Інститут фізико-технічних та комп’ютерних наук

Відділ комп’ютерних технологій

Кафедра програмного забезпечення комп’ютерних систем

КУРСОВИЙ РЕФЕРАТ

на тему:

**ОБ′ЄКТНО-ЗОРІєНТОВАНЕ ПРОГРАМУВАННЯ**

Студента 2 курсу 243 групи

напряму підготовки

«Програмна інженерія»

Джурана А.С.

Керівник

Доцент кафедри МПУіК,

Лазорик В.В.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

**Члени комісії** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

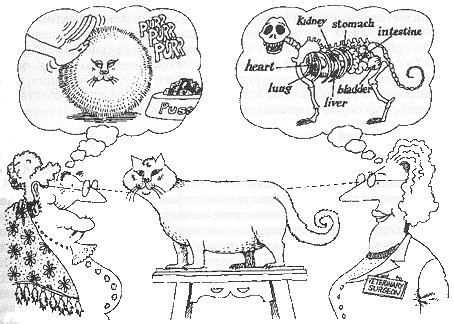
(підпис) (прізвище та ініціали)

Чернівці – 2015

****ОСНОВНІ ПРИНЦИПИ ООП****

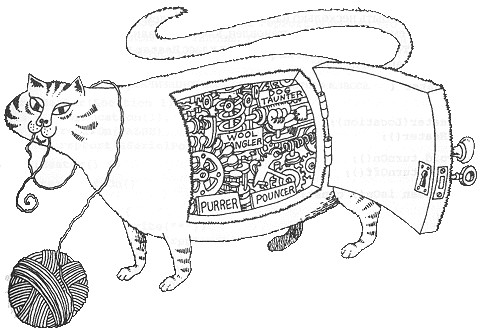
* **абстрагування**
* **інкапсуляція**
* **модульність**
* **ієрархія**
* **Додаткові принципи:**
* **типізація**
* **паралелізм**
* **стійкість**

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.

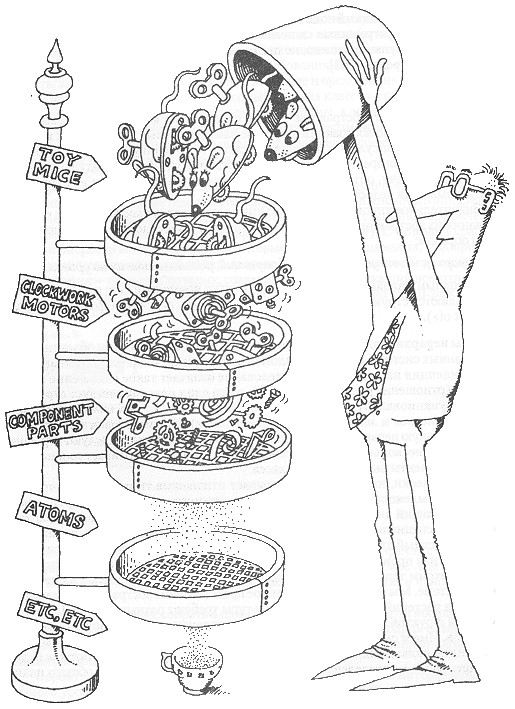
Модульність - це властивість системи, пов'язана з можливістю її декомпозиції на ряд внутрішньо сильно зчеплених, але слабо пов'язаних між собою підсистем (частин).

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



Модульность позволяет хранить абстракции раздельно.

Ієрархія- це впорядкування абстракцій, розташування їх за рівнями.

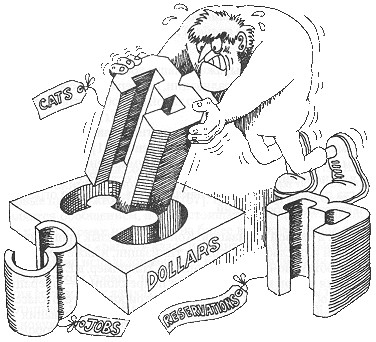


Абстракції утворюють ієрархію.

Типізація - спосіб захиститися від використання об'єктів одного класу замість іншого, або, принаймні, керувати таким використанням**.**

Тип **-** точна характеристика деякої сукупності однорідних об'єктів, що включає структуру і поведінку.

