

МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ

Харківський національний університет внутрішніх справ

Факультет № 4

Кафедра інформаційної безпеки

ТЕКСТ ЛЕКЦІЇ

з дисципліни «Об’єктно-орієнтоване програмування»

за темою: «Основні поняття об’єктно-орієнтованого програмування»

Галузь знань - 1701 «Інформаційна безпека»

Напрямок підготовки - 6.170102 - „Системи технічного захисту інформації”

Ступень вищої освіти - бакалавр

**м. Харків
2016 рік**

Передмова

Текст лекції з дисципліни «Об'єктно-орієнтоване програмування» за темою «Основні поняття об'єктно-орієнтованого програмування» для курсантів за напрямом підготовки 6.170102 - «Системи технічного захисту інформації» на 11 арк.

СХВАЛЕНО

Науково-методичною радою
Харківського національного
університету внутрішніх справ
26.12.2016 Протокол № 10

ЗАТВЕРДЖЕНО

Вченою радою факультету № 4
15.11.2016 Протокол № 12

_____ Марков В.В.

ПОГОДЖЕНО

Секцією науково-методичної ради
з технічних дисциплін
Харківського національного
університету внутрішніх справ
12.12.2016 Протокол № 11

ЗАТВЕРДЖЕНО

На засіданні кафедри інформаційної
безпеки

14.11.2016 Протокол № 8

_____ Сезонова І.К.

_____ Сезонова І.К.

Рецензенти:

Нефьодов Л.І., завідувач кафедри АКТ ХНАДУ, д.т.н., професор

Гнусов Ю.В., завідувач кафедри кібербезпеки ХНУВС, к.т.н., доцент

Розробники: Ільге І.Г., - Харків: Харківський національний університет
внутрішніх справ, 2016р.

© Ільге І.Г., 2016

© Харківський національний університет внутрішніх справ

План лекції

- 1. Потреба використання об'єктно-орієнтованого програмування**
- 2. Об'єктно-орієнтований підхід до розроблення складних програм**
- 3. Основні компоненти об'єктно-орієнтованої мови програмування**

Література:

Основна:

1. Объектно-ориентированный анализ и проектирование с примерами приложений / [Гради Буч, Роберт Максимчук, Майкл Энгл и др.] ;пер. с англ. – М. : ООО "ИД Вильямс", 2008. – 720 с.
2. Кватрани Т. Rational Rose 2000 и UML. Визуальное моделирование. Пер. с англ. - М., ДМК Пресс, 2001. 176 с
3. Леоненков А. Самоучитель UML 2 / Александр Леоненков. – СПб. : BHV, 2007 – 576 с.

Додаткова:

4. Скотт К., Фаулер М. UML. Основы- СПб: "Символ-Плюс", 2002. - 192 с.
5. Буч Г., Джекобсон А., Рамбо Дж. Язык UML. Руководство пользователя - М.: ДМК, 2000. - 432 с.
6. Джеанини М., Кьюу Дж. Объектно-ориентированное программирование. Учебный курс - СПб: "Питер", 2005.- 238 с.

ТЕКСТ ЛЕКЦІЇ

1. Основні поняття об'єктно-орієнтованого програмування

1.1. Потреба використання об'єктно-орієнтованого програмування

Розвиток об'єктно-орієнтованої технології створення сучасних програмних продуктів пов'язаний деякими обмеженими можливостями інших технологій програмування, які широко застосовувалися раніше. Аби краще зрозуміти і оцінити значення ООП, необхідно з'ясувати, у чому ж полягають ці обмеження та як вони відображаються в традиційних мовах програмування.

1. Процедурні мови програмування. Кожен оператор процедурної мови дає вказівку комп'ютеру виконати певну дію чи їх послідовність, наприклад, прийняти дані від користувача, виконати з ними певні арифметичні операції чи логічні дії та вивести одержаний результат на екран чи принтер. Програми, написані процедурними мовами, складаються з послідовності певних настанов. Для невеликих програм не вимагається додаткової внутрішньої їх організації. Програміст записує перелік настанов, а комп'ютер здійснює дії, які відповідають цим настановам. До процедурних мов програмування відносяться Pascal, Visual Basic, C чи інші схожі з ними мови.

