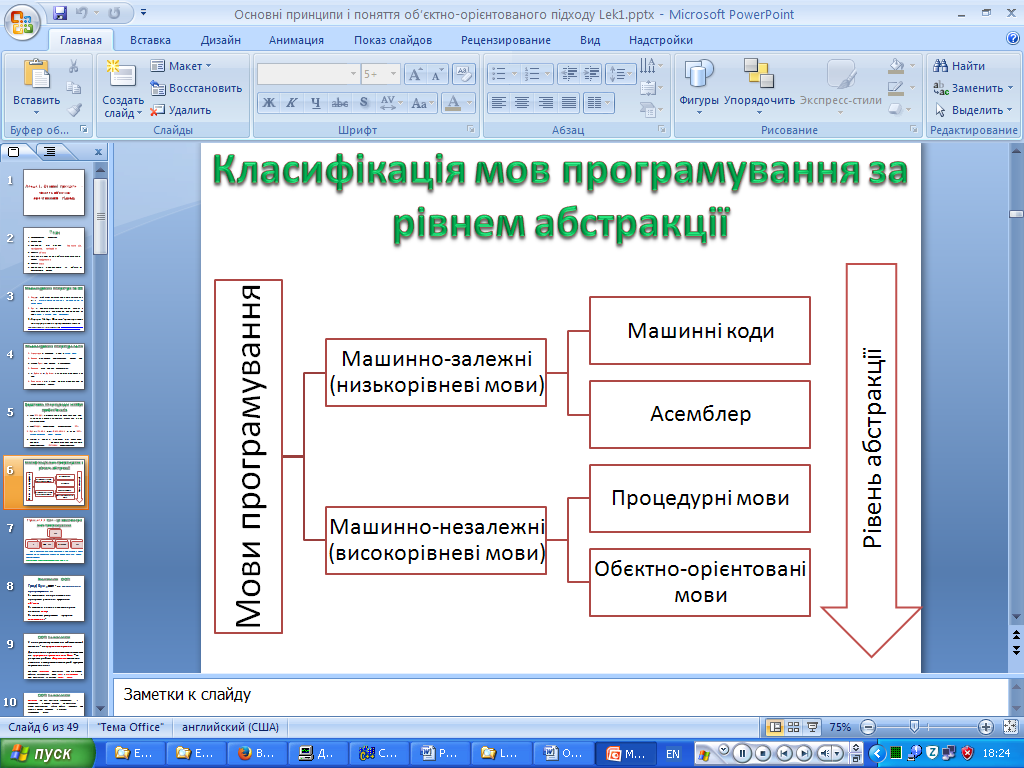
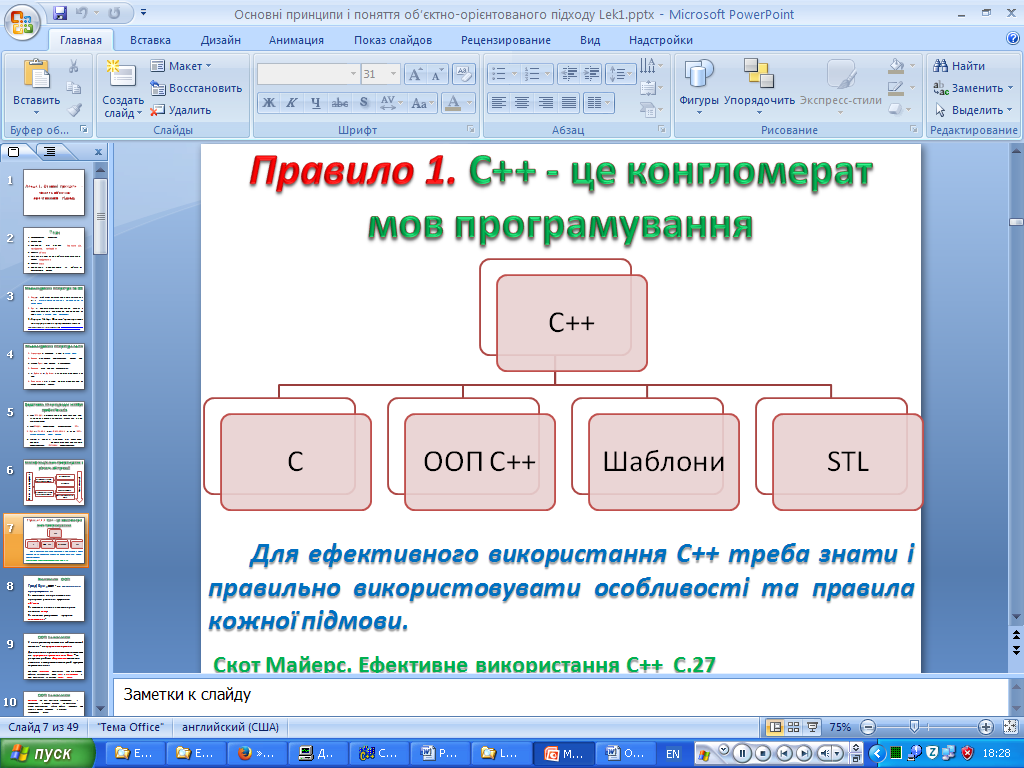
**Лекція 12. Об’єктно-орієнтоване програмування. Основні принципи**

