## 13. Объектно-ориентированные методы анализа и проектирования ПО. Основные принципы построения объектной модели. Основные элементы объектной модели.

http://vmk.ugatu.ac.ru/book/buch/ch02.htm

http://citforum.ru/programming/oop_rsis/glava2_2.shtml

Анализ внешних требований к проектируемой прикладной системе позволяет определить объекты и классы объектов, связанные с прикладной проблемой, которую должна решать эта система. Все классы должны быть осмыслены в рассматриваемой прикладной области; классов, связанных с компьютерной реализацией, как например список, стэк и т.п. на этом этапе вводить не следует.

Начать нужно с выделения возможных классов из письменной постановки прикладной задачи (технического задания и другой документации, предоставленной заказчиком). Следует иметь в виду, что это очень сложный и ответственный этап разработки, так как от него во многом зависит дальнейшая судьба проекта.

При определении возможных классов нужно постараться выделить как можно больше классов, выписывая имя каждого класса, который приходит на ум. В частности, каждому существительному, встречающемуся в предварительной постановке задачи, может соответствовать класс. Поэтому при выделении возможных классов каждому такому существительному обычно сопоставляется возможный класс.

Далее список возможных классов должен быть проанализирован с целью исключения из него *ненужных* классов. Такими классами являются:

- *избыточные классы:* если два или несколько классов выражают одинаковую информацию, следует сохранить только один из них;
- *нерелевантные* (не имеющие прямого отношения к проблеме) *классы*: для каждого имени возможного класса оценивается, насколько он необходим в будущей системе (оценить это часто бывает весьма непросто); нерелевантные классы исключаются;
- *нечетко определенные* (с точки зрения рассматриваемой проблемы) *классы*
- *атрибуты*: некоторым существительным больше соответствуют не классы, а атрибуты; такие существительные, как правило, описывают свойства объектов (например, имя, возраст, вес, адрес и т.п.);
- *операции*: некоторым существительным больше соответствуют не классы, а имена операций (например, телефонный_вызов вряд ли означает какой-либо класс);
- *роли*: некоторые существительные определяют имена ролей в объектной модели (например, владелец, водитель, начальник, служащий; все эти имена связаны с ролями в различных зависимостях объектов класса человек);
- *реализационные конструкции*: именам, больше связанным с программированием и компьютерной аппаратурой, не следует на данном этапе сопоставлять классов, так как они не отражают особенностей проектируемой прикладной системы; примеры таких имен: подпрограмма, процесс, алгоритм, прерывание и т.п.

После исключения имен всех ненужных (лишних) возможных классов будет получен предварительный список классов, составляющих проектируемую систему.

Отдельные слова имеют слишком много интерпретаций. Поэтому необходимо в самом начале проектирования подготовить *словарь данных*, содержащий четкие и недвусмысленные определения всех объектов (классов), атрибутов, операций, ролей и других сущностей, рассматриваемых в проекте. Без такого словаря обсуждение проекта с коллегами по разработке и заказчиками системы не имеет смысла, так как каждый может по-своему интерпретировать обсуждаемые термины.

### 2.2.3. Определение зависимостей

На следующем этапе построения объектной модели определяются зависимости между классами. Прежде всего из классов исключаются атрибуты, являющиеся явными ссылками на другие классы; такие атрибуты заменяются зависимостями. Смысл такой замены в том, что зависимости представляют собой абстракцию того же уровня, что и классы, и потому не оказывают непосредственного влияния на будущую реализацию (ссылка на класс лишь один из способов реализации зависимостей).

Аналогично тому, как имена возможных классов получались из существительных, встречающихся в предварительной постановке прикладной задачи, имена возможных зависимостей могут быть получены из *глаголов* или *глагольных оборотов*, встречающихся в указанном документе. Так обычно описываются: физическое положение (следует_за, является_частью, содержится_в), направленное действие (приводит_в_движение), общение (разговаривает_с), принадлежность (имеет, является_частью) и т.п.

