например взаимодействия, могут быть легко визуализированы в UML, но в чистом коде наглядность теряется.

При прямом проектировании диаграммы классов следует учитывать, что так как модели зависят от семантики выбранного языка программирования, вероятно, придется отказаться от использования некоторых возможностей UML. Например, язык UML допускает множественное наследование, а язык

программирования Smalltalk – только одиночное. В связи с этим можно запретить авторам моделей пользоваться множественным наследованием. Для специфицирования языка программирования применяются помеченные значения как на уровне индивидуальных классов (если нужна тонкая настройка), так и на более высоком уровне, например для пакетов или кооперации.

На рис. 50 изображена диаграмма классов, на которой проиллюстрирована реализация образца цепочки обязанностей. В данном примере представлены три класса: Client (Клиент), EventHandler (ОбработчикСобытий) и GUIEventHandler (ОбработчикСобытийГИП). Первые два из них являются абстрактными, а последний – конкретным. Класс EventHandler содержит обычную для данного образца операцию handleRequest (ОбработатьЗапрос), хотя в рассматриваемом случае было добавлено два скрытых атрибута.

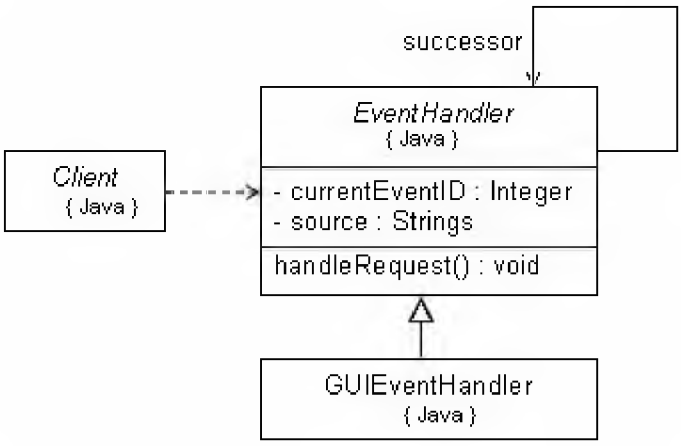

Рис. 50 Прямое проектирование

Определенное для каждого класса помеченное значение показывает, что они будут преобразованы в код на языке Java. Прямое проектирование в данном случае легко осуществимо с помощью специального инструмента. Так, для класса EventHandler будет сгенерирован следующий код:

```
public abstract class EventHandler {
  EventHandler successor;
  private Integer currentEventID;
  private String source;
  EventHandler()   {}
  public void handleRequest ()   {}
}
```

Обратным проектированием (Reverse engineering) называется процесс преобразования в модель кода, записанного на каком-либо языке программирования. В результате этого процесса вы получаете огромный объем информации, часть которой находится на более низком уровне детализации, чем необходимо для построения полезных моделей. В то же время обратное проектирование никогда не бывает полным. Как уже упоминалось, прямое проектирование ведет к потере информации, так что полностью восстановить модель на основе кода не удастся, если только инструментальные средства не

проектировании этих и иных классов, например запрос о необходимых предварительных знаниях перед зачислением студента на курс. Но это, скорее, бизнес-правила, а не операции по поддержанию целостности данных, поэтому лучше располагать их на более высоком уровне абстракции.

На рис. 52 представлен пример диаграммы классов структуры компании.

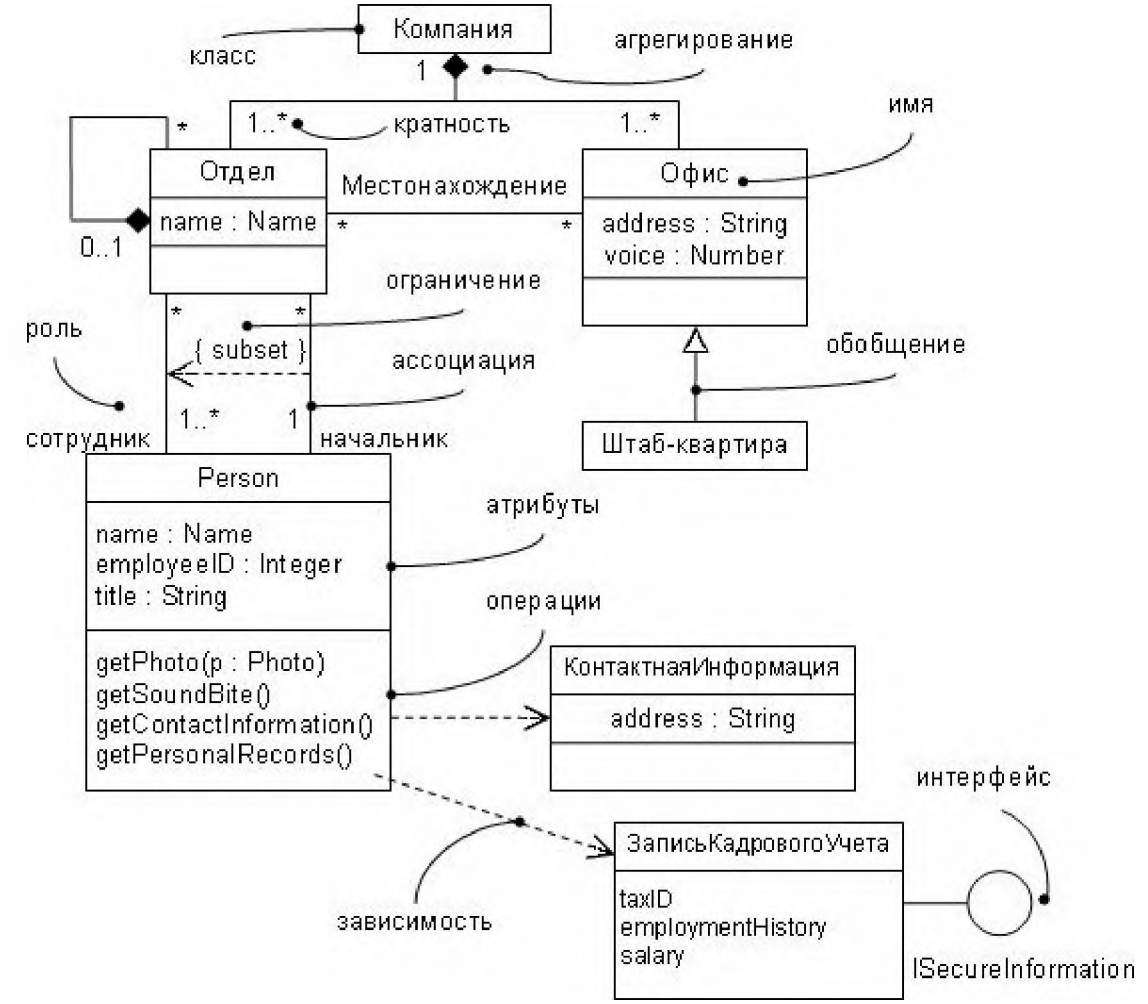

Рис. 52 Диаграмма классов