2. Поділ програми на функції та модулі (структурний підхід). Коли розмір програми зростає, то зміст виконуваних настанов стає надзвичайно громіздким і заплутаним. На сьогодні професійних програмістів, здатних запам'ятати більше 500 рядків коду програми, яку не розділено на дрібні логічні частини, є не так вже й багато. Застосування ж відповідних функцій користувача значно полегшує сприйняття коду програми під час його аналізу. Програмний код, побудований на основі структурного підходу, ділиться на відповідні функції, кожна з яких в ідеальному випадку здійснює певну завершену послідовність дій і має явно виражені зв'язки з іншими функціями коду програми.

Розвиток ідеї поділу коду програми на функції користувача призвів до того, що їх почали об'єднувати по декілька в програмний модуль, який можна записати окремим файлом. Проте навіть при такому підході зберігається структурний принцип: код програми ділиться на декілька структурних компонент, кожна з яких є набором відповідних настанов.

Поділ коду програми на функції та модулі є основою технології структурного програмування, яка протягом багатьох десятиліть, доки не було розроблено концепцію ООП, залишалася важливим способом організації кодів програм і популярною методикою розроблення програмного забезпечення.

Недоліки технології структурного програмування. У безперервному зростанні розміру програми, повсякчасному ускладненні її логіки чи виконуваних нею дій розробники програмних продуктів поступово почали виявляти недоліки технології структурного програмування.

По-перше, існують обмежені можливості доступу функцій до глобальних даних. По-друге, поділ програми на глобальні дані та функції, що є основою технології структурного програмування, погано відображає "картину реального світу" – фізичну сутність технічного завдання.

Неконтрольований доступ до даних. У структурній програмі існує два типи даних. Локальні дані знаходяться усередині будь-якої функції та призначені для використання тільки нею. Локальні дані функції недоступні нікому, окрім неї самої, і не можуть бути змінені іншими функціями.

Якщо існує потреба сумісного використання одних і тих самих даних декількома функціями, то ці дані мають бути оголошені як глобальні. Це, як правило, стосується тих даних програми, які є найважливішими. Будь-яка функція має доступ до глобальних даних. Схему, яка ілюструє концепцію області видимості локальних і глобальних даних, наведено на рис. 1.1.

Великі за розміром програми, як правило, містять багато різних функцій і згрупованих глобальних даних. Проблема структурного підходу полягає в тому, що кількість можливих зв'язків між різними групами глобальних даних і використовуваними функціями може бути дуже великою.

Велика кількість зв'язків між функціями і групами даних, як правило, породжує декілька додаткових проблем. По-перше, ускладнюється структура коду програми. По-друге, у код програми важко вносити нові зміни. Окрім цього, будь-яка зміна структури глобальних даних може вимагати коригування всіх функцій, які використовують ці дані.



Рис. 1.1. Концепція області видимості глобальних і локальних даних

Коли зміни вносяться у глобальні дані великих програм, то не завжди можна швидко визначити, які функції при цьому необхідно скоригувати. Навіть тоді, коли це вдається зробити оперативно, то згодом через значну кількість зв'язків між функціями і даними виправлені функції починають некоректно працювати з іншими глобальними даними. Таким чином, будь-яка зміна у коді програми призводить до виникнення негативних подальших дій і, як наслідок, появи відповідних проблем.

Другий, набагато важливіший недолік структурного програмування полягає в тому, що відокремлення даних від функцій виявляється малопридатним для достовірного "відображення картини реального світу", тобто, для адекватного відтворення фізичного змісту технічного завдання. Йдеться про те, що у реальному світі доводиться мати справу з фізичними об'єктами, такими, наприклад, як люди чи машини. Ці об'єкти не можна віднести ні до даних, ні до функцій, оскільки реальні речі характеризує сукупність певних властивостей чи їх поведінку.

Прикладами властивостей (іноді їх називають характеристиками) для людей можуть бути колір очей або місце роботи; для машин – потужність двигуна і кількість дверей. Таким чином, властивості об'єктів рівносильні даним у програмах: вони набувають певні значення, наприклад карий – для кольору очей або 4 – для кількості дверей автомобіля.

Поведінка – це певна реакція фізичного об'єкта у відповідь на зовнішню дію. Поведінка схожа з роботою функції: викликається функція для того, аби зробити якусь дію (наприклад, вивести на екран обліковий запис), і функція здійснює цю дію.

Таким чином, ні окремо взяті глобальні дані, ні відокремлені від них функції не здатні адекватно відображати фізичні об'єкти реального світу.

1.2. Об'єктно-орієнтований підхід до розроблення складних програм

Визначальною ідеєю об'єктно-орієнтованого підходу до розроблення сучасних програмних продуктів є поєднання даних і дій, що виконуються над ними, в єдине ціле, яке називається об'єктом.

