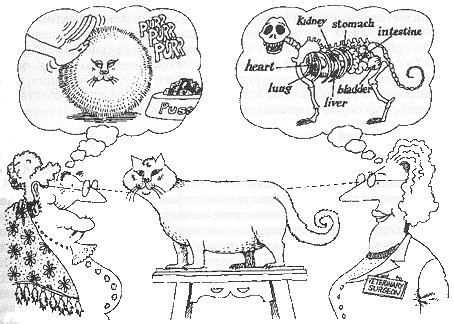
****ОСНОВНІ ПРИНЦИПИ ООП****

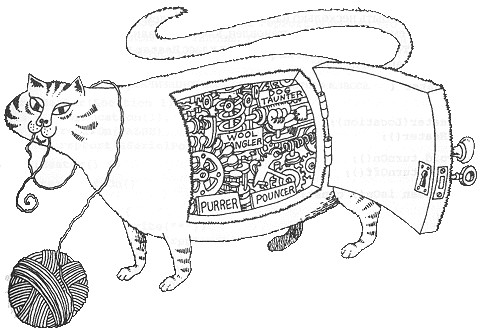
* **абстрагування**
* **інкапсуляція**
* **модульність**
* **ієрархія**
* **Додаткові принципи:**
* **типізація**
* **паралелізм**
* **стійкість**

Абстрагування **- це процес виділення найбільш істотних характеристик деякого об'єкта, що відрізняють його від усіх інших видів об'єктів, важливих з точки зору подальшого розгляду та аналізу, та ігнорування менш важливих або незначних деталей.**

**Об'єкти і класи - основні абстракції предметної області.**

Абстракція фокусується на суттєвих з точки зору спостерігача характеристиках об'єкта.

Інкапсуляція **- це процес відділення один від одного елементів об'єкта, що визначають його пристрій і поведінку; інкапсуляція служить для того, щоб ізолювати контрактні зобов'язання абстракції від їх реалізації.**



Інкапсуляція приховує деталі реалізації об'єкта.

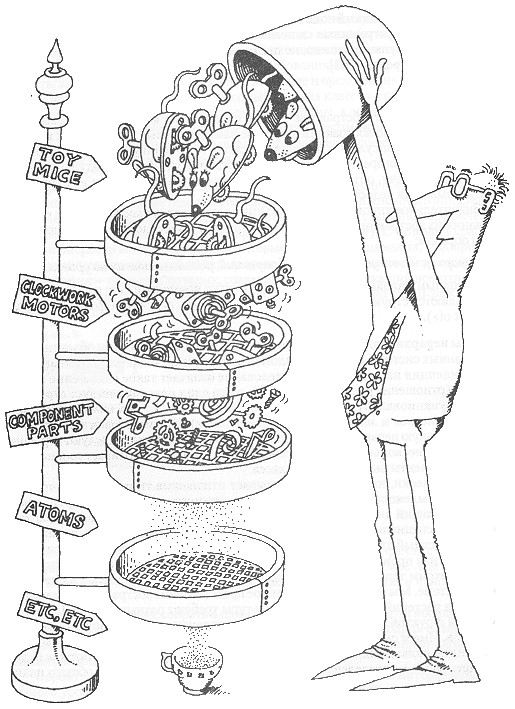
Модульність - це властивість системи, пов'язана з можливістю її декомпозиції на ряд внутрішньо сильно зчеплених, але слабо пов'язаних між собою підсистем (частин).

Модульність знижує складність системи, дозволяючи виконувати незалежну розробку її окремих частин.



Модульность позволяет хранить абстракции раздельно.

**Иерархия —**это упорядочение абстракций, расположение их по уровням.

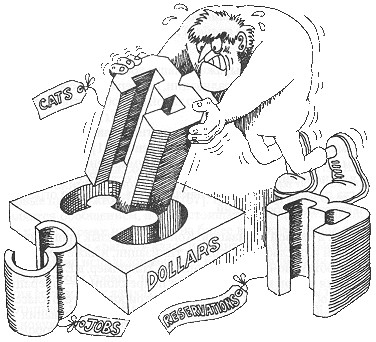


Абстракции образуют иерархию.

**Типизация —**способ защититься от использования объектов одного класса вместо другого, или, по крайней мере, управлять таким использованием.