Затем следует убрать ненужные или неправильные зависимости, используя следующие критерии:

- *зависимости между исключенными классами* должны быть исключены, либо переформулированы в терминах оставшихся классов
- *нерелевантные зависимости* и зависимости, связанные с реализацией, должны быть исключены
- *действия:* зависимость должна описывать структурные свойства прикладной области, а не малосущественные события
- *тренарные зависимости:* большую часть зависимостей между тремя или большим числом классов можно разложить на несколько бинарных зависимостей, используя в случае необходимости квалификаторы в некоторых (очень редких) случаях такое разложение осуществить не удается; например, тренарная зависимость "Профессор читает курс в аудитории 628" не может быть разложена на бинарные без потери информации;
- *производные зависимости:* нужно исключать зависимости, которые можно выразить через другие зависимости, так как они избыточны при исключении избыточных (производных) зависимостей нужно быть особенно осторожным, так как не все дублирующие одна другую зависимости между классами избыточны; в некоторых случаях другие зависимости позволяют установить только существование еще одной производной зависимости, но не позволяют установить кратность этой зависимости; например, в случае, представленном на рисунке 2.36, фирма имеет много служащих и владеет многими компьютерами; каждому служащему предоставлено для персонального использования несколько компьютеров, кроме того, имеются компьютеры общего пользования; кратность зависимости предоставлен_для_использования не может быть выведена из зависимостей служит и владеет; хотя производные зависимости и не добавляют новой информации, они часто бывают удобны; в этих случаях их можно указывать на диаграмме, пометив косой чертой.

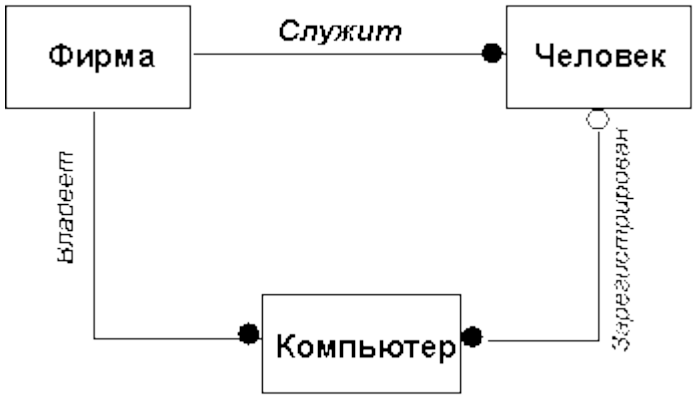


*Рис. 2.36. Неизбыточные зависимости*

Удалив избыточные зависимости, нужно уточнить семантику оставшихся зависимостей следующим образом:

- *неверно названные зависимости:* их следует переименовать, чтобы смысл их стал понятен ;
- *имена ролей:* нужно добавить имена ролей там, где это необходимо; имя роли описывает роль, которую играет соответствующий класс в данной зависимости с точки зрения другого класса, участвующего в этой зависимости; если имя роли ясно из имени класса, его можно не указывать;
- *квалификаторы:* добавляя квалификаторы там, где это необходимо, мы вносим элементы контекста, что позволяет добиться однозначной идентификации объектов; квалификаторы позволяют также упростить некоторые зависимости, понизив их кратность;
- *кратность:* необходимо добавить обозначения кратности зависимостей; при этом следует помнить, что кратность зависимостей может меняться в процессе дальнейшего анализа требований к системе;
- *неучтенные зависимости* должны быть выявлены и добавлены в модель.

### 2.2.4. Уточнение атрибутов

На следующем этапе уточняется система атрибутов: корректируются атрибуты классов, вводятся, в случае необходимости, новые атрибуты. Атрибуты выражают свойства объектов рассматриваемого класса, либо определяют их текущее состояние.

Атрибуты обычно соответствуют существительным; например цвет_автомобиля (свойство объекта), позиция_курсора (состояние объекта). Атрибуты, как правило, слабо влияют на структуру объектной модели.

Не следует стремиться определить как можно больше атрибутов: большое количество атрибутов усложняет модель, затрудняет понимание проблемы. Необходимо вводить только те атрибуты, которые имеют отношение к проектируемой прикладной системе, опуская случайные, малосущественные и производные атрибуты.

Наряду с атрибутами объектов необходимо ввести и атрибуты зависимостей между классами (связей между объектами).