**С++ - це конгломерат мов програмування.**

***Для ефективного використання С++ треба знати і правильно використовувати особливості та правила кожної підмови.***



**Визначення ООП. Граді Буч: „ООП - це методологія програмування ...   
1. заснована на представленні програми у вигляді сукупності об'єктів,   
2. …кожен з яких є екземпляром певного класу,   
3. …а класи утворюють ієрархію спадкування.”**

У попередньому визначенні найважливіший момент - це **ієрархія наслідування**.

Два основних принципи людського мислення - це **групування** і **узагальнення**. **Класи** – це результат роботи **абстрактного** мислення людини з використанням операцій групування та узагальнення. При цьому **групування** досягається тим, що схожим об'єктам зіставляється один **клас**, а **узагальнення** в ООП досягається за рахунок **ієрархії класів**.

**Абстракція** (від лат. Abstractio - відволікання) - відволікання в процесі пізнання від несуттєвих сторін, властивостей, зв'язків об'єкта (предмета або явища) з метою виділення їх істотних, закономірних ознак. Абстракції виконують захисну функцію і допомагають нам не зійти з розуму від надлишку інформації

**Абстрактне мислення** - дозволяє провести **декомпозицію** предметної області на набір понять і зв'язків між ними.

**Витоки ООП – програмування у великому**

* Прості програми – це програми, що які замислюються, розробляються, супроводжуються і використовуються однією людиною – початківцем або професіоналом, що працює ізольовано.
* Складні програми - це промислові програмні продукти. Їм притаманна складність - єдиний розробник практично не в змозі охопити всі аспекти такої системи. Складність таких програм перевищує можливості людського інтелекту.

**Причини складності**

Складність викликається чотирма основними причинами:

* складністю реальної предметної області, з якої виходить замовлення на розробку;
* незадовільними способами опису поведінки великих дискретних систем;
* трудністю управління процесом розробки;
* гнучкістю програмних систем.

**Декомпозиція – засіб боротьби зі складністю.**

**Поняття об'єкта**

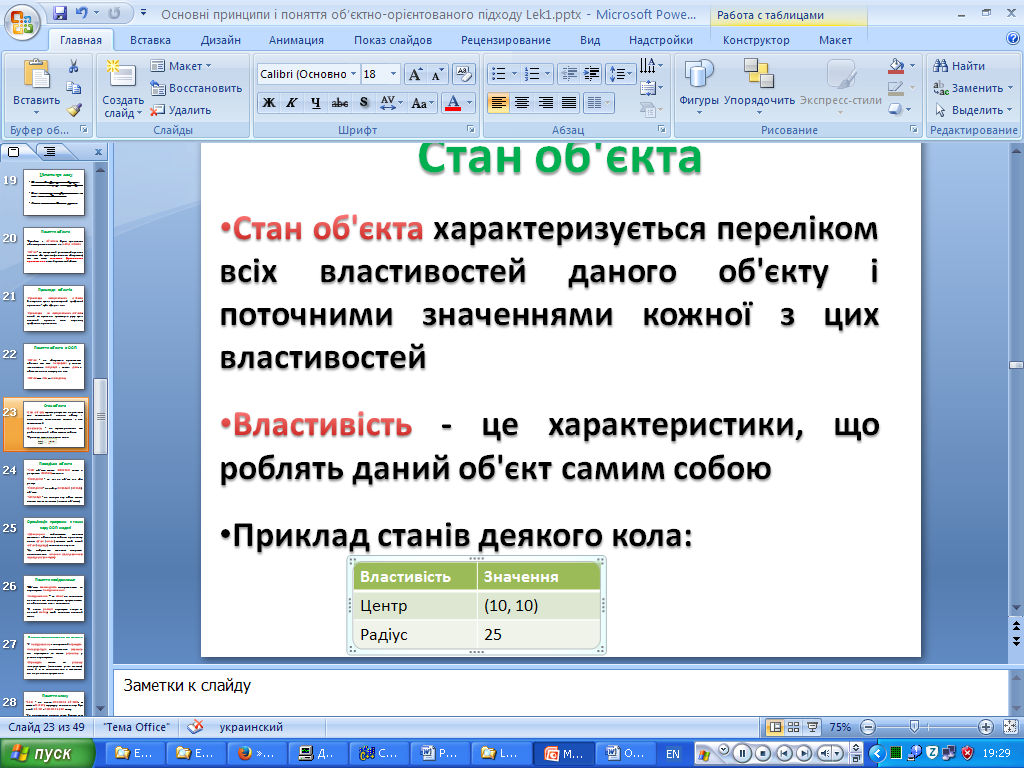
* Будь-що є об'єктом. Будь предметна область розглядається як набір об'єктів
* Об'єкт - це конкретний упізнаваний предмет, одиниця або сутність (реальна чи абстрактна), що має чітко визначене функціональне призначення в даній предметній області

Приклади матеріальних об'єктів: більярдна куля, тривимірний графічний примітив - куб, сфера і т.п.

