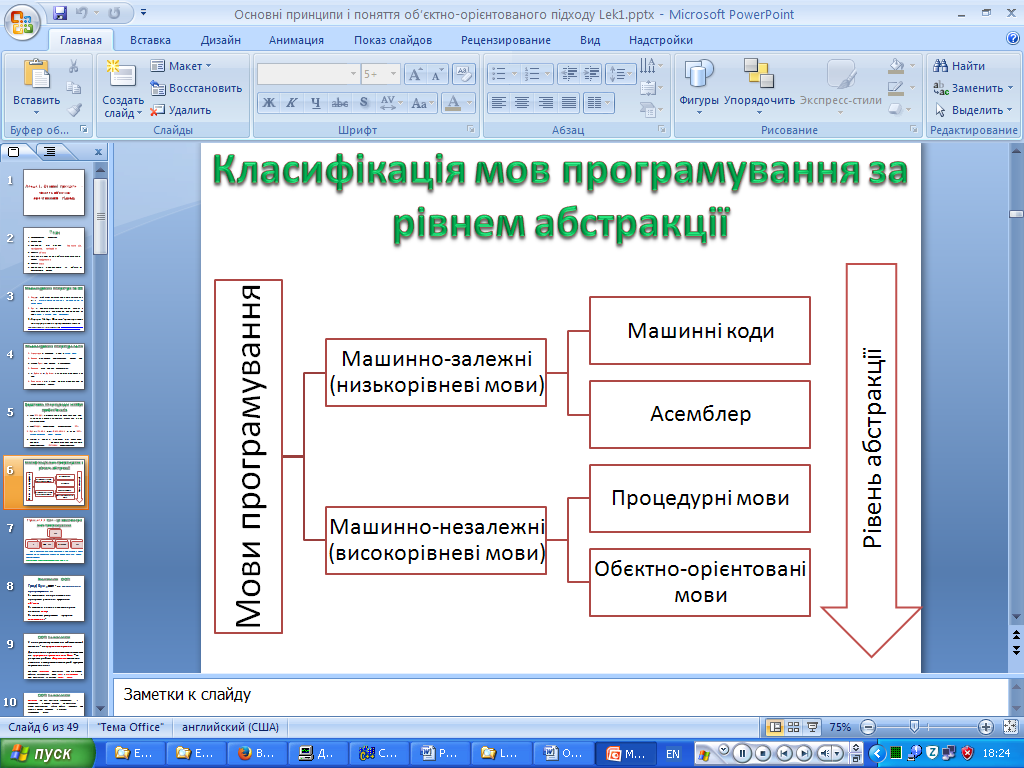
**Лекція №9. Базові поняття програмування мови ООП. Інкапсуляція та приховування інформації.**

****

***Мови програмування низького рівня*** (*машинно-залежні або машинно-орієнтовані*) - такі, у яких принципи керування і структура даних безпосередньо відображають архітектуру ЕОМ. Тобто, такі мови орієнтовані на конкретний тип процесора і враховують його особливості.

Група мов низького рівня включає машинні мови (машинні коди), *мови символічного кодування* (Асемблери) та ряд інших. Програми, написані на таких мовах програмування, представляють собою лінійні послідовності елементарних операцій з регістрами, в яких зберігаються дані.

Зокрема, мова Асемблера представляє кожну команду машинного коду у вигляді спеціальних символьних позначень, які називаються *мнемоніками*, а в якості операндів використовує символічні імена, а не конкретні адреси. Це допомагає програмісту легше запам'ятовувати смисловий зміст операції та забезпечує суттєве зменшення кількості помилок при складанні програм.

Оскільки системи команд різних моделей мікропроцесорів різняться, конкретній архітектурі ЕОМ відповідає своя мова асемблера, і написана на ній програма може бути використана тільки на тих ЕОМ, для яких вона розроблена. Тому асемблери і вважаються машинно-орієнтованими мовами програмування.

Асемблери застосовуються системними програмістами-професіоналами з метою використання всіх можливостей устаткування ЕОМ і отримання ефективної за часом виконання і по необхідному обсягу пам'яті програми. На цих мовах зазвичай розробляються відносно невеликі програми, що входять до складу системного програмного забезпечення: драйвери, утиліти тощо.

