2. Основные понятия объектно-ориентированного подхода. Понятие инкапсуляции

Существует 3 принципа ООП: инкапсуляция, наследование, полиморфизм. Важным элементом ООП является абстракция. Остановимся на инкапсуляции.

Инкапсуляция — это механизм, который связывает код и данные, которыми он манипулирует. Один из возможных способов представления инкапсуляции — представление в виде защитной оболочки, которая предохраняет код и данные от произвольного доступа другого кода, находящегося снаружи оболочки. Доступ к коду и данным, находящимся внутри оболочки, строго контролируется тщательно определенным интерфейсом. Основа инкапсуляции — класс.



Пустые окружности — общедоступные переменные (не рекомендуется)

Пустые треугольники - общедоступные методы

Закрашенные треугольники — частные методы

Закрашенные окружности — приватные переменные экземпляра

48. Содержание основных программных документов. (из интернета)



49.Проектирование модульной структуры .

Первое, что необходимо сделать, начиная этап разработки системы, определить ее разбиение на некоторое количество компонентов - модулей. Модуль не является ни объектом, ни функцией; модуль - это набор (пакет) классов и отдельных объектов, подсистем, зависимостей, операций, событий и ограничений, которые взаимосвязаны и имеют достаточно хорошо определенный и по возможности небольшой интерфейс с другими модулями. Часто модуль включает одну подсистему, являясь ее реализацией. Модуль (подсистема) обычно определяется через службы, которые он обеспечивает. Службой называется набор взаимосвязанных функций, которые совместно обеспечивают какую-либо функциональность, например, выполнение ввода-вывода, отрисовку картинок, выполнение арифметических действий. Подсистема определяет согласованный способ рассмотрения одной из сторон прикладной задачи, для решения которой разрабатывается рассматриваемая система. Например, система файлов - подсистема операционной системы; она обеспечивает набор взаимосвязанных абстрактных операций, которые в большой степени (но не полностью) независимы от абстрактных операций, обеспечиваемых другими подсистемами. Эта подсистема может быть реализована в виде отдельного модуля.

Как уже отмечалось, каждая подсистема имеет хорошо определенный (внешний) интерфейс с остальной частью системы (другими подсистемами). Этот интерфейс определяет форму всех взаимодействий с подсистемой и все потоки данных через ее границы, но не специфицирует внутреннюю структуру и внутреннее окружение подсистемы, а также особенности ее реализации. Поэтому каждая подсистема может разрабатываться независимо от остальных подсистем.

Подсистемы должны определяться таким образом, чтобы большая часть взаимодействий оставалась внутри подсистем, для уменьшения глобальных потоков данных и сокращения зависимостей между подсистемами. Подсистем должно быть не очень много (в пределах десятка). Некоторые подсистемы могут быть в свою очередь подразделены на подсистемы.

Две подсистемы могут взаимодействовать друг с другом либо как клиент и поставщик (клиент-сервер), либо как равноправные партнеры (сопрограммы). В первом случае клиент вызывает сервер, который выполняет некоторый запрос клиента и возвращает результат; клиент должен знать интерфейс сервера, но сервер может не знать интерфейсов клиента, так как все взаимодействия инициируются клиентом. В случае сопрограммного взаимодействия обе подсистемы вызывают друг друга. Обращение подсистемы к другой подсистеме не обязательно связано с немедленным получением ответа. Сопрограммное взаимодействие является более сложным, так как обе подсистемы должны знать интерфейсы друг друга. Поэтому нужно стараться, чтобы большая часть подсистем взаимодействовала как клиент и сервер.

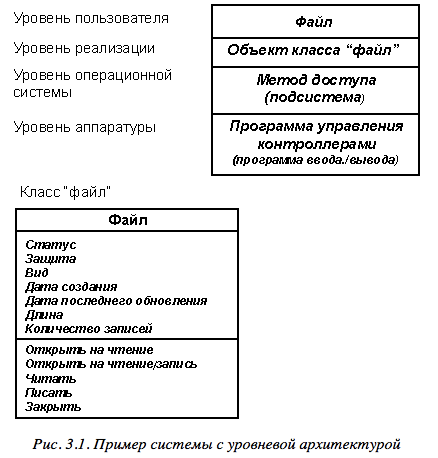
Подсистемы (и реализующие их модули) могут образовывать в системе уровни, либо разделы.

Уровни

Уровневая система может рассматриваться как упорядоченное множество виртуальных миров, каждый из которых построен на основе понятий, поддерживаемых его подсистемами; подсистемы одного уровня обеспечивают реализацию подсистем следующего уровня. Объекты каждого уровня могут быть независимыми, хотя нередко объекты разных уровней могут соответствовать друг другу. Каждая подсистема знает о подсистемах более низких уровней и ничего не знает о более высоких уровнях. Зависимость клиент-сервер существует между более верхним (клиент) и более нижними уровнями (серверы). При этом каждый уровень может иметь свой собственный набор классов и операций. Каждый уровень реализуется через операции объектов и подсистем более нижних уровней. Уровневые архитектуры бывают двух видов: открытые и замкнутые. В замкнутой архитектуре каждый уровень строится на базе непосредственно следующего за ним уровня. Это сокращает зависимости между уровнями и упрощает внесение изменений. В открытой архитектуре каждый уровень строится на базе всех следующих за ним уровней. Это уменьшает потребность в переопределении операций на каждом уровне и приводит к более эффективному и компактному коду. Однако открытая архитектура не удовлетворяет принципу упрятывания информации, поскольку изменения в какой-либо подсистеме могут потребовать соответствующих изменений в подсистемах более высоких уровней.