**Тип —**точная характеристика некоторой совокупности однородных объектов, включающая структуру и поведение.

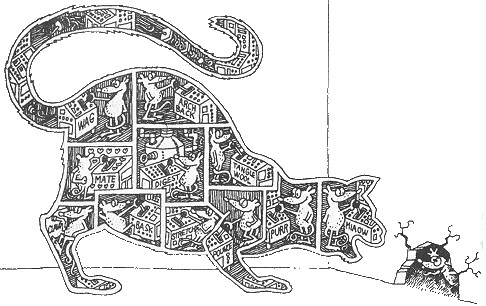
При строгой типизации (например, в языке Оберон) запрещается использование объектов неверного типа, требуется явное преобразование к нужному типу. При менее строгой типизации такого рода запреты ослаблены. В частности, допускается полиморфизм — многозначность имен. Одно из проявлений полиморфизма, использование объект подтипа (наследника) в роли объекта супертипа (предка).



Строгая типизация предотвращает смешивание абстракций.

**Параллелизм —**это свойство, отличающее активные объекты от пассивных.

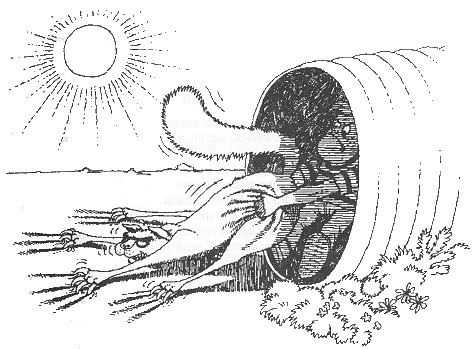
**Параллелизм** — наличие в системе нескольких потоков управления одновременно. Объект может быть активен, т. е. может порождать отдельный поток управления. Различные объекты могут быть активны одновременно.



Параллелизм позволяет различным объектам действовать одновременно.

**Сохраняемость (устойчивость) —**способность объекта существовать во времени, переживая породивший его процесс, и (или) в пространстве, перемещаясь из своего первоначального адресного пространства.

**Устойчивость** — способность объекта сохранять свое существование во времени и/или пространстве (адресном, в частности при перемещении между узлами вычислительной системы). В частности, устойчивость объектов может быть обеспечена за счет их хранения в базе данных.



Сохраняемость поддерживает состояние и класс объекта в пространстве и во времени.

****Основные понятия объектно-ориентированного подхода или элементы объектной модели****

“ Объект в ООП**—**это сущность, способная сохранять свое состояние (информацию) и обеспечивающая набор операций (поведение) для проверки и изменения этого состояния. ”

**Объект**— осязаемая сущность (tangible entity) — предмет или явление (процесс), имеющие четко выраженные границы, индивидуальность и поведение.

Любой объект обладает состоянием, поведением и индивидуальностью.

**Состояние**объекта определяется значениями его свойств (атрибутов) и связями с другими объектами, оно может меняться со временем.

**Поведение**определяет действия объекта и его реакцию на запросы от других объектов. Поведение представляется с помощью набора сообщений, воспринимаемых объектом (операций, которые может выполнять объект).

**Индивидуальность —**это свойства объекта, отличающие его от всех других объектов.

Структура и поведение схожих объектов определяют общий для них класс.

Объект в JavaScript создаётся с помощью функции Object.create. Эта функция из родителя и опционального набора свойств создаёт новую сущность. Пока что мы не будем беспокоиться о параметрах.

**Прототип —**это объект-образец, по образу и подобию которого создаются другие объекты. Объекты-копии могут сохранять связь с родительским объектом, автоматически наследуя изменения в прототипе; эта особенность определяется в рамках конкретного языка.

**Класс —**это множество объектов, связанных общностью свойств, поведения, связей и семантики. Любой объект является экземпляром класса. Определение классов и объектов — одна из самых сложных задач объектно-ориентированного проектирования.

**Класс (class)** — это группа данных и методов(функций) для работы с этими данными. Это шаблон. Объекты с одинаковыми свойствами, то есть с одинаковыми наборами переменных состояния и методов, образуют класс.

