**Мова програмування C++ та об’єктно орієнтоване програмування. Обговорення питань, що даються на півбесіді.**



**ООП** це метод управління складністю. Дозволяє організувати складну логікку в простіші структури.

**Зв’язність (Coupling)** це така залежність між модулями: коли вносяться зміни в модуль А, це вимагає внесення змін в модуль В. Програмісти мають прагнути зменшувати зв’язність.

**Клас** це поєднання даних (полів) з методами, які оперують над ними. Програмісти мають проектувати класи, використовуючи приніцип єдиної відповідальності (single-responsibility principle), тобто один клас – одна відповідальність.

**Інкапсуляція** це механізм мови програмування, який агрегує дані в сутності і дозволяє обмежити доступ до компонентів об’єктів. В С++ це реалізовано за допомогою класів і модифікаторів доступу (private, protected, public).

**Наслідування** це механізм мови програмування, що дозволяє створювати нові класи на основі інших класів (базових). Похідний клас може додавати свої власні методи або використовувати методи базового класу. Програмісти використовують це, щоб будувати ієрархічні структури класів.

Базовий клас може бути **абстрактним (abstract class)**, він також називається **інтерфейсом**. Це базовий класс, який не призначений для створювання екземлярів цього класу (а тільки для того, щоб від нього наслідуватись), та містить принаймні одну чисто віртуальну функцію (pure virtual function).

Є різниця між відношенням **наслідування** (inheritance) і **асоціація** (association), коли об’єкт класу В є полем класу А. Наслідування реалізує відношення «Є» (Ламборгіні є машиною), тоді як асоціація реалізує відношення «МАЄ» (машина має двигун).

**Поліморфізм** “Одна специфікація – багато реалізація” – це механізм мови програмування, що дозволяє створювати різні імплементації для однакових специфікацій (інтерфейсів)

Є два різних типу поліморфізму: динамічний і статичний.

Динамічний поліморфізм реалізується за допомогою наслідування і віртуальних функцій. Похідний клас отримує сигнатури методів від базового класу, але ці методи можуть бути перевизначені, тож похідний клас матиме іншу імплементацію. Клієнський код може працювати із вказівником на похідний клас, при цьому буде викликатися перевизначений метод. Це називається пізнім зв’язуванням (late binding)

Статичний поліморфізм реалізується за допомогою шаблонів (templates) та перевантаження функцій (function overloading).

**Шаблони (Templates)** це механізм мови C++ призначений для того щоб реалізовувати спільні алгоритми без специфікації деяких параметрів. В С++ можна створювати шаблонні функції (template functions) та шаблонні класи (template classes).

*Приклад шаблонної функції:*

*template< typename T >*

*void sort( T array[], int size );*

*Виклик:*

*sort<int>( i, 5 );*

*Приклад Шаблонного класу:*

*template< class T >*

*class List*

*{*

*/\* ... \*/*

*public:*

*void Add( const T& Element );*

*bool Find( const T& Element );*

*/\* ... \*/*

*};*

Екземпляри шаблонних функцій створюються на етапі компіляції по одному екземпляру на кожний виклик з конкретним типом.

**Винятки (Exceptions)**

Коли кидається виняток, починається процес **розкрутки стека** (**stack unwinding)**. Це

When an exception is thrown, **the stack unwinding** happens. Це послідовний аномальний вихід з кожної функції у стеку доки не зустрінеться обробник ексепшену відповідного типу. Далі управління переходить до обробника винятка. Винятки використовуються для обробки помилок. Інший підхід до обробки помилок – це повертання статусів з функцій. Винятки дозволяють уникнути написання великої кількості коду перевірки результатів.

**Best practices** для винятків:

* **Не кидати винятків з деструктору** – це може призвести до негайного припинання роботи, якщо інший виняток був у стадії розкрутки в цей момент
* **Кидати по значенню, Ловити по константному посиланню.**

*throw std::runtime\_error(“Test”);* - throw by value

*catch (const std::exception& ex) {*

*std::cout << ex.what() << std::endl;*

*}* .

* **Не викидати винятків за межі модуля**, використовуйте catch(…), щоб запобігти вильоту ексепшенів за межі модуля. Але НЕ використовуйте catch(…) для нормальної обробки винятків, тому що таким чином ви можете впіймати «не свій» виняток і він не попаде туди, де його чекали.
* **Наслідуйте ваші винятки від стандартних класів винятків**: std::exception або std::runtime error.

**Гарантії безпеки винятків (Exception safety guarantees):**

* **Базова гарантія (Base guarantee)** – не станеться витоку ресурсів, якщо виникне виняток.
* **Посилена гарантія (Strong guarantee)** – виняток не вплине на стан застосунку. Посилена гарантія забезпечує транзакційність: якщо кинувся виняток, застосунок має відкотитися на попередній стан або має перейти до наступного стану, але не має застрягнути в деякому неконсистентному стані.
* **Гарантія відсутності винятку (no-throw guarantee)** – Функція не згенерує жодного винятку.

