# Об’єктно-орієнтованого програмування: основні поняття.

# Об’єктно-орієнтованого програмування: основні принципи (властивості).

# Концепція об’єктно-орієнтованого підходу: складові об’єктно-орієнтованої технології, їх загальна характеристика.

# Основи об’єктно-орієнтованих технологій: об’єктна модель, загальна характеристика.

# Основи об’єктно-орієнтованих технологій: об’єктна модель, складові об’єктного підходу.

# Оголошення класу: формат, специфікатори доступу, їх загальна характеристика.

Оголошення класу являє собою узагальнення комбінованого типу даних і складається з двох частин: заголовка, що включає ключове слово class, і тіла, укладеного у фігурні дужки:

class ім'я\_класу

{ тіло\_класу;

};

Однак, для підтримки принципа інкапсуляції, доступ до членів класу (атрибутів і методів) з об'єктів інших класів може бути обмеженим. Це забезпечуються введенням в клас областей доступу:

- public (відкритий/загальний доступ) – доступ з поза класу не обмежений;

- protected (захищений доступ) – доступ можливий тільки власним методам і методам похідних класів;

- private (закритий/приватний доступ) – доступ можливий тільки власним методам.

Специфікатори доступу – це спеціальні синтаксичні конструкції, що явно задають область видимості кожного члена класу.

# Реалізація класу: зовнішні та вбудовані методи, особливості їх задання.

. Реалізація класу відображає внутрішній його устрій і складається з визначення методів, оголошених в інтерфейсі класу. Кожен метод класу повинен бути визначений (описаний) в програмі. Визначити метод можна або безпосередньо усередині класу (в тілі класу), або поза його межами (після тіла класу).

***Особливості задання зовнішніх методів класу*:**

Оскільки різні класи можуть мати методи з однаковими іменами, при визначенні зовнішнього методу необхідно вказати до якого класу він належить. Для цього треба кваліфікувати ім'я метода іменем відповідного класу за допомогою операції розширення області видимості "::":

тип\_результату ім'я\_класу :: ім'я\_методу ([параметри])

{ тіло };

Оператор "::" «прив'язує» ім'я члена до імені класу, тим самим однозначно ідентифікуючи методи даного класу. Це дозволяє компілятору самостійно визначити до якого класу належить та чи інша функція.

Наприклад, ось як можна записати код зовнішнього методу Set\_Сolor():

void TPoint :: Set\_Сolor(int color\_new)

{ color = color\_new;

}

***Особливості задання вбудованих методів класу*:**

Будь-яка функція, яка визначається в оголошенні класу, автоматично стає вбудо-ваною. У цьому випадку необов'язково специфікувати її оголошення ключовим сло-вом inline.

class TPoint

{ int x, y;

int color;

public :

int GetX() { cout<<" X= "<< x;} // автоматично вбудована функціїя

int GetY() { cout<<" Y= "<< y;}

int Get\_Сolor() { cout<<" Сolor - "<< color;}

void Set\_XY(int хх, int yy) { х = хх; y = yy; }

void Set\_Сolor(int c) { color = c; }

};

# Організація доступу до членів об'єкта: способи створення об'єктів, особливості реалізації доступу до їх членів.

Створення об'єкта в програмі здійснюється оголошенням відповідної змінної (для статичних об'єктів) або динамічним розміщенням змінної в пам'яті (для динамічних об'єктів). Способи створення таких об'єктів залежать від наявності або відсутності в класі спеціального методу, що визначає необхідні для створення об'єкта дії, – так званого конструктора.

1. Якщо в класі відсутній конструктор, але описані захищені (protected) або прихо-вані (private) атрибути, то можливе створення тільки неініціалізованих об'єктів.

Формат створення об'єктів:

ім'я\_класу ім'я\_об'єкта;

ім'я\_класу \* покажчик\_на\_об'єкт = new ім'я\_класу;

1. За відсутності в класі конструктора і захищених (protected) або прихованих (pri-vate) полів для оголошення ініціалізованих об'єктів використовують оператор ініці-алізації, застосовуваний при створенні ініціалізованих структур, наприклад,

TPoint рoint = {3, 4, 2};

Ініціалізуючі значення при цьому повинні перераховуватися в порядку задання полів в описі класу.