***Мови програмування високого рівня*** - такі, у яких засоби керування і роботи з даними відображають потреби програміста, а не можливості апаратних засобів. Програми, що складаються на таких мовах, являють собою послідовності операторів, структуровані відповідно до правил мови. Вони оперують сутностями, ближчими і зрозумілішими людині, такими як змінні, функції тощо. Особливості конкретних комп’ютерних архітектур в них не враховуються, тому розроблені на цих мовах програми можуть виконуватися на інших комп’ютерах тієї ж платформи.

Розробляти програми на високорівневих мовах програмуванняза допомогою зрозумілих команд значно простіше, а помилок при їх розробці допускається набагато менше. При цьому програмісти отримали можливість не розписувати в деталях обчислювальний процес на рівні машинних команд, а зосередитися на основних особливостях алгоритму. Саме тому такі мови програмування називають *алгоритмічними*.

Основні відмінності мови програмування високого рівня від машинно-орієнтованої :

* алфавіт алгоритмічної мови значно ширше алфавіту машинно-орієнтованої, що істотно підвищує наочність тексту такої програми;
* набір допустимих в алгоритмічній мові операцій не залежить від набору машинних операцій, а вибирається із міркувань зручності формулювання послідовності дій;
* формат речень у алгоритмічній мові досить гнучкий, що дозволяє за допомогою одного речення задати досить змістовний етап обробки даних;
* необхідні операції в алгоритмічній мові задаються в зручному для людини вигляді, наприклад, за допомогою загальноприйнятих математичних позначень;
* в алгоритмічній мові може бути передбачений значно більший набір типів даних у порівнянні з набором машинних типів даних.

Наочність і потужні можливості високорівневих мов програмуванняобумовили саме їх використання для вирішення різноманітних прикладних задач. Алгоритмічними мовами програмування високого рівня, наприклад, є:

**Fortran** (*Фортран*). Створена у IBM під керівництвом Дж. Бекуса (1954-57 р.р.). По суті це перша широко застосовувана мова високого рівня, орієнтована на науково-інженерні і чисельні задачі. Для цієї мови було створено величезну кількість бібліотек, починаючи від статистичних комплексів і закінчуючи пакетами управління супутниками, тому до цих пір Фортран продовжує активно використовуватися в деяких організаціях.

**Ваsic** (*Бейсик*). Багатоцільова мова символічних інструкцій, розроблена у 1965 р.. Створена як учбова мова для початківців. Згідно концепціям, закладеним у Basic, ця мова в сенсі строгості і стрункості є антиподом мови Pascal. Зокрема, в ній широко поширені різні правила умовчання, що вважається поганим тоном в більшості мов програмування подібного типу. Існує багато діалектів цієї мови, мало сумісних між собою.

**Раsсаl** (*Паскаль*). Створена Н. Віртом у 1970 р. спеціально для вивчення програмування. Пред'являє жорсткі вимоги до структури програми, підтримує велику кількість призначених для користувача типів даних. Крім того, в мові реалізована концепція визначення нових типів даних на основі вже наявних.

**С**. Створена Д.Рітчі на початку 1970-х років для розробки операційної системи UNIX. Має засоби для прямої роботи з пам'яттю. Була задумана як мова системного програмування для заміни асемблера, щоб мати можливість створювати такі ж ефективні і короткі програми, але не залежати від конкретного процесора. Вона є найпопулярнішою мовою для створення системного програмного забезпечення. Однак великий набір операцій і типів даних, сучасне оформлення і висока ступінь машинно-незалежності зробили її привабливою мовою програмування загального призначення. Незважаючи на те, що С не розроблялася для новачків, вона активно використовується для навчання програмуванню. Надалі синтаксис мови С став основою для багатьох інших мов. На мові С написана безліч прикладних і системних програм і ряд відомих ОС (зокрема, UNIX).

Подальшим розвитком ідеї алгоритмічної мови стали мови програмування більш загального, не обов'язково алгоритмічного характеру. Як і алгоритмічні мови, такі мови, зрештою, теж націлені на отримання машинних програм, але в багатьох випадках їх тексти допускають певну свободу у виконанні і, як правило, дають лише матеріал для синтезу шуканих алгоритмів, а не самі ці алгоритми. Такими мовами , наприклад, є:

**С++.** Об'єктно-орієнтоване розширення мови С, створене Б.Страуструпом в 1980 р. Поєднує властивості як високорівневих, так і низькорівневих мов. Заснована на використанні класів і об'єктів. При створенні мови С++ прагнули зберегти її синтаксис, сумісний з мовою С. Більшість програм, написаних мовою С, справно працюють і з компілятором мови С++. Нововведеннями мови С++ порівняно з мовою С підтримка об'єктно-орієнтованого програмування через класи і об'єкти, підтримка узагальненого програмування через шаблони, доповнення до стандартної бібліотеки, додаткові типи даних, обробка виключень (виключних ситуацій), простори імен, вбудовані функції, перевизначення операторів та імен функцій, посилання та оператори управління вільно розподіленою пам'яттю.