**Конструктор класса —**специальный блок инструкций, вызываемый при создании объекта.

var s = new String();

**Деструктор —**специальный метод класса, служащий для деинициализации объекта (например освобождения памяти).

**Атрибут —**поименованное свойство класса, определяющее диапазон допустимых значений, которые могут принимать экземпляры данного свойства. Атрибуты могут быть скрыты от других классов, это определяет видимость атрибута: рublic (общий, открытый); private (закрытый, секретный); protected (защищенный).

Требуемое поведение системы реализуется через взаимодействие объектов. Взаимодействие объектов обеспечивается механизмом пересылки сообщений. Определенное воздействие одного объекта на другой с целью вызвать соответствующую реакцию называется операцией или посылкой сообщения. Сообщение может быть послано только вдоль соединения между объектами. В терминах программирования соединение между объектами существует, если один объект имеет ссылку на другой.

**Дескриптор —**это атрибут объекта со связанным поведением (англ. binding behavior), т.е. такой, чьё поведение при доступе переопределяется методами протокола дескриптора.

**Операция —**это услуга, которую можно запросить у любого объекта данного класса. Операции реализуют поведение экземпляров класса. Описание операции включает четыре части:имя; список параметров; тип возвращаемого значения; видимость.Реализация операции называется методом.

**Метод** — это функция или процедура, принадлежащая какому-то классу или объекту.

Различают простые методы и статические методы (методы класса):

простые методы имеют доступ к данным объекта (конкретного экземпляра данного класса),

статические методы не имеют доступа к данным объекта и для их использования не нужно создавать экземпляры (данного класса).

Методы предоставляют интерфейс, при помощи которого осуществляется доступ к данным объекта некоторого класса, тем самым, обеспечивая инкапсуляцию данных.

В зависимости от того, какой уровень доступа предоставляет тот или иной метод, выделяют:

открытый (public) интерфейс — общий интерфейс для всех пользователей данного класса;

защищённый (protected) интерфейс — внутренний интерфейс для всех наследников данного класса;

закрытый (private) интерфейс — интерфейс, доступный только изнутри данного класса.

Такое разделение интерфейсов позволяет сохранять неизменным открытый интерфейс, но изменять внутреннюю реализацию.

**Полиморфизм —**способность скрывать множество различных реализаций под единственным общим именем или интерфейсом.

Понятие полиморфизма может быть интерпретировано, как способность объекта принадлежать более чем одному типу.

**Интерфейс —**это совокупность операций, определяющих набор услуг класса или компонента. Интерфейс не определяет внутреннюю структуру, все его операции открыты.

**Компонент —**это относительно независимая и замещаемая часть системы, выполняющая четко определенную функцию в контексте заданной архитектуры.

Компонент представляет собой физическую реализацию проектной абстракции и может быть: компонентом исходного кода (cpp-шник); компонентом времени выполнения (dll, ActiveX и т. п.); исполняемый компонентом (exe-шник). Компонент обеспечивает физическую реализацию набора интерфейсов. Компонентная разработка (component-based development) представляет собой создание программных систем, состоящих из компонентов (не путать с объектно-ориентированным программированием (ООП).

Компонентная разработка — технология, позволяющая объединять объектные компоненты в систему.

**Пакет —**это общий механизм для организации элементов в группы. Это элемент модели, который может включать другие элементы. Каждый элемент модели может входить только в один пакет.

Пакет является:

-средством организации модели в процессе разработки, повышения ее управляемости и читаемости;

-единицей управления конфигурацией.

**Подсистема —**это комбинация пакета (может включать другие элементы модели) и класса (обладает поведением). Подсистема реализует один или более интерфейсов, определяющих ее поведение. Она используется для представления компонента в процессе проектирования.

Середовище розробки: бібліотека базових класів

Основною перевагою об’єктно-зорієнтованих мов програмування є високий ступінь можливості повторного використання в добре-спроектованих системах.

Вимоги до бібліотеки базових класів

Бібліотека повинна містити універсальні структури даних і алгоритми , здатні задовольнити потреби більшості стандартних додатків C + + . Крім того, бібліотека повинна бути:

∙ Повною. Бібліотека повинна містити сімейство класів , об'єднаних узгодженим зовнішнім інтерфейсом , але з різними уявленнями , так щоб розробники могли вибрати те , семантика чого найбільш точно відповідає додатку .

