Лабораторна робота №2

ЛАБОРАТОРНА РОБОТА 2: ФОРМАЛЬНІ ТА ВІЗУАЛЬНІ МЕТОДИ КОНСТРУЮВАННЯ КОМПОНЕНТІВ.

Мета роботи: Ознайомитись із формальними та візуальними методами конструювання компонентів.

2.1 Теоретичні відомості

Для створення конкретної фізичної системи необхідно деяким чином реалізувати всі елементи логічного подання у конкретні матеріальні сутності. Насамперед необхідно розробити вихідний текст програми на деякій мові програмування. При цьому уже в тексті програми передбачається така організація програмного коду, що припускає його розбивку на окремі модулі.

Проте, вихідні тексти програми ще не є остаточною реалізацією проекту, хоча й служать фрагментом його фізичного подання. Очевидно, програмна система може вважатися реалізованою в тому випадку, коли вона буде здатна виконувати функції свого цільового призначення. Для цього необхідно, щоб програмний код системи був реалізований у формі виконуваних модулів, бібліотек класів і процедур, стандартних графічних інтерфейсів, файлів баз даних. Саме ці компоненти є необхідними елементами фізичного подання.

У мові UML для фізичного подання моделей систем використовуються діаграми реалізації, які включають дві окремі канонічні діаграми: діаграму компонентів і діаграму розгортання. Діаграма розгортання описує платформу й обчислювальні засоби для роботи системи.

2.1.1 Діаграма компонентів

Діаграма компонентів розробляється для наступних цілей:

1. візуалізації загальної структури вихідного коду програмної системи;
2. специфікації варіанта виконуваної програмної системи;
3. забезпечення багаторазового використання окремих фрагментів програмного коду;
4. подання концептуальної й фізичної схеми баз даних.

У розробці діаграми компонентів беруть участь системні аналітики, архітектори системи й програмісти. Одні компоненти існують на етапі компіляції програмного коду (вихідні модулі), інші на етапі його виконання. У бізнес-системах діаграми включають відділи, служби й документи, які реально існують у системі. Фізична сутність в UML називається компонентом. Компонент реалізує деякий набір інтерфейсів. На діаграмі він позначається таким способом (рисунок 2.1):

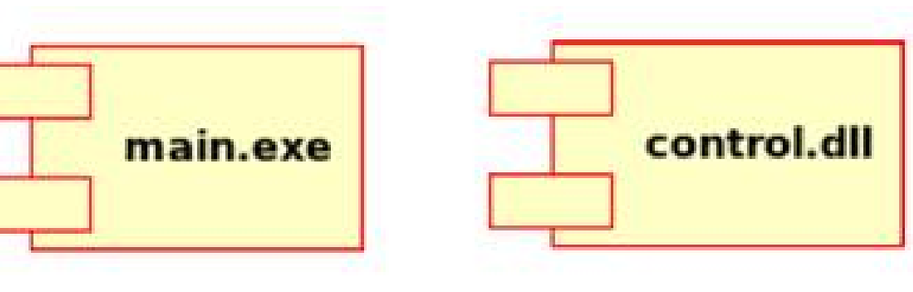


Рис. 2.1 Представлення компонентів

Окремий компонент може бути представлений на рівні типу або на рівні компілятора. Їхні імена при цьому розрізняються. У першому випадку записується тільки ім'я типу із заголовної букви. У другому випадку записується ім'я компонента, а потім через двокрапку - ім'я типу. Весь рядок імені в цьому випадку підкреслюється. Ім'я компонента й ім'я типу може складатися з будь-якого числа букв, цифр і деяких розділових знаків. Іноді для імені екземпляра використовують іншу нотацію: ім'я компонента починається з малої літери, а підкреслення імені не використовується. У якості простих імен компонентів використовують імена файлів: *.exe, *.dll, *.htm, *.txt, *.doc, *.hlp, *.db, *.mdb, *.h, *.cpp, *.java, *.class, *.pl, *.asp і т.д.