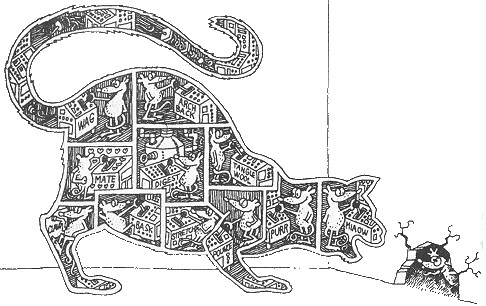
При строгій типізації (наприклад, у мові Оберон) забороняється використання об'єктів невірного типу, потрібно явне перетворення до потрібного типу. В мовах з менш суворою типізацією такого роду заборони ослаблені. Зокрема, допускається поліморфізм - багатозначність імен. Один з проявів поліморфізму, використання об'єкт підтипу (спадкоємця) в ролі об'єкта супертипа (предка).



Строга типізація запобігає змішуванню абстракцій.

Паралелізм **-** це властивість, що відрізняє активні об'єкти від пасивних.

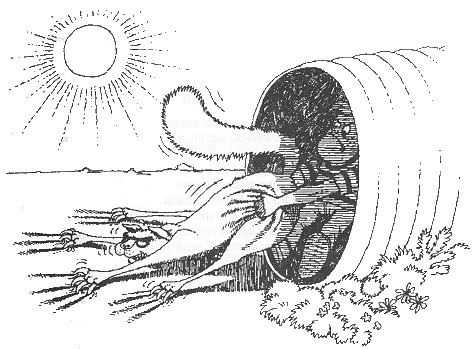
Паралелізм **-** наявність в системі декількох потоків керування одночасно. Об'єкт може бути активний, тобто може породжувати окремий п отік управління. Різні об'єкти можуть бути активні одночасно.



Паралелізм дозволяє різним об'єктам діяти одночасно.

**Збереженість (стійкість) -** здатність об'єкта існувати в часі, переживаючи породивший його процес, і (або) в просторі, переміщаючись зі свого первісного адресного простору**.**

**Стійкість -** здатність об'єкта зберігати своє існування в часі та / або просторі (адресному, зокрема при переміщенні між вузлами обчислювальної системи). Зокрема, стійкість об'єктів може бути забезпечена за рахунок їх зберігання в базі даних.



Збереженість підтримує стан і клас об'єкта в просторі і часі.

****Основні поняттия об′ектно-орієнтованного підходу або Елементи об′ектної моделі****

“ Об'єкт в ОЗП - це сутність, здатна зберігати свій стан (інформацію) і забезпечує набір операцій (поведінку) для перевірки і зміни цього стану.”

**Об'єкт - відчутна сутність (tangible entity) - предмет або явище (процес), що мають чітко виражені межі, індивідуальність і поведінку.**

Будь-який об'єкт володія властивостями, поведінкою й індивідуальністю.

Стан об'єкта визначається значеннями його властивостей (атрибутів) і зв'язками з іншими об'єктами, він може змінюватися з часом.

Поведінкавизначає дії об'єкта та його реакцію на запити від інших об'єктів. Поведінка представляється за допомогою набору повідомлень, які сприймаються об'єктом (операцій, які може виконувати об'єкт).

Індивідуальність - це властивості об'єкта, що відрізняють його від усіх інших об'єктів.

Структура і поведінку схожих об'єктів визначають загальний для них клас.

Об'єкт в JavaScript створюється за допомогою функції Object.create. Ця функція з батьківського класу та опціонального набору властивостей створює нову сутність

Прототип **-** це об'єкт-зразок, за образом і подобою якого створюються інші об'єкти. Об'єкти-копії можуть зберігати зв'язок з батьківським об'єктом, автоматично наслідуючи зміни в прототипі; ця особливість визначається в рамках конкретної мови.

Клас - це безліч об'єктів, пов'язаних спільністю властивостей, поведінки, зв'язків і семантики. Будь-який об'єкт є екземпляром класу. Визначення класів і об'єктів - одна з найскладніших завдань об'єктно-орієнтованого проектування.

Клас (class) **-** це група даних і методів (функцій) для роботи з цими даними. Це шаблон. Об'єкти з однаковими властивостями, тобто з однаковими наборами змінних стану і методів, утворюють клас.