∙ Мати можливість адаптації. Всі фрагменти коду , що залежать від платформи , повинні бути виділені і ізольовані в окремі класи для забезпечення можливості локальних змін в них. Зокрема , розробник повинен мати контроль над механізмами зберігання даних і синхронізації процесів .

∙ Ефективною. Процедура підключення різних фрагментів бібліотеки до додатку повинна бути простою (ефективність при компіляції ) . Непродуктивні витрати оперативної пам'яті і процесорного часу на обслуговування і підключення повинні бути зведені до мінімуму (ефективність при виконанні). Бібліотека повинна забезпечувати більш надійну роботу , ніж механізми , розроблені користувачем вручну (ефективність при розробці) .

∙ Безпечною. Кожна абстракція повинна бути безпечною з точки зору типів , так щоб статичні припущення про поведінку класу могли бути забезпечені компілятором. Для виявлення порушень динамічної семантики класів повинен бути використаний механізм винятків. Збудження виключення не повинно зіпсувати стан об'єкта, що викликав виняток.

∙ Простою. Бібліотека повинна мати прозору структуру , що дає можливість користувачеві легко знаходити і підключати до додатку її фрагменти .

∙ Розширюваною. Для користувача має бути реалізована можливість включення до бібліотеки нових класів . При цьому архітектурна цілісність середовища розробки не повинна порушуватися.

Бібліотека повинна бути відносно невеликих розмірів; треба завжди пам'ятати , що користувач з більшою охотою займеться розробкою власного коду , ніж вивченням чужого малозрозумілого класу.

Передбачається наявність трансляторів мови C++ , що підтримують параметризовані класи та обробку винятків. З метою забезпечення переносимості бібліотеки вона не повинна залежати від служб операційної системи.

Таким чином , першим результатом нашого аналізу буде поділ усіх абстракцій на дві категорії:

∙ Структури. Містить всі структурні абстракції.

∙ Інструменти. Містить всі алгоритмічні абстракції.

Між цими двома категоріями існує відношення використання: деякі інструменти побудовані на базі більш примітивних властивостей, забезпечуваних структурами.

На другому етапі аналізу варто виділити базові класи, які можуть бути використані в різних стандартних програмах (чим ширше буде коло розглянутих додатків , тим краще). Якщо в результаті виявиться , що деякі з даних класів мають багато спільного з абстракціями , визначеними на першій стадії аналізу , це буде знаком того , що ключові абстракції були виявлені правильно . Можна скласти довгий список специфічних абстракцій, притаманних конкретним видам людської діяльності: валюта, астрономічні координати, одиниці вимірювання маси і довжини. Ми не будемо включати подібні абстракції в нашу бібліотеку, так як вони або занадто погано піддаються формалізації (валюта), або дуже специфічні (астрономічні координати), або настільки примітивні, що немає сенсу організовувати спеціально для них окремі класи (одиниці вимірювання маси і довжини ) .

Провівши аналіз , ми виділимо такі типи структур :

* ∙ Набір. Безліч різних елементів (у тому числі дублікатів ) .
* ∙ Множина. Набір неповторюваних елементів .
* ∙ Колекція. Індексована множина елементів .
* ∙Список. Послідовність елементів, що має початок; структурний поділ допускається.
* ∙Стек. Послідовність елементів; елементи можуть видалятися і додаватися тільки з одного кінця.
* ∙Черга. Послідовність елементів , до якої можна додавати елементи з одного кінця , а видаляти - з іншого .
* ∙Дека. Послідовність елементів , до якої можна додавати і з якої можна видаляти елементи з обох кінців.
* ∙Кільце. Послідовність елементів , до якої можна додавати і з якої можна видаляти елементи, що знаходяться на вершині кругової структури .
* ∙Рядок. Індексована послідовність елементів, в якій можливі операції з підрядками .
* ∙ Асоціативний масив. Словник пар " елемент / значення " .
* ∙ Дерево. Набір (що має початок - корінь дерева) вершин і ребер , які не можуть утворювати цикли і перетинатися ; структурний поділ допускається.
* ∙ Граф. Множина вершин і ребер (без виділеного початкового елемента), яка може містити цикли і перетинатись ; структурний поділ допускається.