Оскільки компонент, як елемент фізичної реалізації моделі представляє окремий модуль коду, іноді його коментують із вказівкою додаткових графічних символів, що ілюструють конкретні особливості реалізації: для *.dll - два зубчасті колеса на листі, для *.htm – зафарбоване коло, для *.hlp - рядок тексту із крапками переходу у вигляді зафарбованих прямокутників на листі, для *.cpp - рядок тексту і т.д. Ці додаткові позначення не специфіковані в мові UML. У мові UML виділяють тільки три види компонентів:

1. компоненти розгортання, які забезпечують безпосереднє виконання системою своїх функцій: *.dll, *.htm, *.hlp;
2. компоненти - робочі продукти, як правило - це файли з вихідними текстами програм: *.h, *.cpp і т.і.;
3. компоненти виконання - файли *.exe.

Ці компоненти називають артефактами розробки. Інший спосіб специфікації різних видів компонентів - явна вказівка стереотипу компонента перед його ім'ям:

- - бібліотека (library) - для динамічної або статичної бібліотеки;
- - таблиця (table) -для таблиці бази даних;
- - файл (file) - для файлів з вихідними текстами програм;
- - документ (document) - для документів;
- - виконуваний (executable) - для компонентів, які можуть виконуватися у вузлі.

Компонентами також є інтерфейси, які зображаються кружечком, що з'єднують із програмними компонентами відрізками ліній без стрілок. При цьому ім'я інтерфейсу повинне починатися з букви “I” й записується поруч із окружністю. Лінія означає, що програмний компонент реалізує цей інтерфейс.

Іншим способом подання інтерфейсу є його зображення у вигляді прямокутника класу зі стереотипом Interface й можливими секціями атрибутів й операцій. Якщо компонент реалізує деякий інтерфейс, то такий інтерфейс називається експортованим. Використовуваний інтерфейс іншого модуля називається імпортованим й зображується пунктирною лінією зі стрілкою, спрямованою до інтерфейсу.

Загальні рекомендації по побудові діаграми компонентів:

1. максимально враховувати інформацію при поданні моделі системи;
2. точно знати обчислювальні платформи й операційні системи, на яких буде реалізовуватися система;
3. добре представляти можливості обраної мови програмування;
4. визначитися з вибором бази даних або знань;
5. при розподілі програмної системи на фізичні модулі або файли варто домагатися такої реалізації, що забезпечила би можливість повторного використання коду й створення об'єктів тільки при їхній необхідності. Для цієї мети необхідно більшу частину описів класів, їхніх операцій і методів винести в DDL, залишивши у виконуваних компонентах тільки самі необхідні для ініціалізації програми фрагменти виконуваного коду;
6. після загальної структуризації фізичного подання системи необхідно доповнити її інтерфейсами й схемами бази даних, звертаючи особливу увагу на узгодження різних частин програмної системи, специфікацію таблиць і встановлення зв'язків між таблицями;
7. завершувати побудову діаграми компонентів необхідно встановленням і нанесенням взаємозв'язків між компонентами, в тім числі й відносин реалізації, використовуючи при цьому різні види графічного зображення компонентів. Отримана діаграма повинна ілюструвати всі найважливіші аспекти фізичної реалізації системи, починаючи з особливостей компіляції вихідних текстів програм і закінчуючи виконанням окремих частин програми на етапі її виконання;
8. при розробці діаграми компонентів необхідно використати уже наявні в мові UML компоненти й стереотипи, прибігаючи лише у виняткових випадках до механізмів розширення: додаткові стереотипи й позначені значення.

Приклади діаграми компонентів показано на рис 2.2 та 2.3.