Приклади не матеріальних об'єктів, подій чи процесів: траєкторія руху кулі, хімічний процес, лінія перетину графічних примітивів

**Поняття об'єкта в ООП**

* **Об'єкт - це абстракція предметної області, що має інтерфейс у вигляді іменованих операцій і власні дані, з обмеженням доступу до них**
* **Об'єкт має стан та поведінку.**

**Стан об'єкта характеризується переліком всіх властивостей даного об'єкту і поточними значеннями кожної з цих властивостей.**

**Властивість - це характеристики, що роблять даний об'єкт самим собою. Приклад станів деякого кола:**

**Стан об'єкта може змінитися тільки в результаті виклику методів.**

* **Поведінка - це те, як об'єкт діє або реагує**
* **Поведінка - це набір операцій (методів) об'єкта**

**Операція - це послуга, яку об'єкт може надати своїм клієнтам (іншим об'єктам)**

**Організація програми з точки зору ООП моделі**

**Обчислення здійснюється шляхом взаємодії об'єктів між собою, при якому один об'єкт (клієнт) вимагає щоб інший об'єкт (сервер) виконав деяку дію**

**Це відбувається шляхом відправки повідомлення клієнтом (відправником) серверу (отримувачу) .**

**Поняття повідомлення**

**Об'єкти взаємодіють відправляючи та отримуючи повідомлення.**

**Повідомлення – це запит на виконання деякої дії, що доповнюється аргументами необхідними для її виконання.**

**В якості реакції отримувач запускає деякий метод, щоб виконати наданий запит.**

**Відмінність повідомлень від методів**

**У повідомлень є конкретний отримувач**

**Інтерпретація повідомлення залежить від отримувача та може різнитися у різних отримувачів**

**Отримувач може по різному інтерпретувати (викликати різні методи) одне й те ж повідомлення в залежності від переданих аргументів**

**Поняття класу**

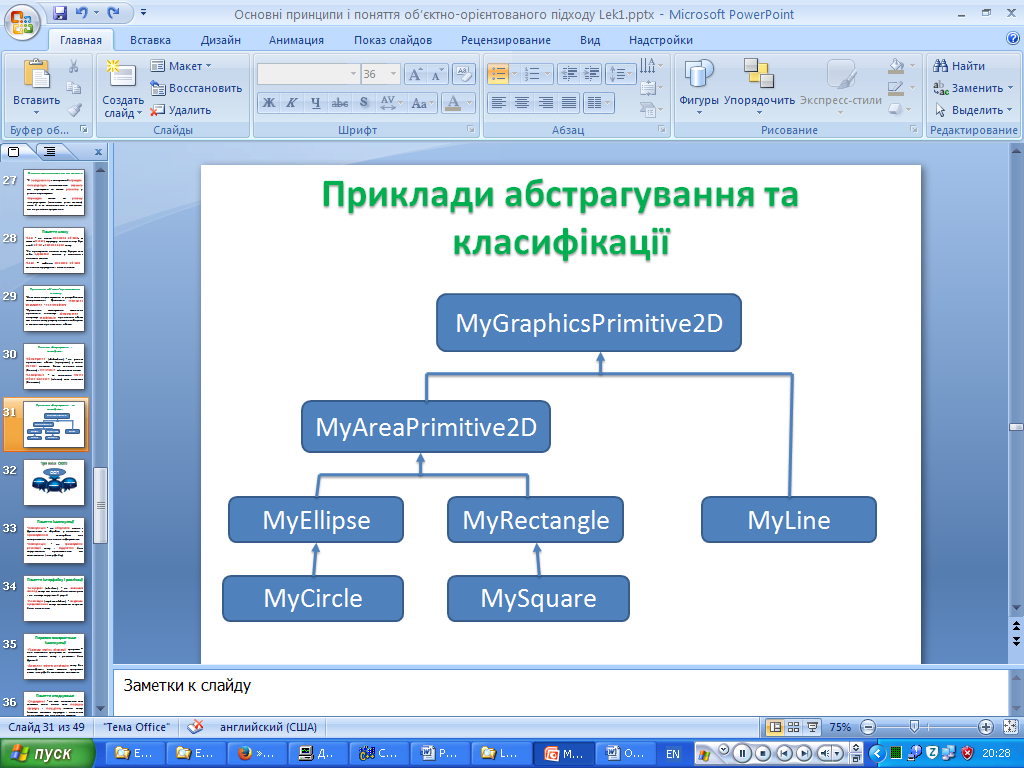
**Клас - це деяка множина об'єктів, що мають спільну структуру та поведінку. Будь-який об'єкт є екземпляром класу.**