1. **За допомогою конструктора**

Реалізація доступу до членів класу визначається областю їх видимості відносно коду (методу), з якого здійснюється звернення.

Звертання до відкритих (public) членів класу (як атрибутів, так і методів) ззовні об'єкта даного класу (з методів інших класів або з функцій, не пов'язаних ні з яким класом), здійснюється за допомогою повних імен, кожне з яких має вигляд:

ім'я\_об'єкта . ім'я\_класу :: ім'я\_члена\_класу;

Якщо атрибут оголошений як private або protected , то доступ до нього із методів того ж самого класу здійснюється аналогічно, як і до відкритих (public) атрибутів цього класу (за іменем, без явного вказання імені об'єкта).

Доступ же до таких атрибутів з методів інших класів або з функцій, не пов'язаних ні з яким класом, здійснюється тільки через методи цього ж класу.

# Структури як різновиди класів: особливості визначення, порівняльна характеристика.

Синтаксис класу в С++ збігається з синтаксисом структури С++:

struct ім'я\_структури

{ визначення\_членів\_структури;

}

По суті, класи і структури є спорідненими типами. За одним винятком, вони взає-мозамінні, оскільки структура також може містити дані та програмні коди, які мані-пулюють цими даними так само, як це може робити і клас. Єдина відмінність між С++-структурою і С++-класом полягає у тому, що за замовчуванням члени класу є закритими (private), а члени структури – відкритими (public).

# Об'єднання як різновиди класів: особливості визначення, порівняльна характеристика.

Згідно з визначенням мови програмування C++, об'єднання – це, по суті, аналогічний клас, у якому всі члени даних зберігаються в одній і тій самій області. Об'єднання може містити конструктор і деструктор, а також функції-члени. Звичайно ж, члени об'єднання за замовчуванням відкриті (public), а не закриті (private).

# Конструктори: поняття конструктора, класифікація конструкторів.

Конструктор класу – це спеціальний відкритий (public) метод, що автоматично викликається при створенні об'єкта класу і виконує дії по виділенню пам'яті під об'єкт та, за необхідності, ініціалізацію його атрибутів.

Загалом конструктори можна класифікувати наступними чином:

1) за призначенням

- ініціалізуючий конструктор,

- неініціалізуючий конструктор,

2) за наявністю параметрів

- з параметрами,

- без параметрів,

3) за кількістю та типом параметрів

- конструктор за замовчуванням,

- конструктор з двома і більше параметрами,

- конструктор копіювання,

- конструктор перетворення.

# Конструктори: поняття конструктора, конструктор за замовчуванням, особливості реалізації і використання.

Конструктор класу – це спеціальний відкритий (public) метод, що автоматично викликається при створенні об'єкта класу і виконує дії по виділенню пам'яті під об'єкт та, за необхідності, ініціалізацію його атрибутів.

Конструктор, який не має параметрів, називають конструктором за замовчуванням. Такий конструктор, зазвичай, ініціалізує атрибути класу константними значеннями.

Формат визначення конструктора за замовчуванням:

ім'я\_класу()

{ тіло\_конструктора

}

Якщо в класі явно не описано жодного конструктора, то компілятор автоматично генерує «порожній» конструктор, який теж грає роль конструктора за замовчуванням.

Клас може містити тільки один конструктор, що дозволяє створити об'єкт без вказання аргументів. Якщо б таких конструкторів було декілька, то компілятор не зміг би вирішити, який з них повинен бути викликаний.

Особливо важлива наявність такого конструктора при створенні масиву об'єктів.

# Конструктори: поняття конструктора, конструктор з двома і більше параметрами, особливості реалізації і використання.

Конструктор класу – це спеціальний відкритий (public) метод, що автоматично викликається при створенні об'єкта класу і виконує дії по виділенню пам'яті під об'єкт та, за необхідності, ініціалізацію його атрибутів.

Конструктор, який разом із створенням об'єкту присвоює його атрибутам деякі початкові значення, отримуючи їх як параметри, називають конструктором з параметрами.