Виділяються наступні типи інструментів:

* ∙ Дата / Час. Операції з датою і часом.
* ∙ Фільтри. Введення , обробка і виведення .
* ∙Пошук за зразком. Операції пошуку послідовностей всередині інших послідовностей.
* ∙ Пошук. Операції пошуку елементів усередині структур.
* ∙ Сортування. Операції упорядкування структур.
* ∙ Утиліти. Складові операції , що базуються на базових структурних операціях.

Безсумнівно, існує маса різних функціональних варіантів цих абстракцій. Можна, наприклад, виділити кілька видів сортувань (швидке сортування методом бульбашки, сортування купи і т.д.) або пошуку (послідовний, двійковий, різні способи обходу дерева і т.д.).

Ізольовані абстракції самі по собі - ще не середовище розробки. Як зазначив Вірфс-Брок: "Середовище розробки надає користувачеві модель взаємодій між об'єктами класів, що входять до нього ... Щоб освоїти середовище розробки, перш за все слід вивчити методи взаємодії та відповідальності його класів" . Це і є той критерій , за яким можна відрізнити середовище розробки від простого набору класів: середовище - це сукупність класів і механізмів взаємодії примірників цих класів .

Аналіз показує , що існує певний набір основних механізмів, необхідний для бібліотеки базових класів:

семантика часу і пам'яті;

управління зберіганням даних;

обробка виключень ;

ідіоми ітерації ;

синхронізація при багатопоточності .

При проектуванні системи базових класів необхідно зберігати баланс між перерахованими технічними вимогами [Дійсно, як зазначає Страуструп, " Розробка універсальної бібліотеки значно складніша, ніж розробка окремої програми"] . Якщо ми будемо намагатися вирішити кожну задачу окремо, то, швидше за все, отримаємо ряд ізольованих рішень, не пов'язаних між собою ні загальними протоколами, ні загальною концепцією, ні реалізацією . Такий наївний підхід призведе до великої кількості різних підходів, яка злякає потенційного користувача бібліотеки, яка лиш встигла вийти.

Встанемо на точку зору користувача нашої бібліотеки. Які абстракції представляють наявні в ній класи? Як вони взаємодіють між собою? Як їх можна пристосувати до предметної області ? Які класи грають ключову роль , а які можна взагалі не використовувати? Ось ті питання, на які потрібно дати відповідь перед тим, як пропонувати користувачам бібліотеку для вирішення нетривіальних завдань. На щастя для користувача, йому не обов'язково у всіх деталях уявляти собі, як працює бібліотека, подібно до того, як не потрібно розуміти принципи роботи мікропроцесора для програмування на мові високого рівня. В обох випадках реалізації нижнього рівня може бути продемонстрована кожному користувачеві , але тільки за його бажання .

Розглянемо опис абстракцій нашої бібліотеки з двох точок зору: користувача, який тільки оголошує об'єкти вже існуючих класів, і клієнта, який конструює власні підкласи на базі бібліотечних. При проектуванні з розрахунком на першого користувача бажано якомога сильніше обмежити доступ до реалізацій абстракцій і сконцентруватися на їх відповідальності; проектування з урахуванням запитів другого користувача передбачає відкритість деяких внутрішніх деталей реалізації, однак, не настільки, щоб стало можливим порушити фундаментальну семантику абстракції. Таким чином, доводиться відзначити деяку суперечливість основних вимог до системи.

Однією з головних проблем при роботі з великою бібліотекою є труднощі у розумінні того, які, власне, механізми вона включає в себе. Перераховані вище моделі являють собою як би душу архітектури бібліотеки: чим більше розробник знає про ці механізми, тим легше йому буде використовувати існуючі в бібліотеці компоненти, а не складати з нуля власні. На практиці виходить так, що користувач спочатку знайомиться з утриманням та роботою найбільш простих класів, і тільки потім, перевіривши надійність їх роботи, поступово починає використовувати все більш складні класи. У процесі розробки, у міру того як починають вимальовуватися нові, властиві предметній області користувача, абстракції, вони теж можуть додаватися до бібліотеки. Розвиток об'єктно-орієнтованої бібліотеки - це тривалий процес, що проходить через ряд проміжних етапів.

Саме так ми будемо будувати нашу бібліотеку: спочатку визначимо той архітектурний мінімум , який реалізує всі п'ять виділених нами механізмів , і потім почнемо поступово нарощувати на цьому кістяку все нові і нові функції.