Функції об'єкта, які у мові програмування C++ називаються методами або функціями-членами класу, як правило, призначені для доступу до даних об'єкта та виконання певних дій над ними. Якщо необхідно змінити значення даних об'єкта, то, очевидно, ця дія також має бути покладена на відповідну функцію об'єкта. Ніякі інші функції не можуть змінювати значення даних об'єкта. Такий підхід полегшує написання, налагодження та використання програми. Таким чином, типова програма, написана мовою C++, складається із сукупності об'єктів, що взаємодіють між собою за допомогою методів, які викликають один одного.

Для кращого розуміння призначення об'єктів, уявімо собі деяку промислову фірму, яка складається з бухгалтерії, відділу кадрів, відділу реалізації продукції, відділів головного технолога та головного механіка і т.д. Поділ фірми на відділи є важливою частиною структурної організації її виробничої діяльності. Для більшості фірм (за винятком невеликих) в обов'язки окремого співробітника не входить вирішення одночасно кадрових, виробничих і обліково-бухгалтерських питань. Різні обов'язки чітко розподіляються між підрозділами, тобто у кожного підрозділу є дані, з якими він працює: у бухгалтерії – заробітна плата, у відділі реалізації продукції – інформація, яка стосується торгівлі, у відділі кадрів – персональна інформація про співробітників, у відділі головного технолога – стан

технологічного процесу виготовлення продукції, у відділі головного механіка – технічний стан обладнання та устаткування, транспортних засобів і т. д. Співробітники кожного відділу виконують операції тільки з тими даними, які їм належать.

Структурний поділ обов'язків дає змогу легко стежити за виробничою діяльністю фірми та контролювати її, а також підтримувати цілісність інформаційного простору фірми. Наприклад, бухгалтерія несе відповідальність за інформацію, яка стосується нарахування заробітної плати, сплати податків, здійснення обліку матеріальних цінностей тощо. Якщо у менеджера з реалізації продукції виникне потреба дізнатися про загальний оклад співробітників фірми за деякий місяць, то йому не потрібно йти в бухгалтерію і ритися в картотеках; йому достатньо послати запит бухгалтеру з нарахування заробітної плати, який має знайти потрібну інформацію, обробити її та надати достовірну відповідь на запит. Така схема організації роботи фірми забезпечує правильне оброблення даних, а також здійснює їх захист від можливої дії сторонніх осіб. Аналогічні об'єкти коду програми створюють таку саму її організацію, яка має забезпечити цілісність її даних, доступ до них і виконання над ними відповідних дій.

ООП – ефективний спосіб організації програми. ООП ніяк не пов'язане з процесом виконання програми, а є тільки способом її ефективної організації.

1.3. Основні компоненти об'єктно-орієнтованої мови програмування

Розглянемо декілька основних компонент, що входять до складу будь-якої об'єктно-орієнтованої мови програмування, у тому числі і мови C++: це об'єкти, класи, успадкування, повторне використання коду програми, типи даних користувача, поліморфізм, перевизначення операторів тощо.

Поділ програми на об'єкти. Якщо доведеться розв'язувати деяку прикладну задачу з використанням об'єктно-орієнтованого підходу, то замість проблеми поділу програми на функції наштовхнетеся на проблему поділу її на об'єкти. Згодом зрозумієте, що мислення в термінах об'єктів виявляється набагато простішим і більш наочним, ніж у термінах функцій, оскільки програмні об'єкти схожі з фізичними об'єктами реального світу.

Тут дамо відповідь тільки на таке запитання: що у коді програми треба подавати у вигляді об'єктів? Остаточну відповідь на це запитання може дати тільки власний досвід програмування з використанням об'єктно-орієнтованого підходу, а також досконале знання фізичної сутності розв'язуваної задачі. Проте нижче наведено декілька прикладів, які можуть виявитися корисними як для початківців-студентів, так і для програмістів-нефахівців з ООП:

- фізичні об'єкти: верстати та устаткування під час моделювання перебігу технологічного процесу виготовлення продукції; транспортні засоби під час моделювання процесу переміщення продукції; схемні елементи під час моделювання роботи ланцюга електричного струму; виробничі

підприємства під час розроблення економічної моделі; літальні апарати під час моделювання диспетчерської системи тощо;