Формат визначення конструктора з двома і більше параметрами:

ім'я\_класу(тип параметр\_1, …, тип параметр\_n)

{ тіло\_конструктора

}

# Конструктори: поняття конструктора, конструктор копіювання, особливості реалізації і використання.

Конструктор класу – це спеціальний відкритий (public) метод, що автоматично викликається при створенні об'єкта класу і виконує дії по виділенню пам'яті під об'єкт та, за необхідності, ініціалізацію його атрибутів.

При створенні об'єкта його атрибути можуть бути проініціалізовані значеннями атрибутів іншого об'єкта цього ж типу. Для такого створення об'єкту у С++ викорис-товується так званий конструктор копіювання.

Формат визначення конструктора копіювання:

ім'я\_класу\_1([const] ім'я\_класу\_1 &ім'я\_об'єкта);

Якщо у класі немає явно описаного конструктора копіювання, то компілятор генерує його автоматично. Такий конструктор виконує *поверхневе копіювання*, тобто поелементне копіювання атрибутів класу.

# Конструктори: поняття конструктора, конструктор перетворення, особливості реалізації і використання.

Конструктор класу – це спеціальний відкритий (public) метод, що автоматично викликається при створенні об'єкта класу і виконує дії по виділенню пам'яті під об'єкт та, за необхідності, ініціалізацію його атрибутів.

При створенні об'єкта його можна проініціалізувати об'єктом іншого типу, тобто перетворити об’єкт, тип якого відрізняється від типу конструктора, в об’єкт даного класу. Для такого створення об'єкту використовується конструктор перетворення.

Можливі формати заголовка конструктора перетворення:

ім'я\_класу(ім'я\_класу ім'я\_об'єкта);

ім'я\_класу(ім'я\_класу &ім'я\_об'єкта);

ім'я\_класу(const ім'я\_класу &ім'я\_об'єкта);

Конструктор перетворення може викликатися автоматично. Таке перетворення типу неявно застосовується тільки якщо воно унікальне.

# Конструктори: поняття конструктора, список ініціалізаціі, особливості його задання і використання.

Конструктор класу – це спеціальний відкритий (public) метод, що автоматично викликається при створенні об'єкта класу і виконує дії по виділенню пам'яті під об'єкт та, за необхідності, ініціалізацію його атрибутів.

Для таких випадків застосовується спеціальна конструкція, що включається в опис конструктора, - список ініціалізації, який дозволяє ініціалізувати змінні без ви-користання операцій присвоєння значень. Формат конструктора зі списком ініціалі-зації:

ім'я\_класу(параметри) : список\_ініціалізаціі

{ тіло\_конструктора

}

Конструктор з параметрами, що містить список ініціалізації часто називаюють конструктором ініціалізації.

Основне призначення даного конструктора – ініціалізація атрибутів особливих типів, наприклад, константних і посилальних.

# Конструктори: поняття конструктора, перевантаження конструкторів.

Конструктор класу – це спеціальний відкритий (public) метод, що автоматично викликається при створенні об'єкта класу і виконує дії по виділенню пам'яті під об'єкт та, за необхідності, ініціалізацію його атрибутів.

У класі може бути визначено декілька конструкторів з одним і тим самим іменем. Тоді конструктор вважається перевантаженим. Єдина вимога: не може бути двох конструкторів з однаковим (за типами) набором формальних параметрів, зокрема, не може бути двох конструкторів за умовчанням.

# Конструктори: особливості використання.

конструктор имеет тоже имя, что и его класс;

– конструктор не содержит оператора return и не возвращает никакого значения (даже void);

– конструктор может определяться явно пользователем или неявно компилятором. Неявно происходит объявление конструктора в таких случаях: определение нового объекта класса, при копировании объекта, при динамическом создании объекта с помощью оператора new

* определение конструктора может производиться внутри и вне класса;

– конструктор может иметь, а может не иметь параметры. Конструктор с параметрами инициализирует объект при объявлении. Конструктор без параметров инициализирует пустой объект;

– конструкторы не могут быть вложенными;

– конструктор не может быть константным, статическим или виртуальным (const, static или virtual);

– ошибки, генерируемые конструктором, обрабатываются только механизмом обработки исключительных ситуаций;

– конструктор не наследуется.