Конструктор класу - спеціальний блок інструкцій, що викликається при створенні об'єкта.

var s = new String ();

Деструктор класу **-** спеціальний метод класу, який слугує для деініцілізації об'єкта (наприклад, звільнення пам'яті).

Атрибут- йменована властивість класу, що визначає інтервал допустимих значень, які можуть приймати екземпляри даної властивості. Атрибути можуть бути приховані від інших класів, це визначає видимість атрибута: рublic (загальний, відкритий); private (закритий, секретний); protected (захищений).

Необхідну поведінку системи реалізується через взаємодію об'єктів. Взаємодія об'єктів забезпечується механізмом пересилання повідомлень. Певний вплив одного об'єкта на інший з метою викликати відповідну реакцію називається операцією або посилкою повідомлення. Повідомлення може бути надіслано тільки протягом з'єднання між об'єктами. У термінах програмування з'єднання між об'єктами існує, якщо один об'єкт має посилання на інший.

**Дескриптор -** це атрибут об'єкта зі зв'язаною поведінкою (англ. Binding behavior), тобто такий, чия поведінка при доступі перевизначається методами протоколу дескриптора.

**Операція -** це послуга, яку можна запросити у будь-якого об'єкта даного класу. Операції реалізують поведінку екземплярів класу. Опис операції включає чотири частини: ім'я; список параметрів; тип значення; видимість.

Реалізація операції називається методом.

**Метод** - це функція або процедура, що належить певному класу або об'єкту.

Розрізняють прості методи і статичні методи (методи класу):

прості методи мають доступ до даних об'єкта (конкретного екземпляра даного класу), статичні методи не мають доступу до даних об'єкта і для їх використання не потрібно створювати екземпляри (даного класу).

Методи предаставляють інтерфейс, за допомогою якого здійснюється доступ до даних об'єкта деякого класу, тим самим, забезпечуючи інкапсуляцію даних.

Залежно від того, який рівень доступу надає той чи інший метод, виділяють:

**відкритий (public) інтерфейс** - загальний інтерфейс для всіх користувачів даного класу;

**захищений (protected) інтерфейс -** внутрішній інтерфейс для всіх спадкоємців даного класу;

**закритий (private) інтерфейс -** інтерфейс, доступний тільки зсередини даного класу.

Такий поділ інтерфейсів дозволяє зберігати незмінним відкритий інтерфейс, але змінювати внутрішню реалізацію.

Поліморфізм - здатність приховувати безліч різних реалізацій під єдиним загальним ім'ям або інтерфейсом.

Поняття поліморфізму може бути інтерпретовано, як здатність об'єкта належати більш ніж одному типу.

Інтерфейс **-** це сукупність операцій, що визначають набір послуг класу або компонента. Інтерфейс не визначає внутрішню структуру, всі його операції відкриті.

Компонент -це відносно незалежна і частина системи, яку можна замінити, що виконує чітко визначену функцію в контексті заданої архітектури.

Компонент являє собою фізичну реалізацію проектної абстракції і може бути: компонентом вихідного коду (cpp-шник); компонентом часу виконання (dll, ActiveX і т. п.); виконуваний компонентом (exe-шник). Компонент забезпечує фізичну реалізацію набору інтерфейсів. Компонентна розробка (component-based development) являє собою створення програмних систем, що складаються з компонентів (не плутати з об'єктно-орієнтованим програмуванням (ООП).

Компонентна розробка **-** технологія, що дозволяє об'єднувати об'єктні компоненти в систему.

Пакет **-** це загальний механізм для організації елементів у групи. Це елемент моделі, який може включати інші елементи. Кожен елемент моделі може входити тільки в один пакет.

Пакет являється:

**-**Засобом організації моделі в процесі розробки, підвищення її керованості і читаності;

-одиницею управління конфігурацією.

Підсистема- це комбінація пакетів (може включати інші елементи моделі) і класу (володіє поведінкою). Підсистема реалізує один або більше інтерфейсів, що визначають її поведінку. Вона використовується для представлення компонента в процесі проектування.

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. Виклик виключної ситуації не повинен зіпсувати стан об'єкта, що викликав дану виключну ситуацію.

∙ 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 більшим бажанням 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, бінарний, 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.

Отже, 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.