**Усі примірники одного класу будуть вести себе однаковим чином у відповідь на однакові запити.**

**Клас – шаблон множини об'єктів із спільною структурою і поведінкою.**

**Принципи об'єктно-орієнтованого підходу**

**Клас винен проектуватися и розроблятися з використання. Принципів інкапсуляції, успадкування і поліморфізму.**

**Правильне застосування зазначених принципів підвищує абстрагування та покращує класифікацію предметної області, що в кінцевому рахунку дозволяти боротися зі складністю предметної області.**

**Абстрагування (abstraction) - це уявлення предметної області (програми) у вигляді меншої кількості більш великих понять (блоків) і мінімізація зв'язків між ними.**

**Класифікація - це виділення істотних, стійких відносин (зв'язків) між поняттями (блоками).**

**Поняття інкапсуляції**

**Інкапсуляція - це об'єднання даних з функціями їх обробки у поєднанні з приховуванням непотрібної для використання цих даних інформацією.**

**Інкапсуляція - це приховування реалізації класу і відділення його внутрішнього представлення від зовнішнього (інтерфейсу).**

**Поняття інтерфейсу і реалізації**

**Інтерфейс (interface) - це зовнішній вигляд класу, що виділяє його істотні риси і не показує внутрішній устрій.**

**Реалізація (implementation) - внутрішнє представлення класу, включаючи секрети його поведінки.**

**Переваги використання інкапсуляції**

**Підвищує ступінь абстракції програми – для написання програми не вимагається знання даних класу і реалізації його функцій.**

**Дозволяє змінити реалізацію класу без модифікації іншої частини програми, якщо інтерфейс залишився колишнім.**

**Поняття спадкування**

**Спадкування - це таке відношення між класами, коли один клас повторює структуру і поведінку іншого класу. Іншими словами, структура і поведінка передається від предка до нащадка.**

**Спадкування реалізує ставлення "is-a" між двома класами, тобто дочірній клас повинен бути частковим або спеціалізованим випадком батьківського класу.**

**Переваги використання спадкування**

**При спадкуванні загальні властивості і поведінка не описуються, що скорочує обсяг програми.**

**Виділення загальних рис різних класів в один клас-предок є потужним механізмом абстракції та класифікації .**

**Поняття поліморфізму**

**Поліморфізм - це використання одного імені для різних сутностей. При цьому різнорідні суті, виступаючи під одним ім'ям, сприймаються як однотипні.**

**Поліморфізм - це можливість обробки даних змінного типу.**

**Поліморфізм - це можливість оперувати об'єктами, не володіючи точним знанням їх типів.**

**Переваги використання поліморфізму**

**Дозволяє записувати алгоритми лише одного разу і потім повторно їх використовувати для різних типів даних.**

**Звужує концептуальний простір, тобто зменшує кількість інформації, яку необхідно пам'ятати програмісту.**

**Образне порівняння процедурного і об'єктно-орієнтованого підходів**

**Програма з точки зору процедурного підходу - це конвеєр, що перемелює структури даних.**

**Програма з точки зору об'єктно-орієнтованого підходу - це сукупність взаємно ввічливими об'єктів, які просять один одного про послугу.**

**В об'єктно-орієнтованому підході в центрі уваги не "порядок виконання програми", а функціональні можливості об'єктів. "Порядок", в якому ці можливості будуть реалізовані в процесі виконання конкретної системи, є вторинною властивістю.**

**Відмінності між процедурним і об'єктно-орієнтованим підходами**

| **Процедурний** | **Об'єктно-орієнтований** |
| --- | --- |
| **За допомогою покрокового уточнення вихідне завдання розбивається на все більш дрібні підзадачі, поки вони не стануть настільки простими, що їх можна буде реалізувати безпосередньо.** | **Спочатку виділяються класи, а лише потім визначаються їх методи. При цьому кожен метод пов'язаний з класом і клас відповідає за їх виконання.** |
| **Програма являє собою однорідну множину процедур, що обмінюються даними.** | **Класи надають зручний механізм кластеризації методів. Крім того приховують деталі представлення даних від будь-якого коду, крім своїх методів. Це означає, що якщо помилка програмування спотворює дані, то її легше знайти.** |
| **Неможливо отримати декілька копій одного модуля. Модуль - це набір пов'язаних даних і процедур, зібраних в окремому файлі. Модуль може мати інтерфейсну частину і реалізацію.** | **На основі класу можна створити кілька об'єктів з однаковою поведінкою** |