При уточнении атрибутов руководствуются следующими критериями:

- *Замена атрибутов на объекты*. Если наличие некоторой сущности важнее, чем ее значение, то это объект, если важнее значение, то это атрибут: например, начальник - это объект (неважно, кто именно начальник, главное, чтобы кто-то им был), зарплата - это атрибут (ее значение весьма существенно); город - всегда объект, хотя в некоторых случаях может показаться, что это атрибут (например, город как часть адреса фирмы); в тех случаях, когда нужно, чтобы город был атрибутом, следует определить зависимость (скажем, находится) между классами фирма и город.
- *Квалификаторы*. Если значение атрибута зависит от конкретного контекста, его следует сделать квалификатором.
- *Имена*. Именам обычно лучше соответствуют квалификаторы, чем атрибуты объектов; во всех случаях, когда имя позволяет сделать выбор из объектов некоторого множества, его следует сделать квалификатором.
- *Идентификаторы*. Идентификаторы объектов связаны с их реализацией. На ранних стадиях проектирования их не следует рассматривать в качестве атрибутов.
- *Атрибуты связей*. Если некоторое свойство характеризует не объект сам по себе, а его связь с другим объектом (объектами), то это атрибут связи, а не атрибут объекта.
- *Внутренние значения*. Атрибуты, определяющие лишь внутреннее состояние объекта, незаметное вне объекта, следует исключить из рассмотрения.
- *Несущественные детали*. Атрибуты, не влияющие на выполнение большей части операций, рекомендуется опустить.

### 2.2.5. Организация системы классов, используя наследование

Далее необходимо постараться найти суперклассы для введенных классов. Это полезно, так как проясняет структуру модели и облегчает последующую реализацию.

### 2.2.6. Дальнейшее исследование и усовершенствование модели

Лишь в очень редких случаях построенная объектная модель сразу же оказывается корректной. Модель должна быть исследована и отлажена. Некоторые ошибки могут быть найдены при исследовании модели без компьютера, другие - при ее интерпретации совместно с динамической и функциональной моделями на компьютере (эти модели строятся после того, как объектная модель уже построена).

Здесь мы рассмотрим приемы бескомпьютерного поиска и исправления ошибок в объектной модели. В их основе лежат внешние признаки, по которым можно находить ошибки в модели; эти признаки могут быть объединены в следующие группы.

***Признаки пропущенного объекта (класса):***

- несимметричности связей и обобщений (наследований); для исправления ошибки необходимо добавить пропущенные классы;
- несоответствие атрибутов и операций у класса; для исправления ошибки необходимо расщепить класс на несколько других классов, так чтобы атрибуты и операции новых классов соответствовали друг другу;
- обнаружена операция, не имеющая удовлетворительного целевого класса; для исправления ошибки необходимо добавить пропущенный целевой класс;
- обнаружено несколько зависимостей с одинаковыми именами и назначением; для исправления ошибки необходимо сделать обобщение и добавить пропущенный суперкласс.

***Признаки ненужного (лишнего) класса:***

- нехватка атрибутов, операций и зависимостей у некоторого класса; для исправления ошибки необходимо подумать, не следует ли исключить такой класс.

***Признаки пропущенных зависимостей:***

- отсутствуют пути доступа к операциям; для исправления ошибки необходимо добавить новые зависимости, обеспечивающие возможности обслуживания соответствующих запросов.

***Признаки ненужных (лишних) зависимостей:***

- избыточная информация в зависимостях; для исправления ошибки необходимо исключить зависимости, не добавляющие новой информации, или пометить их как производные зависимости;
- не хватает операций, пересекающих зависимость; для исправления ошибки необходимо подумать, не следует ли исключить такую зависимость.

***Признаки неправильного размещения зависимостей:***

- имена ролей слишком широки или слишком узки для их классов; для исправления ошибки необходимо переместить зависимость вверх или вниз по иерархии классов.

***Признаки неправильного размещения атрибутов:***

- нет необходимости доступа к объекту по значениям одного из его атрибутов; для исправления ошибки необходимо рассмотреть нужно ли ввести квалифицированную зависимость.