**Управління ресурсами (Resource management)**

Ідеома RAII (resource acquisition is initialization) говорить про те, що ресурс (пам’ять з купи, відкриті дескриптори, тощо) має бути обернутий в інший об’єкт (guard), який буде «володіти» ресурсом, і має його звільнити, коли guard об’єкт буде зруйнований. В найпростішому варіанті це виглядає так:

*class CGuard*

*{*

*private: int\* m\_i;*

*public:*

*CGuard(int\* i) : m\_i(i) {}*

*~CGuard()*

*{*

*delete m\_i;*

*}*

*};*

Guard вивільняє пам’ять під вказівником m\_i автоматично на виклику деструктора CGuard.

**Розумний вказівник (Smart pointer)** це клас (зазвичай шаблонний клас), який реалізує інтерфейс вказівника, а також додає нову можливіть як то вивільнення пам’яті.

Популярні розумні вказівники:

*std::auto\_ptr<>*

*std::unique\_ptr<>*

*std::shared\_ptr<>*

*std::weak\_ptr<>*

*boost::intrusive\_ptr<>*

*boost::scoped\_ptr<>*

*boost::shared\_array<>*

*boost::ptr\_vector<>*

**Тонкі моменти С++**

**Версія стандарту**: Коли Ви читаєте документацію до різних C++ функцій, класів або ключових слів, пам’ятайте яку версію стандарту реалізовує Ваш компілятор. Існують розбіжності у поведінці, або навіть фіча може бути відсутня в деяких версіях стандарту.

**Бітові оператори (Bitwise operation)** – реалізують операції над групами бітів. Здебільшого вони використовуються для перевірки флагів і бітових масок.

unsigned char option\_viewed = 0x01;

unsigned char option\_edited = 0x02;

unsigned char option\_favorited = 0x04;

unsigned char option\_shared = 0x08;

unsigned char option\_deleted = 0x80;

unsigned char myArticleFlags;

myArticleFlags |= option\_viewed; //mark as read

bool deleted = myArticleFlags & option\_deleted; //check if deleted

**Ключові слова**

* mutable – відміняє константність для поля класу.
* volatile – вказання компілятору не оптимізувати змінну.
* auto – використовується замість прописувати точний тип змінної при ініціалізації.
* decltype – decltype(var) – повертає тип змінної var.
* extern – для імпорту змінних і функцій з інших файлів
* inline – рекомендація компілятору оптимізувати функцію
* explicit – запобігає неявному перетворенню типів на ініціалізації
* const\_cast, static\_cast, dynamic\_cast, reinterpret\_cast – перетворення типів
* override – метод перевизначений, final – метод не може бути перевизначений, default – конструктор за замовчанням, delete – забороняє певний конструктор в класі
* nullptr – нульовий вказівник

**Множинне наслідування (Multiple inheritance)**

Множинне наслідування дозволяє класу отримувати функціональність з декількох інших класів. Наприклад класс StudentMusician може бути віднаслідуваний від класу Person, Musician і Worker.

*class StudentMusician : Person, Musician, Worker.*

**Віртуальне наслідування (Virtual inheritance)** – це фукнціональність C++, яка вирішує деякі проблеми множинного наслідування, зокрема проблему ромбовидного наслідування (diamond inheritance). Воно вирішує невизначеність: який саме метод базового класу потрібно викликати.

В такому разі базовий клас вказується з ключовим словом virtual:

class Mammal : public virtual Animal {};

class WingedAnimal : public virtual Animal {};

class Bat : public Mammal, public WingedAnimal

**Порядок ініціалізації (Initialization order)**

Члени класу ініціалізуються в порядку, в якому вони задекларовані, а не в тому порядку, в якому вони знаходяться в списку ініціалізації (initialization list)

Клас конструюється в такому порядку:

* Віртуальні базові класи (тобто ті, що наслідуються з ключовим словом virtual)
* Інші базові класи
* Поля – члени класу
* Виконується власний конструктор класу

Деструктори виконуються в зворотньому порядку.

**Патерни проектування (Design patterns)**

Патерн проектування – це повторювана архітектурна конструкція, яка розв’язує деяку популярну задачу програмування

Класифікація патернів проектування:

**Породжувальні (Creational)**описують створення об’єкту або групи пов’язаних об’єктів: Абстрактна фабрика, Фабричний метод, тощо

**Структурні (Structural)** описують компонування об’єктів, тобто як сутності використовують одна одну: Адаптер, Міст, Декоратор, тощо.

**Поведінкові (Behavioral)** пов’язані із наданням відповідальності об’єктам, відрізняються від структурних бо описують не тільки структуру, але й описують шляхи передачі даних та забезпечення взаємоді: Спостерігач, Визитер, Команда, тощо.