**Javа**. Створена компанією SunMicrosystems на початку 90-хроків на основі С++. Покликана спростити розробку застосувань на С++ шляхом виключення із нього низькорівневих можливостей. Головна особливість мови – те, що вона транслює програму не в машинний код, а в платформно незалежний байт-код (кожна команда займає один байт). Цей байт-код може виконуватися за допомогою інтерпретатора- віртуальної Javа–машини JVM (*Javа Virtual Масhine*), версії якої створені для будь-яких апаратних платформ. Завдяки цьому програми на Javа можна переносити не лише на рівні вихідних текстів, але і на рівні двійкових байт-кодів. Це дозволяє створювати незалежні програмні модулі, здатні працювати на серверах в глобальних і локальних мережах з різними ОС.

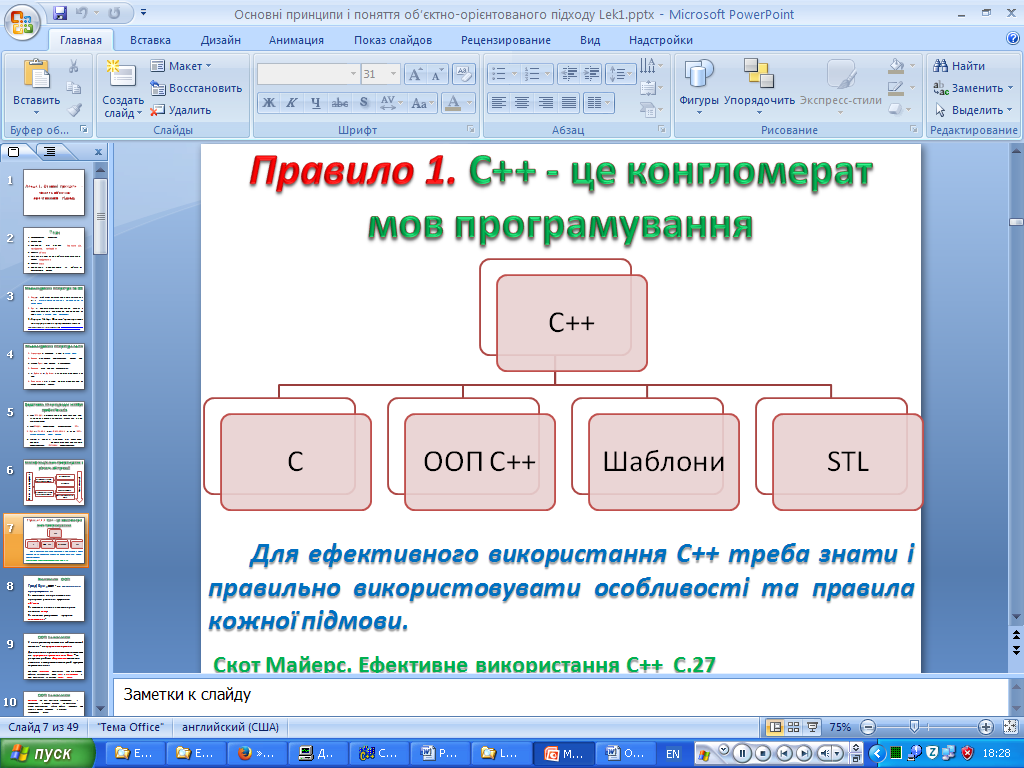
**C#.** Відноситься до сім'ї мов із C-подібним синтаксисом, з них її синтаксис найбільш близький до C++ і Java.

**Perl.** Інтерпретована високорівнева мова програмування загального призначення. Головною особливістю мови є її можливості для роботи з текстом, у тому числі реалізовані за допомогою регулярних виразів.

**Python.** Мова програмування загального призначення з акцентом на продуктивність розробника і читаність коду. Синтаксис ядра **Python** мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

**Ruby.** Інтерпретована високорівнева мова програмування для швидкого і зручного об'єктно-орієнтованого програмування. Мова має незалежну від операційної системи реалізацію багатопоточності і багато інших можливостей. **Ruby** близька за особливостями синтаксису до мови **Perl**, за об'єктно-орієнтованим підходом – до **Smalltalk**. Також деякі риси мови взяті з **Python**,**Lisp** та інших.