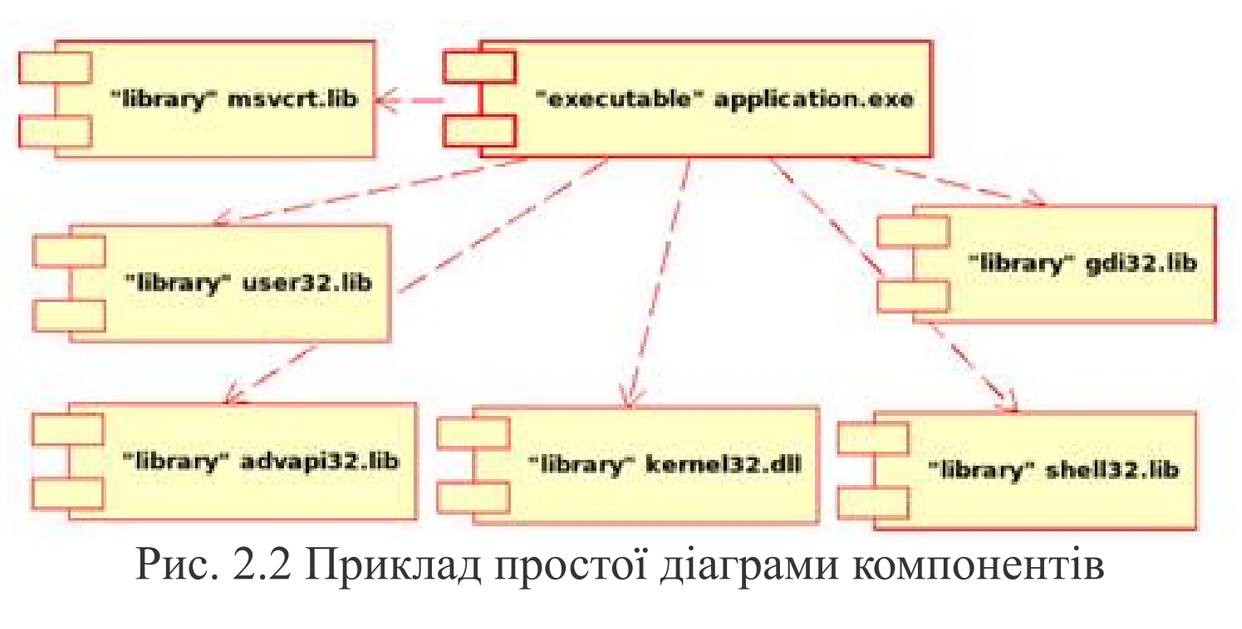


Рис. 2.2 Приклад простої діаграми компонентів

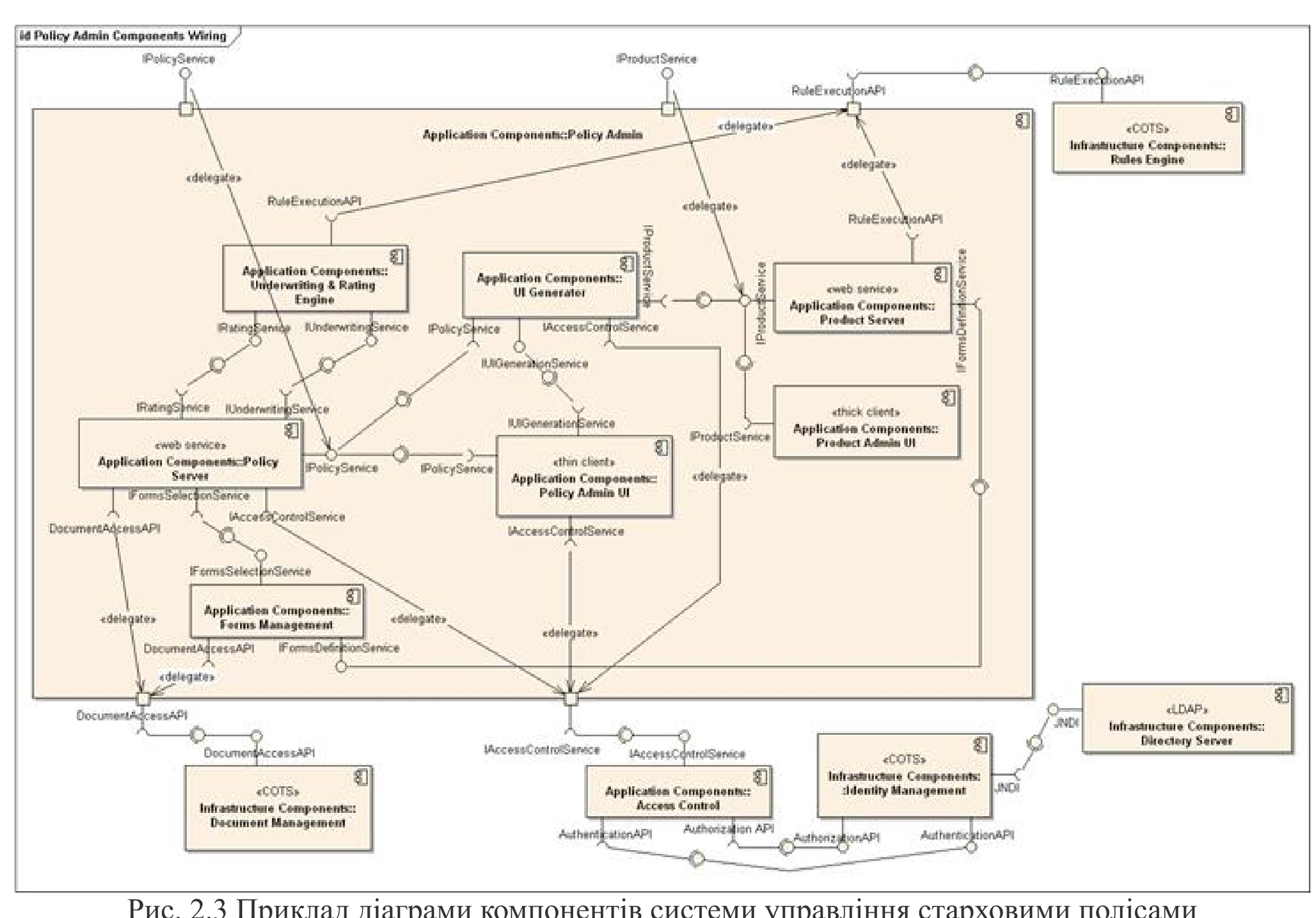


Рис. 2.3 Приклад діаграми компонентів системи управління старховими полісами

2.1.2 Даграма класів

Діагрáма клáсів – це статичне представлення структури програми в термінології об'єктно-орієнтованого програмування. Вона відображає статичні (декларативні) елементи, такі як: класи, тип даних, їх зміст та відношення між ними. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Графічно клас зображується у вигляді прямокутника із вказівкою імені класу й списку атрибутів, а також операцій. Атрибути й операції утворюють секції й відокремлюються один від одного й від імені горизонтальними лініями. При проектуванні в прямокутнику спочатку записується ім'я класу, а атрибути й операції записуються по мірі пророблення програмної системи. Іноді використовується й четверта секція для вказівки виняткових ситуацій або визначення семантики класу.