**Переваги ОО підходу**

* **Об'єктна модель орієнтована на людське сприйняття світу: багато людей, які не мають поняття про те, як працює комп'ютер, знаходять цілком природним об'єктно-орієнтований підхід до систем.**
* **Більш ефективна боротьба зі складністю програмного забезпечення.**
* **Більш високий відсоток повторного використання коду.**
* **Підвищення надійності програмного забезпечення.**
* **Забезпечення можливості модифікації окремих компонентів програмного забезпечення без зміни інших його компонентів .**
* **Підвищується рівень уніфікації розробки → придатність до повторного використання проектів.**
* **Об'єктно-орієнтовані системи часто є більш компактними → економія часу і ресурсів → зменшення вартості.**
* **Використання об'єктної моделі призводить до побудови систем на основі стабільних проміжних описів, що спрощує процес внесення змін.**
* **Це дає можливість розвиватися поступово і не призводить до повної її переробки навіть у разі істотних змін вихідних вимог**
* **Об'єктна модель зменшує ризик розробки складних систем, насамперед тому, що процес інтеграції розтягується на весь час розробки, а не перетворюється в одноразову подію.**
* **Об'єктний підхід складається з ряду добре продуманих етапів проектування, що також зменшує ризики і підвищує впевненість у правильності прийнятих рішень.**

Клас являє собою абстрактний тип даних, що визначається користувачем і являє собою модель реального об’єкта у вигляді даних та функцій для роботи з ними.

Оголошення класу має таку форму:

class <ім’я класу> : <список класів-батьків>

{ public:             // доступно для всіх **відкритий доступ**

<дані, методи, властивості, події>

protected: // доступно тільки нащадка **захищений доступ**

<дані, методи, властивості, події>

private:   // доступно тільки в класі **закритий доступ**

<дані, методи, властивості, події>

} <список змінних>

Об’єкт — це змінна типу клас. Дані класу називаються *полями,* а функції — *методами,* що призначені для обробки полів. Крім методів, тип клас може мати спеціальні функції — конструктори і деструктори.

Визначення або оголошення класу починається з ключового слова class. Далі у фігурних дужках ({}) описується тіло класу. Визначення класу закінчується крапкою з комою. Тіло класу включає дані (члени-дані), функції (члени-функції) і специфікатори доступу до членів класу. Специфікатори доступу до членів класу завжди закінчуються двокрапкою (:) і можуть з'являтися у визначенні класу багато разів і у будь-якому порядку.

Властивість ООП - можливість успадкування даних і функцій. У випадку, коли один клас успадковує інший, базовий клас називають **батьківським**, а той, що успадковує, — похідним або **нащадком**.

Розміщення описів змінних і функції з їх обробки, тобто полів та методів в одному класі називається **інкапсуляцією**.

Класи в С++ мають три різних рівні доступу до своїх елементів, тобто даних (полів) і функцій (методів):

* закриті елементи (private);
* захищені елементи (protected);
* відкриті елементи (public).

До даних у закритому розділі (private) мають доступ тільки функції-елементи свого класу. Класам-нащадкам забороняється доступ до закритих даних своїх базових (батьківських) класів. За замовчуванням усі дані-члени класу мають атрибут private(закритий).

До даних у захищеному розділі (protected) мають доступ функції свого класу і функції класів-нащадків.

У свою чергу до даних відкритого розділу (public) можуть звертатися будь-які функції.