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



**Шаблони** — засіб мови C++, який призначений для кодування узагальнених алгоритмів, без прив'язки до деяких параметрів: типу даних, розміру буфера та стандартного значення. В C++ можливе створення шаблону функції і шаблону класу. Хоча шаблони надають коротку форму запису ділянки коду, насправді їх використання не скорочує виконуваний код, тому що для кожного набору параметрів компілятор створює окремий екземпляр функції або класу.

**Стандартна бібліотека шаблонів** (*Standard Template Library*; **STL**) — бібліотека для C++, що містить набір узгоджених узагальнених алгоритмів, контейнерів, засобів доступу до їхнього вмісту і різних допоміжних функцій.

***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 наслідування. ”**

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 програмного забезпечення - аж ніяк не випадкова його властивість". Брукс**

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**

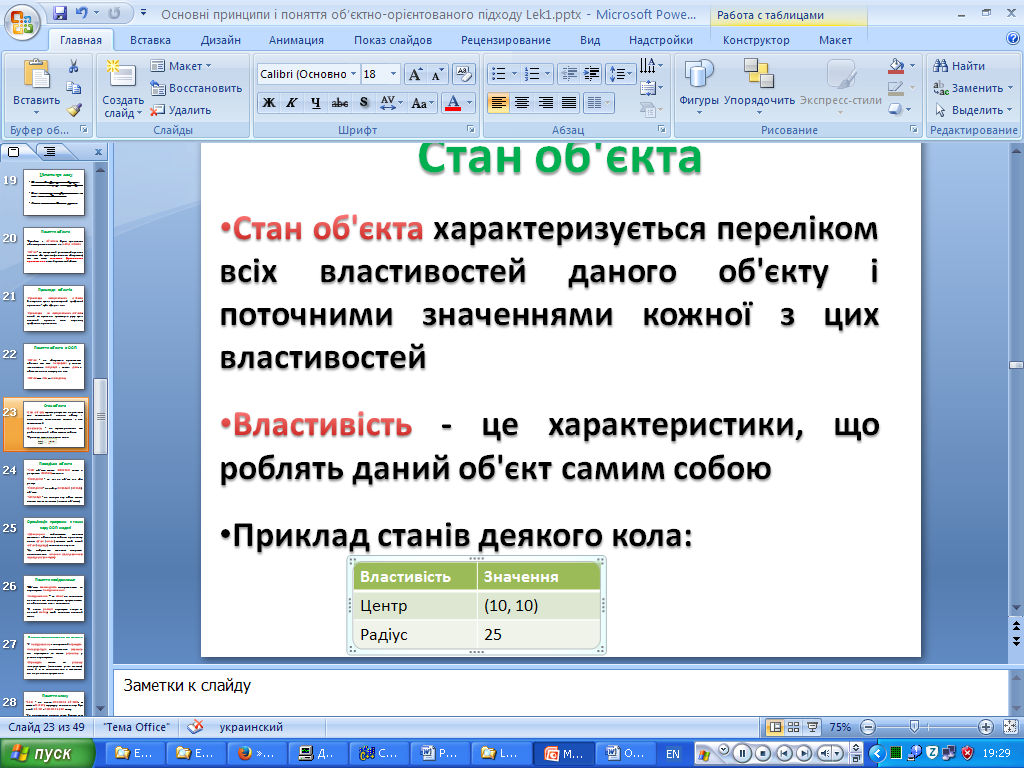
* Будь-що є **об'єктом.** Будь предметна область розглядається як набір **об'єктів**
* **Об'єкт -** це конкретний упізнаваний предмет, одиниця або сутність (реальна чи абстрактна), що має чітко визначене функціональне призначення в даній предметній області