Ім'я класу повинно відображати призначення класу й записуватися на англійській або російській/українській мові з великої літери. Ім'я абстрактного класу записується курсивом. Кожен атрибут класу має певну область видимості:

+ загальнодоступний (public);
# захищений (protected);
- закритий (private).

Відношення між елементами програми:

Наслідування (успадкування)

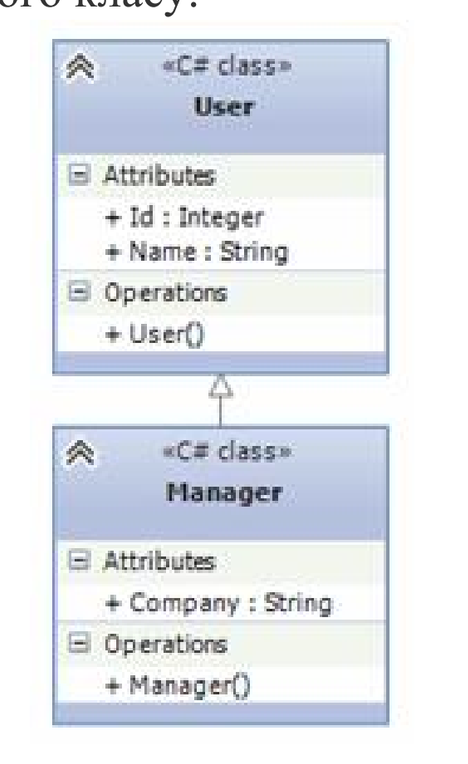
Наслідування є базовим принципом ООП і дозволяє одному класу (спадкоємцю) успадкувати функціонал іншого класу (батьківського). Наслідування визначає відношення IS A, тобто "є". наприклад:

```
class User
{
    public int Id { get; set; }
    public string Name { get; set; }
}

class Manager : User
{
    public string Company{ get; set; }
}
```

В даному випадку клас Manager наслідує клас User, а об'єкти класу Manager також є і об'єктами класу User.