# Деструктори: поняття деструктора, особливості реалізації.

Деструктор – спеціальний метод, який автоматично викликається для коректного знищення об'єктів.

Ім'я деструктора, як і ім'я конструктора, співпадає з іменем классу, але йому пере-дує ще додатковий символ "~" («тильда»):

~ім'я\_класу() { … }.

У деструкторів немає параметрів. Подібно до конструкторів, деструктори не по-вертають значень, а отже, в їх оголошеннях відсутній тип значення, що поверта-ється.

Клас може мати тільки один деструктор або не мати жодного.

# Динамічні об'єкти: класи з динамічними атрибутами, особливості створення, ініціалізація і знищення динамічних об'єктів.

При створенні класів з динамічними полями необхідно враховувати велику кількість особливостей:

* Наявність свого конструктора
* Деструктора
* Конструктора копіювання
* Перевантаженого оператора «=»

При виділенні динамічної пам'яті для окремих об'єктів використовують такі форми звернення до операції new:

ім'я\_покажчика\_на\_об'єкт = new ім'я\_класу;

ім'я\_покажчика\_на\_об'єкт = new ім'я\_класу (список\_параметрів);

Другу форму застосовують за наявності у конструктора списку параметрів.

Операція delete вимагає вказання тільки імені об'єкта:

delete ім'я\_покажчика\_на\_об'єкт;

# Особливості роботи з об'єктами: присвоєння об'єктів, способи реалізації.

Об'єкти одного і того ж класу можна присвоювати один одному. При виконанні операції присвоювання за замовчуванням здійснюється поверхневе копіювання, тобто дані першого об'єкта порозрядно копіюються у другий. Тому якщо об’єкт містить динамічні атрибути, то необхідно перевантажити конструктор копіювання та оператор «=».

# Особливості роботи з об'єктами: способи передачі об'єктів функціям, їх загальна характеристика.

Об'єкти можна передавати у функції так само, як і змінні будь-яких інших типів даних. За замовчуванням об'єкти класу передаються функціям за значенням. Тобто, у функцію передається не сам об'єкт, а його копія, яка стає параметром функції. Створення копії означає появу нового тимчасового об'єкта. Усі зміни, внесені в об'єкт-копію в процесі виконання функції, не впливають на вхідний об'єкт, який використовується як аргумент для функції.

Якщо в класі безпосередньо не визначено конструктор копіювання, то він генерується за замовчуванням. Конструктор копіювання за замовчуванням створює побітову (тобто однакову) копію об'єкта.

Коли виконання функції, яка містить аргумент-об'єкт завершується, копія аргумента руйнується, для чого викликається деструктор цього локального об'єкта. Потреба виклику даного деструктора пов'язана з виходом локального тимчасового об'єкта з області видимості функції, у якій він використовувався.

Також можна передавати об’єкти у функції за посиланнями та покажчиками. Тоді їх побітові копії не створюються.

# Особливості роботи з об'єктами: способи повернення об'єктів функціями, їх загальна характеристика.

Щодо механізму повернення об'єктів функціями, то тут виникає ситуація, анало-гічна ситуації з передачею об'єктів у функції. Якщо тип значення, що повертається функцією, є об’єктом класу, то під час виклику функції компілятор автоматично ге-нерує тимчасовий об'єкт цього класу і використовує значення, яке визначено в опе-раторі return, для ініціалізації цього об’єкта. Після повернення значення із функції негайно викликається деструктор тимчасового об'єкта і цей об'єкт руйнується.

Руйнування тимчасового об'єкта в деяких ситуаціях може викликати непередба-чувані побічні ефекти. Так, якщо повернутий функцією об'єкт має деструктор, який звільняє динамічно виділену область пам'яті, то ця пам'ять буде звільнена навіть у тому випадку, якщо об'єкт, який отримує повернуте функцією значення, все ще ви-користовує цю пам'ять.

Одним із варіантів вирішення цієї проблеми є використання для повернення покажчика або посилання на об'єкт. Але здійснити це не завжди вдається.

Ще один спосіб вирішення цієї проблеми полягає у використанні конструктора копіювання.