Обычно лишь подсистемы самого верхнего и самого нижнего уровней могут быть выведены из постановки задачи: самый верхний уровень - это требуемая система, а самый нижний уровень - это доступные ресурсы: аппаратура, операционная система, имеющиеся библиотеки. Промежуточные уровни вводятся разработчиком системы.

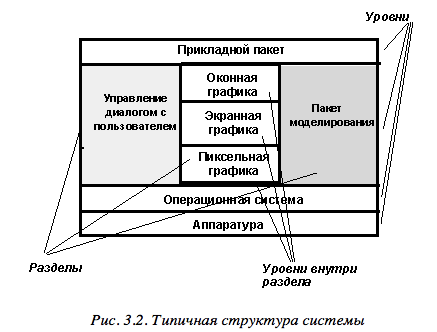
Система с уровневой архитектурой при переносе на другую платформу требует переписывания только одного (самого нижнего) уровня. Пример системы с уровневой архитектурой представлен на рисунке 3.1.



Разделы

Разделы подразделяют систему на несколько независимых или слабо связанных модулей (подсистем), каждая из которых обеспечивает один из видов услуг. Например, операционная система компьютера включает систему файлов, управление процессами, управление виртуальной памятью и управление устройствами.

Обычно система подразделяется на модули (подсистемы) с использованием обоих способов разбиения в самых разных комбинациях: уровни делятся на разделы, разделы содержат в себе уровни. Пример такой структуры системы показан на рисунке 3.2.





Топология системы

Когда все модули и подсистемы всех уровней названы, необходимо показать информационные потоки между модулями и подсистемами, построив ДПД (примеры см. в п. 3.1.8). Это позволит понять топологию системы. Топология системы определяется совокупностью потоков информации в системе; например, у компилятора конвейерная топология (рисунок 3.3), можно представить себе топологию звезды (рисунок 3.4) и т.п.; нужно стремиться, чтобы топология системы была как можно проще.

(http://citforum.ru/programming/oop\_rsis/glava3\_1\_1.shtml)

55.CASE-технологии разработки ПО

CASE (англ. Computer-Aided Software Engineering) — набор инструментов и методов программной инженерии для проектирования программного обеспечения, который помогает обеспечить высокое качество программ, отсутствие ошибок и простоту в обслуживании программных продуктов. Также под CASE понимают совокупность методов и средств проектирования информационных систем с использованием CASE-инструментов.

Средства автоматизации разработки программ (CASE-средства) — инструменты автоматизации процессов проектирования и разработки программного обеспечения для системного аналитика, разработчика ПО и программиста. Первоначально под CASE-средствами понимались только инструменты для упрощения наиболее трудоёмких процессов анализа и проектирования, но с приходом стандарта ISO/IEC 14102 CASE-средства стали определять как программные средства для поддержки процессов жизненного цикла ПО.

Общая информация.

Основной целью CASE-технологии является разграничение процесса проектирования программных продуктов от процесса кодирования и последующих этапов разработки, максимально автоматизировать процесс разработки. Для выполнения поставленной цели CASE-технологии используют два принципиально разных подхода к проектированию: структурный и объектно-ориентированный.

Структурный подход предполагает декомпозицию (разделение) поставленной задачи на функции, которые необходимо автоматизировать. В свою очередь, функции также разбиваются на подфункции, задачи, процедуры. В результате получается упорядоченная иерархия функций и передаваемой информацией между функциями.

Структурный подход подразумевает использование определенных общепринятых методологий при моделировании различных информационных систем:

SADT (Structured Analysis and Design Technique);

DFD (Data Flow Diagrams);

ERD (Entity-Relationship Diagrams).

Существует 3 основных типа моделей, используемых при структурном подходе:

Функциональные модели;

Информационные модели;

Динамические модели.

Программные средства для реализации структурного подхода:

BPWin;

ERWin;

Business Studio;

IBM WEBsphere business modeler;

Sybase PowerDesigner;

Основным инструментом объектно-ориентированного подхода является язык UML — унифицированный язык моделирования, который предназначен для визуализации и документирования объектно-ориентированных систем с ориентацией их на разработку программного обеспечения. Данный язык включает в себя систему различных диаграмм, на основании которых может быть построено представление о проектируемой системе.

Примерами программных средств, реализующих данный подход, являются:

Rational Rose;

ARIS.

Классификация[править | править исходный текст]

В функции CASE входят средства анализа, проектирования и программирования программных средств, проектирования интерфейсов, документирования и производства структурированного кода на каком-либо языке программирования.[4]

CASE-инструменты классифицируются по типам и категориям.

Классификация по типам отражает функциональную ориентацию средств на те или иные процессы жизненного цикла разработки программного обеспечения, и, в основном, совпадают с компонентным составом крупных интегрированных CASE-систем, и включает следующие типы:

средства анализа — предназначены для построения и анализа модели предметной области;

средства проектирования баз данных;

средства разработки приложений;

средства реинжиниринга процессов;

средства планирования и управления проектом;

средства тестирования;

средства документирования.

Классификация по категориям определяет степень интегрированности по выполняемым функциям и включают — отдельные локальные средства, решающие небольшие автономные задачи, набор частично интегрированных средств, охватывающих большинство этапов жизненного цикла и полностью интегрированных средств, охватывающий весь жизненный цикл информационной системы и связанных общим репозиторием.

Типичными CASE-инструментами являются:

инструменты управления конфигурацией;

инструменты моделирования данных;

инструменты анализа и проектирования;

инструменты преобразования моделей;

инструменты редактирования программного кода;

инструменты рефакторинга кода;

генераторы кода;

инструменты для построения UML-диаграмм. (http://ru.wikipedia.org/wiki/CASE)