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**

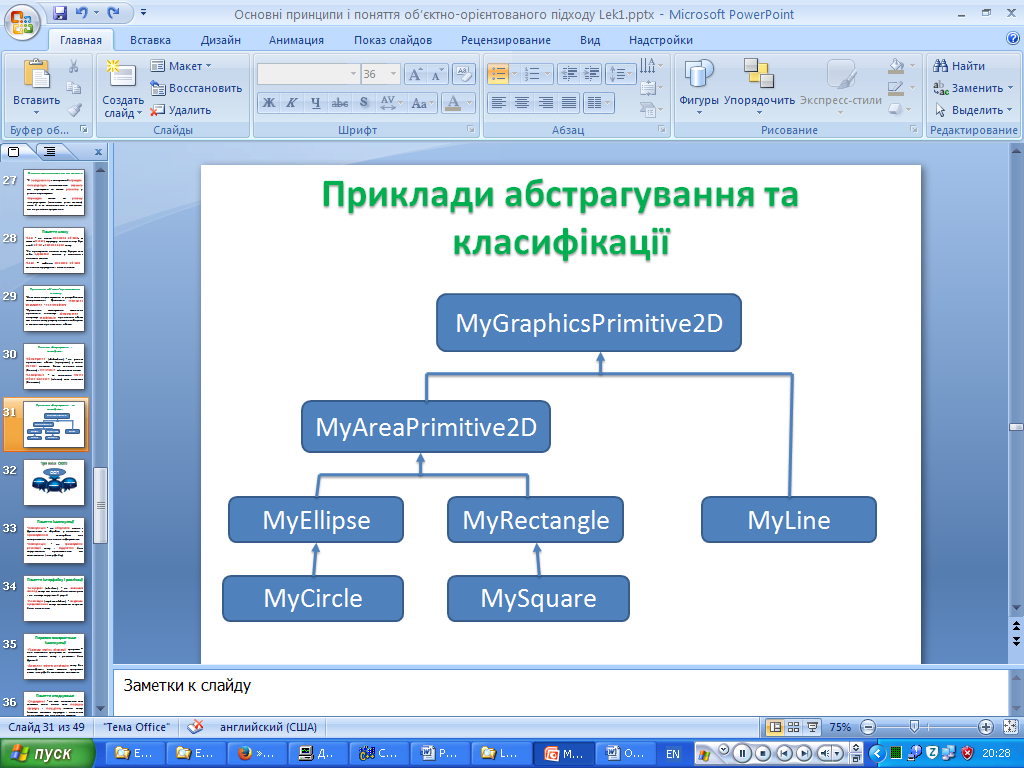
**Клас -** це деяка **множина об'єктів,** що мають **спільну структуру** та **поведінку.** Будь-який об'єкт є екземпляром класу.

Усі екземпляри 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 повинен проектуватися и розроблятися з використанням 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 наслідування / спадкування**

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

**Наслідування** реалізує ставлення "is-a" між двома класами, тобто **дочірній** класповинен бути **частковим** або спеціалізованим випадком **батьківського** класу. Відношення типу is-a (є, являється) при якому один клас є підвидом іншого класу. При такому співвідношенні один клас розширює (деталізує) можливості іншого класу.

**Переваги використання наслідування**

При наслідуванні загальні властивості і поведінка не описуються в базовому класі (батьківському класі, класі-предку від якого ведеться наслідування) , 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** їхвикористовувати 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 реалізуватибезпосередньо. | 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 якщопомилка 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.

**З роботи Бадда. ООП в дії**

**Короткий виклад принципів**

1. Все є об‘єктом.

2. Обчислення здійснюються шляхом взаємодії (обміну даними) між об‘єктами, при який один об‘єкт вимагає, щоб інший об‘єкт виконав деяку дію. Об‘єкти взаємодіють, посилаючи та отримуючи повідомлення.

Повідомлення — це запит на виконання дії, доповнений набором аргументів, які можуть знадобиться при виконанні дії.

3. Кожний об‘єкт має незалежну пам‘ять, яка складається з інших об‘єктів.

4. Кожний об‘єкт є представником класу, який виражає загальні властивості об‘єктів (таких, як цілі числа або списки).

5. В класі задається поведінка (функціональність) об‘єкта. Таким чином всі об‘єкти, які є екземплярами одного класу, можуть виконати одні і ті ж самі дії.

6. Класи організовані в єдину деревовидну структуру із загальним коренем, яку називають ієрархією наслідування. Пам‘ять та поведінка, зв‘язані з екземплярами визначеного класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

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 <сім’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.

Властивість ООП - можливість наслідування 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.

***Контрольні запитання для самоперевірки***.

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

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

**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>

6. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++ . [2-е изд.] / Буч Г. - СПб.: Невский Диалект, 1998. - 560 с.

7. Бадд Т. Объектно-ориентированное программирование в действии. [2-е изд.] – СПб.: Изд-во "Питер". 1997. URL: http://khizha.dp.ua/library/Timothy\_Budd\_-\_Introduction\_to\_OOP\_(ru).pdf