- елементи інтерфейсу: вікна програми; меню користувача; графічні об'єкти (лінії, прямокутники, круги); мишка, клавіатура, дискові пристрої, принтери, плоттери тощо;

- структури даних: звичайні та розріджені масиви; стеки; одно- і двозв'язні списки; бінарні дерева тощо;

- групи людей: співробітники; студенти; покупці; продавці тощо;

- сховища даних: описи обладнання та устаткування; інформація про виготовлену продукцію; списки співробітників; словники; географічні координати точок тощо;

- типи даних користувача: час; довжини; грошові одиниці; величини кутів; комплексні числа; точки на площині чи у просторі;

- учасники комп'ютерних ігор: автомобілі на перегонах; позиції в настільних іграх (шашки, шахи); тварини в іграх, пов'язані з живою природою; друзі та вороги в пригодницьких іграх.

Відповідність між програмними і реальними об'єктами є наслідком об'єднання даних і відповідних їм функцій. Об'єкти, які одержано внаслідок такого об'єднання, свого часу викликали фурор, адже жодна програмна модель, розроблена на основі структурного підходу, не відображала наявні речі так точно, як це вдалося зробити за допомогою об'єктів.

Визначення класу. Коли йдеться про об'єкти, то вважається, що вони є екземплярами класів. Що це означає? Розглянемо таку тривіальну аналогію.

Практично всі комп'ютерні мови мають стандартні типи даних; наприклад, у мові програмування C++ є цілий тип `int`. Ми можемо визначати змінні таких типів у наших програмах:

```
int day, count, divisor, answer;
```

За аналогією можемо визначати об'єкти класу. Тобто, клас – це тип форми, що визначає, які дані та функції будуть включені в об'єкт. Під час оголошення класу не створюються ніякі об'єкти цього класу, за аналогією з тим, що існування типу `int` ще не означає наявності змінних цього типу.

Таким чином, визначальним для класу є тип сукупності об'єктів, схожих між собою. Це відповідає нестрогому в технічному сенсі розумінню терміну "клас". Не існує конкретної людини з іменем "рок-музикант", проте люди зі своїми унікальними іменами є об'єктами цього класу, якщо вони володіють певним набором характеристик. Об'єкт класу часто також називають екземпляром класу.

Поняття про успадкування в класах. Поняття класу тісно пов'язане з поняттям успадкування в класах. У повсякденному житті часто маємо справу з поділом класів на підкласи: наприклад, клас тварин можна розбити на підкласи ссавці, земноводні, комахи, птахи і т.д. Клас наземний транспорт ділиться на класи автомобілі, вантажівки, автобуси, мотоцикли, автокрани і т.д.

Принцип, закладений в основу такого поділу, полягає в тому, що кожен підклас має властивості, притаманні тому класу, з якого виділений даний

підклас. Автомобілі, вантажівки, автобуси і мотоцикли мають колеса і двигун, який є характеристиками наземного транспорту. Окрім тих властивостей, які є загальними у даних класу і підкласу, підклас може мати і власні: наприклад, автобуси мають велику кількість посадкових місць для пасажирів, тоді як вантажівки мають значний простір і потужність двигуна для перевезення вантажів, негабариту, в т.ч. й інших машин.

Подібно до наведеного вище класу наземного транспорту, у програмуванні клас також може породити множину підкласів. У мові програмування C++ клас, який породжує всю решту класів, називається базовим класом; решта класів, що успадковує його властивості, водночас має власні характеристики. Такі класи називають похідними.

Не проводьте помилкових аналогій між відносинами "об'єкт-клас" і "базовий клас – похідний клас". Об'єкти, які існують в пам'яті комп'ютера, є втіленням властивостей, притаманних класу, до якого вони належать. Похідні класи мають властивості як успадковані від базового класу, так і свої власні.

Успадкування можна вважати аналогом використання функцій у структурному підході. Якщо виявимо декілька функцій, які здійснюють схожі дії, то вилучимо з них однакові частини і винесемо їх в окрему функцію. Тоді початкові функції будуть однаковою мірою викликати свою загальну частину і водночас у кожній із них міститимуться свої власні настанови. Базовий клас містить елементи, які є загальними для групи похідних класів. Значення успадкування в ООП таке ж саме, як і функцій у структурному програмуванні, – скоротити розмір коду програми і спростити зв'язки між її елементами.

Розроблений раніше клас часто можна використовувати в інших програмах. Ця властивість називається повторним використанням коду. Аналогічну властивість в структурному програмуванні мають бібліотеки функцій, які можна вносити в різні програмні проекти.