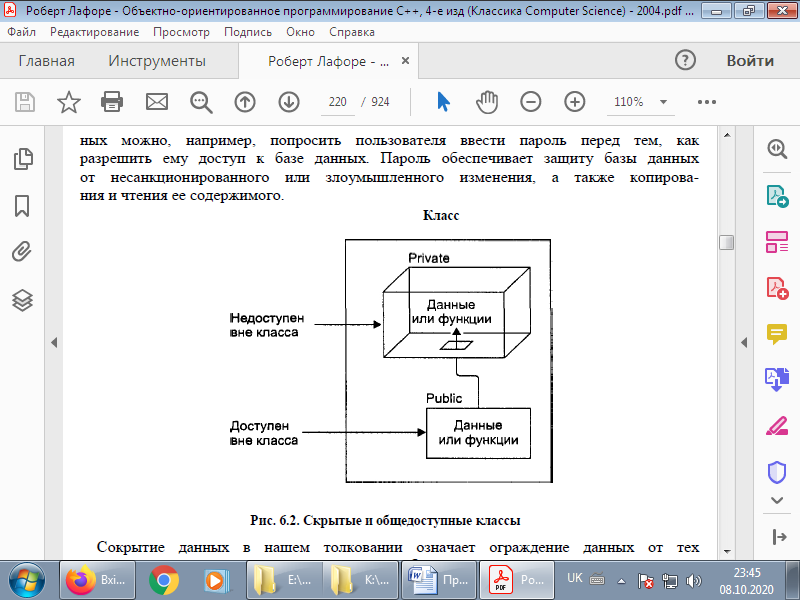


Рис.1 Приховані та загальнодоступні класи

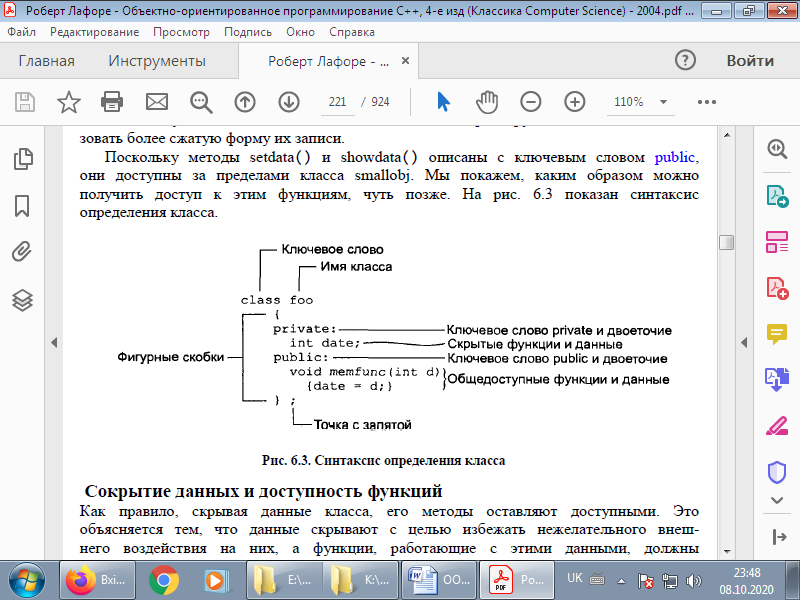


Рис.2 . Синтаксис визначення класу

**Правила створення розділів класу**:

* розділи можуть з’являтися в будь-якому порядку і декілька разів;
* якщо не оголошено жодного розділу, компілятор за замовчуванням оголошує усі елементи закритими;
* розміщати дані-елементи класу у відкритому розділі можна тільки за необхідності. Дані-елементи класу звичайно розміщують у закритому, або захищеному розділі, щоб до них мали доступ функції-члени класу, а також функції класів-нащадків;
* для зміни значень даних (полів) слід використовувати функції-члени класу;
* клас може мати декілька конструкторів, але тільки один деструктор.

  Після того, як клас визначений, і його функції-члени оголошені, ці функції-члени повинні бути визначені (описані). Кожна функція-член може бути описана прямо в тілі класу або після тіла класу. Коли функція-член описується після відповідного визначення класу, імені функції передує ім'я класу і оператор дозволу області дії (::). Оскільки різні класи можуть мати члени з однаковими іменами, оператор дозволу області дії «прив'язує» ім'я члена до імені класу, щоб однозначно ідентифікувати функції-члени даного класу.

Розглянемо приклад використання класу.

*Приклад 11.1.* Навести просту програму з використанням класу.

// P11\_1.CPP — использование класса, структура класса

#include <iostream.h>

#include <conio.h>

class myclass {

int p;                // закрытая переменная no умолчанию

public:

void set\_p(int x);        // прототипы функций-членов класса

int get\_p();

}; //---------- описание функций-членов класса

void myclass::set\_p(int x)

{ p = x; }

int myclass::get\_p( )

{ return p;}

main()

{ myclass ob1, ob2;        // объявление объектов

// вызов функций-членов класса

ob1.set\_p(10);

ob2.set\_p(30);

cout<<"p1= "<<ob1.get\_p()<<" ";

cout<<"p2= "<<ob2.get\_p()<<endl;

getch ();

return 0; }

Результати обчислень: **p1= 10 p2= 30**

У програмі оголошено клас з ім’ям myclass, що має одну закриту змінну р і дві відкриті функції-члени класу set\_p() і get\_p(). Перша з функцій призначена для ініціювання закритої змінної р, а друга — для її повернення. В описі класу оголошено тільки прототипи функцій, їх реалізацію наведено за межами опису. Але якщо функція складається з декількох операторів, її можна розмістити всередині класу. Оскільки ці функції є членами класу, то вони мають доступ до закритої змінної р. У випадку, коли функції-члени описано за межами класу, їхні заголовки повинні складатися з імені класу, операції розширення області класу «::», імені функції та її формальних ар­гументів, якщо вони є, а далі йтиме зміст функції. У головній функції main() спочатку оголошуються два об’єкти типу myclass, потім ці об’єкти ініціюються конкретними зна­ченнями змінної р (ob1.set\_p(10) та ob2.set\_p(30),після цього значення цієї змінної виводяться на екран (cout <<ob1.get\_p(); та cout<<ob2.get\_p0).