# Особливості роботи з об'єктами: неявний параметр *this*.

# Особливості роботи з об'єктами: константні члени класу і константні об'єкти, особливості використання.

Коли метод, що належить класу, викликається для обробки даних конкретного об'є-кта, йому, крім явно оголошених параметрів, автоматично і неявно передається ще один прихований параметр: покажчик на той об'єкт, для якого викликається цей ме-тод. У С++ цей покажчик має спеціальне ім'я this і неявно визначений в кожному методі класу наступним чином:

ім'я\_класу \*const this = адреса оброблюваного об'єкта.

# Особливості роботи з об'єктами: статичні члени класу, особливості використання.

Члени класу (як методи, так і атрибути) можуть бути загальними (спільними) для усіх об'єктів даного класу. Тоді їх називають статичними.

Статичні атрибути використовуються для збереження даних, спільних для всіх ек-земплярів даного класу (значення яких є однаковими для всіх об'єктів даного класу).

Створення статичних атрибутів виконується в три етапи:

- оголошення (декларація);

- визначення (виділення пам’яті);

- ініціалізація.

Особливістю визначення статичного атрибута є те, що повторювати ключове слово static при цьому не можна. Воно повинно бути присутнім тільки в тілі класу.

Ініціалізацію статичних атрибутів можна поєднати з їх визначенням. Формат такого визначення:

тип\_змінної ім'я\_класу :: ідентифікатор [= ініціалізатор];

Cтатичний атрибут може належати до типу того ж класу, членом якого він є.

Також статичний атрибут може виступати в ролі аргументу за замовчуванням для метода класу, а для нестатичних це заборонено.

Статичні методи – це методи, які є спільними для усіх об'єктів даного класу.

Статичні методи класу вирізняються тим, що мають доступ тільки до статичних членів (атрибутів і методів) даного класу.

# Особливості роботи з об'єктами: статичні атрибути, особливості створення і використання.

Члени класу (як методи, так і атрибути) можуть бути загальними (спільними) для усіх об'єктів даного класу. Тоді їх називають статичними.

Статичні атрибути використовуються для збереження даних, спільних для всіх ек-земплярів даного класу (значення яких є однаковими для всіх об'єктів даного класу).

Створення статичних атрибутів виконується в три етапи:

- оголошення (декларація);

- визначення (виділення пам’яті);

- ініціалізація.

Особливістю визначення статичного атрибута є те, що повторювати ключове слово static при цьому не можна. Воно повинно бути присутнім тільки в тілі класу.

Ініціалізацію статичних атрибутів можна поєднати з їх визначенням. Формат такого визначення:

тип\_змінної ім'я\_класу :: ідентифікатор [= ініціалізатор];

Cтатичний атрибут може належати до типу того ж класу, членом якого він є.

Також статичний атрибут може виступати в ролі аргументу за замовчуванням для метода класу, а для нестатичних це заборонено.

# Особливості роботи з об'єктами: статичні методи, особливості створення і використання.

Статичні методи – це методи, які є спільними для усіх об'єктів даного класу.

Статичні методи класу вирізняються тим, що мають доступ тільки до статичних членів (атрибутів і методів) даного класу.

# Дружні функції і дружні класи, особливості реалізації.

Дружньою функцією класу називають функцію, яка, не будучи членом класу, має доступ до усіх його членів, у тому числі і захищених.

Для того щоб оголосити зовнішню функцію дружньою даному класу, то необхідно помістити її прототип з ключовим словом friend в тіло класу.

Працюючи з дружніми функціями, потрібно враховувати наступні особливості:

- для дружньої функції несуттєві специфікації доступу;

- дружня функція при виклику не отримує покажчика this;

- об'єкти класів повинні передаватися дружній функції тільки явно через апарат параметрів;

- дружні функції не можуть мати специфікатори static і extern;

- у класі, функція-член якого є дружньою до не цілком оголошеного класу, слід розміщати тільки прототип; реалізація функції повинна знаходитися після пов-ного оголошення класу;

- при виклику дружньої функції не можна використовувати операції вибору:

імя\_об'екта.імя\_функціі

указатель\_на\_об'ект-> імя\_функціі

- дружні функції не взаємні;

- одна і та ж функція може бути "другом" декількох класів;

- дружні функції не успадковуються.