В ООП концепція успадкування відкриває нові можливості повторного використання коду програми. Програміст може взяти наявний клас, і, нічого не змінюючи в ньому, додати до нього свої елементи. Всі похідні класи успадкують ці зміни і, водночас, кожний із похідних класів також можна окремо модифікувати.

Припустимо, що розробили клас, який подає систему меню, аналогічну графічному інтерфейсу Microsoft Windows або іншому графічному інтерфейсу користувача (GUI). Не має бажання змінювати цей клас, але необхідно мати можливість встановлювати та знімати опції. У цьому випадку створюється новий клас, який успадковує всі властивості початкового класу, і додається до нього необхідний код програми.

Зручність повторного використання кодів уже написаних програм є важливою перевагою технології ООП над іншими технологіями. Чимало комп'ютерних компаній стверджують, що можливість вносити в нові версії програмного забезпечення старі коди програм сприяє зростанню їх прибутків, які вони приносять.

Однією з переваг об'єктів є те, що вони дають змогу програмісту створювати свої власні типи даних.

Нехай необхідно працювати з об'єктами, які мають дві координати, наприклад x і y . Вам хотілося б здійснювати звичайні арифметичні операції над такими об'єктами, наприклад:

$$Ob1 = Ob2 + obj;$$

де змінні $Ob1$, $Ob2$ і obj є наборами з двох координат.

Описавши клас, який містить пару координат, і оголосивши об'єкти цього класу з іменами $Ob1$, $Ob2$ і obj , фактично створимо новий тип даних. У мові програмування C++ є засоби, які полегшують створення подібних типів даних користувача.

Варто звернути увагу на те, що операції присвоєння ($=$) і додавання ($+$) для типу `Coordinate` мають виконувати дії, які відрізняються від тих, які вони виконують для об'єктів стандартних типів, наприклад `int`. Об'єкти $Ob1$ та інші не є стандартними, оскільки визначені користувачем як такі, що належать до класу `Coordinate`. Як же оператори $-$ і $+$ розпізнають, які дії необхідно зробити над операндами?

Відповідь на це запитання полягає в тому, що програміст сам може задати ці дії, зробивши потрібні оператори методами класу `Coordinate`. Використання окремо операцій і функцій залежно від того, з якими типами величин їм доводиться у даний момент працювати, називається поліморфізмом. Коли наявна операція, наприклад, $-$ або $+$, наділяється можливістю здійснювати дії над операндами нового типу, то вважається, що така операція є перевизначеною. Перевизначення є окремим випадком поліморфізму і є важливим інструментом ООП.

Література

1. Объектно-ориентированный анализ и проектирование с примерами приложений / [Гради Буч, Роберт Максимчук, Майкл Энгл и др.] ; пер. с англ. – М. : ООО "ИД Вильямс", 2008. – 720 с.
2. Лабор В. В. Си Шарп: Создание приложений для Windows. — Мн.: Харвест, 2003. — 384 с.
3. Троелсен Э. Язык программирования C# 2010 и платформа.NET4.0 / Эндрю Троелсен ; пер. с англ. – М. : ООО "ИД Вильямс", 2011 – 1392 с.
4. Фомин Г.В. Введение в программирование на C# в среде MS Visual Studio. : ЮФУ, 2011.– 181с.
5. Нэш Т. C# 2010: ускоренный курс для профессионалов / Трей Нэш ; пер. с англ. – М. : ООО "ИД Вильямс", 2010 – 592 с.
6. Фридл Дж. Регулярные выражения / Джон Фридл ; пер. с англ. – СПб. : Питер, 2003 – 464 с.
7. Шилдт Г. C# 3.0: руководство для начинающих / Герберт Шилдт ; пер. с англ. – М. : ИД "Вильямс", 2009 – 688 с.
8. Леоненков А. Самоучитель UML 2 / Александр Леоненков. – СПб. : BHV, 2007 – 576 с.

9. Маки А. Введение в .NET 4.0 и Visual Studio 2010 для профессионалов / Алекс Маки ; пер. с англ. – М. : ООО ИД "Вильямс", 2010. – 416 с.
10. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1 / Пер. с англ. — М.: Издательско-торговый дом Русская Редакция, 2002. — 576 с.
11. Программирование для Microsoft Windows на C#. В 2-х томах. Том 2 / Пер. с англ. — М.: Издательско-торговый дом Русская Редакция, 2002. — 624 с.