***Елементи класу записуються через крапку після імені об’єкта.***

За замовчанням доступ до членів класу – private. Доступ до відкритих членів класу мають будь-які функції в програмі, доступ до закритих членів класу мають тільки члени-функції даного класу і друзі класу. Дані-члени класу зазвичай робляться закритими, а функції-члени – відкритими. Глибокий сенс такого підходу полягає в тому, що дійсне представлення даних усередині класу не торкається клієнтів (користувачів) класу. Клієнти класу використовують клас, не знаючи внутрішніх деталей його реалізації. *Якщо реалізація класу змінюється, інтерфейс класу залишається незмінним і дійсний код клієнта класу не вимагає змін*. Це значно спрощує модифікацію систем.

З того, що дані класу закриті, не витікає, що клієнти не можуть змінювати ці дані. Дані можуть бути змінені функціями-членами або друзями цього класу. Наприклад, щоб дозволити клієнтам класу прочитати закрите значення даних, клас може мати функцію «одержати» (get). Щоб дати клієнтам можливість змінювати закриті дані, клас може мати функцію «встановити» (set). Таким чином, доступ до закритих членів класу здійснюється через відкриті члени-функції даного класу.

Основне завдання відкритих членів полягає в тому, щоб дати клієнтам класу уявлення про можливості (послуги), які забезпечує клас. Цей набір послуг складає відкритий інтерфейс класу. Клієнтів класу не повинно торкатися, яким чином клас виконує їх завдання. Закриті члени класу і опис відкритих функцій-членів недоступні для клієнтів класу. Ці компоненти складають реалізацію класу.

Отже, клас - це структурований тип даних, визначений користувачем. Коли клас визначений, ім'я класу може бути використано для оголошення змінної цього типу , які називають екземплярами класу або об'єктами. Функції, що оголошені усередині класу, викликаються ззовні об'єкта даного класу за допомогою оператора вибору елемента «.» або оператора вибору елемента за допомогою покажчика «->».

Наприклад, об'єкт товар характеризується двома атрибутами: найменуванням і ціною. Необхідно ввести з клавіатури дані про товар і вивести їх на екран. Отже, для об'єкта товар можна оголосити клас, що буде містити два поля даних (найменування товару й ціну), і два методи (уведення даних з клавіатури і вивід даних на екран).

**Поняття класу і поняття об‘єкту**

**Клас -** це визначений програмістом тип. Оголошення класу задає представлення об'єктів цього класу і набір операцій, які можна застосовувати до таких об'єктів.

Якщо клас – це шаблон, то об’єкт це конкретний представник класу, створений за цим шаблоном.

**X x; // Створюємо змінну х – об’єкт класу (типу) Х**

Об'єкти класу можна створювати та ініціалізувати, знищувати, присвоювати, передавати в якості параметрів функції і повертати як результат виконання функції.

***Ініціалізація не дорівнює Створенню.***

**Члени класу**

У списку членів класу можна оголошувати змінні, функції, класи, перерахування, а також дружні функції та класи.

* ***Не можна двічі оголошувати члена класу в списку членів***
* ***Не можна оголосити в класі змінну та функцію з одним ім'ям.***
* ***Членом класу не може бути об’єкт цього ж класу, але може бути покажчик або посилання на об'єкт того ж класу.***

**Різниця між struct і class**

Члени класу без спеціфікатора доступу за замовчанням є приватними. Члени структур та об'єднань за замовчуванням є публічними

**Селектори і модифікатори**

**Селектори (Get)** — методи класу, що повертають значення приватного атрибута (даних);

**Модифікатори (Set)** — методи класу, що встановлюють коректне значення приватного атрибута(даних).

Як добратися до атрибутів, якщо вони закриті? ― За допомогою методів доступу: селекторів і модифікаторів

Селектор

double getX() { return **\_x**; };

Модифікатор

void setX (double x) { \_x = x;}

**Покажчик this**

Ключове слово **this** позначає вказівник на об'єкт, для якого викликана ця функція, тобто всередині функції класу член того ж класу з ім'ям **х** можна позначати як **x,** і як **this->x.** Щоб забезпечити роботу методу з полями того об’єкта, для якого він був викликаний, в метод автоматично передається прихований параметр this, в якому зберігається посилання на об’єкт, що викликав функцію.

**Використання явного this**

У явному вигляді параметр this застосовується:

1) щоб повернути з метода посилання на об’єкт, що його викликав:

class Demo

{ double y;

public Demo T() { return this; }

2) для ідентифікації поля, якщо його ім’я співпадає з

іменем параметра метода:

public void Set\_y( double y ) { this.y = y; }

Покажчик на об'єкт, для якого викликана функція, є неявним параметром цієї функції.