Коли усі функції деякого класу є дружніми стосовно іншого класу, можна оголосити весь клас ***дружнім***.

# Масиви об'єктів: особливості створення і ініціалізації.

Якщо при створенні масиву використовувався неініціалізуючий конструктор, то всі об'єкти - елементи масиву будуть неініціалізованими. Застосування ініціалізуючого конструктора без параметрів або конструктора з параметрами, заданими за умовчанням, може:

- ініціалізувати всі створювані об'єкти однаково, як задано в ініціалізуючому конструкторі (рідко потрібно за умовою задачі);

- ініціалізувати всі створювані об'єкти випадковим чином (ще рідше потрібно за умовою задачі);

- ініціалізувати всі створювані об'єкти значеннями, що вводяться в процесі ініціалізації (введення в методі ініціалізації вважається нетехнологічним).

Приклад:

myClass array[4][2] = { myClass(1, 2), myClass(3, 4),

myClass(5, 6), myClass(7, 8),

myClass(9, 10), myClass(11, 12),

myClass(13, 14), myClass(15, 16) };

# Динамічні масиви об'єктів: способи створення і знищення, їх загальна характеристика.

Масиви можуть формуватися і з динамічних об'єктів. Це можна зробити трьома способами:

- створити динамічний масив об'єктів - пам'ять під нього виділяють одним безпе-рервним фрагментом рівним обсягу всіх об'єктів масиву, наприклад,

B mas[] = new B[n];

- створити статичний масив покажчиків на об'єкти і потім динамічно виділити пам'ять під елементи, наприклад,

B \*mas[n]; // пам'ять під масив покажчиків виділена статично

for (i=0; i<n; i++) mas[i] = new B; // виділення пам'яті під об'єкти

- створити динамічний масив покажчиків і потім також динамічно виділити па-м'ять під елементи наприклад,

B \*\*mas = new B \*[n]; // пам'ять під масив покажчиків виділена динамічно

for (i=0; i<n; i++) mas[i] = new B; // виділення пам'яті під об'єкти

Звільняти виділену пам'ять потрібно так, як вона була виділена:

- одним фрагментом - для динамічного масиву об'єктів:

delete [] mas;

- поелементно - для масиву покажчиків на об'єкти:

for (i = 0; i <n; i ++) delete mas [i];

- одним фрагментом, якщо пам'ять під масив покажчиків виділялася динамічно:

delete [] mas;

# Реалізація успадкування в классах: різновиди наслідування, їх загальна характеристика.

# Просте успадкування, його різновиди.

# Різновиди простого успадкування: відкрите успадкування, особливості реалізації.

# Різновиди простого успадкування: закрите успадкування, особливості реалізації.

# Різновиди простого успадкування: захищене успадкування, особливості реалізації.

# Механізми використання захищених членів класу, особливості реалізації.

# Просте успадкування: повернення успадкованим членам класу початкової специфікації доступу, основні способи, особливості їх реалізації.

# Просте успадкування: використання конструкторів і деструкторів.

# Механізми множинного успадкування, особливості реалізації, порядок виклику конструкторів і деструкторів.

# Віртуальне наслідування, особливості реалізації.

# Поліморфізм: поняття, різновиди, їх загальна характеристика.

# Статичний поліморфізм, особливості реалізації.

# Динамічний поліморфізм, особливості реалізації.

# Механізм віртуальних функцій, особливості реалізації.

# Суто віртуальні функції та абстрактні класи, особливості реалізації.

# Шаблонні функції: поняття шаблонної функції, виведення її аргументів, спеціалізація та конкретизація шаблонної функції.

# Шаблонні функції: перевантаження шаблонної функції, використання стандартних параметрів.

# Шаблонні класи: поняття, особливості задання і використання.

# Спеціалізація шаблонних класів: різновиди, особливості реалізації.

# Виняткові ситуації: поняття, загальний механізм обробки.

# Механізм обробки виняткових ситуацій: засоби реалізації виключень, їх загальна характеристика.

# Особливості обробки виняткових ситуацій: тотальне перехоплення, список дозволених виняткових ситуацій, непередбачені виняткові ситуації.

# Стандартні виняткові ситуації: категорії стандартних класів помилок, їх загальна характеристика.