**Особливості використання явного this**

* покажчик this у функції-члена класу Х має тип X\*.
* покажчик this у константного методу має тип const X\*.
* this не має адреси &this
* this нічого не можна присвоїти \*this=.
* значення this можна запам’ятати у іншій змінній цього ж типу =\*this;

Метод класу може повертати за допомогою покажчика **this** посилання на об'єкт класу відповідного об'єкта return **\*this**. Покажчик **\*this** це прихований константний покажчик адреси об‘єкту, який викликає метод класу.

Покажчик **\*this** завжди вказує на поточний об‘єкт.

Розглянемо ще приклад, де ми не передаємо параметри в методи ні при визначенні, ні при виклику методів, тобто не вказуємо з якими параметрами працюватимуть методи класу. При виклику методів, які належать класу, їм неявно передається той самий покажчик this**.** Це відбувається автоматично і непомітно для нас, оскільки він є прихованим першим параметром будь-якого методу-елемента класу. Покажчик this вказує на адресу створеного об'єкта класу – в нашому випадку він отримує адреса об'єкта object і вказує методу setData() в які елементи об'єкта треба внести змінені дані, а методу getData() – елементи якого об'єкта треба відобразити на екрані. Можна використовувати покажчик this явно, ці способи надаються в за коментованих рядках тексту функції getData():

#include <iostream>

using namespace std;

class SetGetData // Визначаємо клас

{

char str[128];

int number;

public:

void setData() // Метод класу, що приймає дані

{

cout << "Input line: ";

cin.getline(str, 128);

cout << "Input integer: ";

cin >> number;

}

void getData() // Метод класу, що надає дані

{

cout << "\n 1 " << str << number << endl;

cout << "\n 2 " << this->str << this->number << endl ;

cout << "\n 3 " << (\*this).str << (\*this).number << endl;

//Використання покажчика (\*this) при виведенні даних

}

};

int main()

{ system("color F0");

SetGetData object; // Створити об‘єкт

object.setData(); // Вносимо дані

system("cls"); // Очищуємо екран перед виведенням

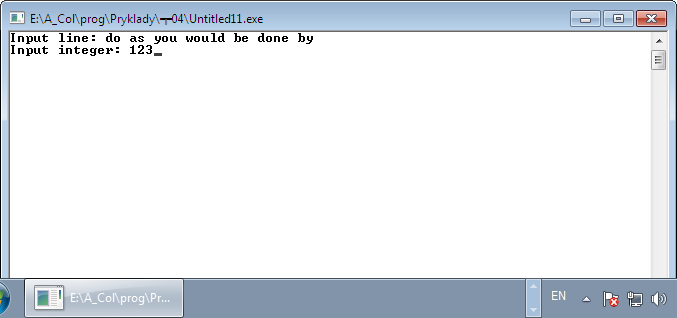
object.getData(); // Дані відображуємо на екрані

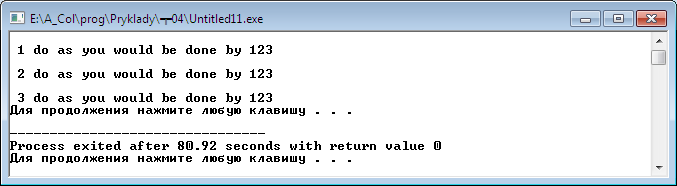
system("pause");

return 0;

}

**Результати виконання**





***Контрольні запитання*.**

1. Назвіть основні засади об’єктно-орієнтованого програмування
2. Дайте визначення об’єкта та чим він характеризується?
3. Назвіть фундаментальні властивості об'єктно-орієнтованого програмування.
4. Дайте визначення інкапсуляції, спадкуванню та поліморфізму
5. Назвіть переваги об'єктно-орієнтованого програмування.
6. Дайте визначення класу та об’єкту в ООП
7. Що може містити оголошення класу?
8. Як оголосити клас на мові С++?
9. Де може бути визначена (описана) функція класу?
10. Порівняйте виклик члена-функції класу у функціях класу та поза функціями цього класу.
11. Назвіть специфікатори доступу до членів класу.
12. Чи може з'являтися у визначенні класу один і той же специфікатор доступу багато разів?
13. У якому порядку можуть з'являтися специфікатори доступу до членів класу?

**Для самостійного вивчення** *(2 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

**Рекомендована література**

1. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL: <http://www.ph4s.ru/bookprogramir_1.html>
2. Страуструп Б. Язык программирования С++. — СПб.; М.: Невский диалект — ЗАО “Изд-во “Бином”, 1999.
3. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

4. Липпман С. Б., Лажойе Ж. Язык программирования С++: Вводный курс. — М.: ДМК, 2001. URL: <http://www.insycom.ru/html/metodmat/inf/Lipman.pdf>

5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL: <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>