За допомогою діаграм UML відношення наслідування між класами зображується незафарбованою стрілочкою від класу-спадкоємця до батьківського класу:

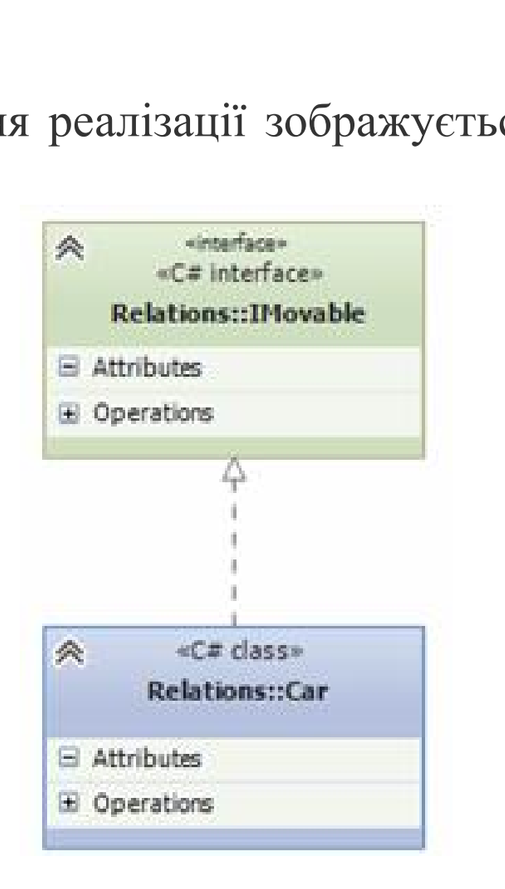


Реалізація

Реалізація передбачає визначення інтерфейсу і його реалізація в класах. Наприклад, є інтерфейс IMovable з методом Move, який реалізується в класі Car:

```
public interface IMovable
{
    void Move();
}
public class Car : IMovable
{
    public void Move()
    {
        Console.WriteLine("Машина їде");
    }
}
```

За допомогою діаграм UML відношення реалізації зображується пунктирною незафарбованою стрілочкою від класу до інтерфейсу.

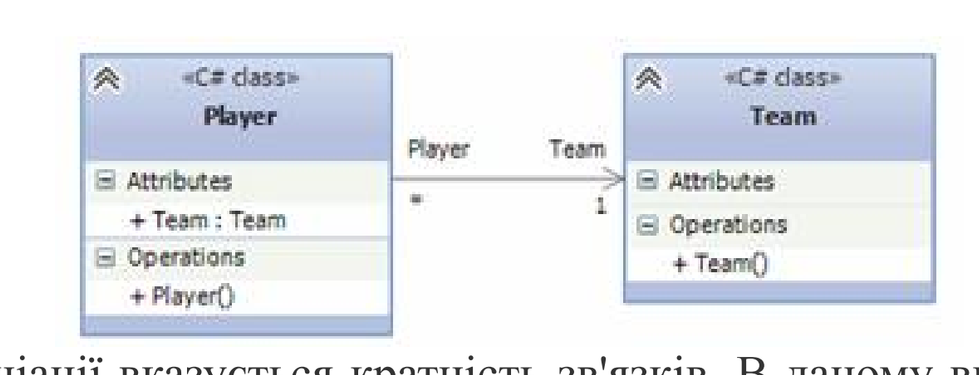


Асоціація

Асоціація - це відношення, при якому об'єкти одного типу якимось чином пов'язані з об'єктами іншого типу. Наприклад, об'єкт одного типу містить або використовує об'єкт іншого типу. Наприклад, гравець грає в певній команді:

```
class Team
{
}
class Player
{
    public Team Team { get; set; }
}
```

Клас Player пов'язаний відношенням асоціації з класом Team. На схемах UML асоціація позначається у вигляді стрілки:



Часто при відношенні асоціації вказується кратність зв'язків. В даному випадку одиниця у Team і зірочка у Player на діаграмі відображає зв'язок 1 до багатьох. Тобто одна команда буде відповідати багатьом гравцям.

Агрегація і композиція є окремими випадками асоціації.

Композиція

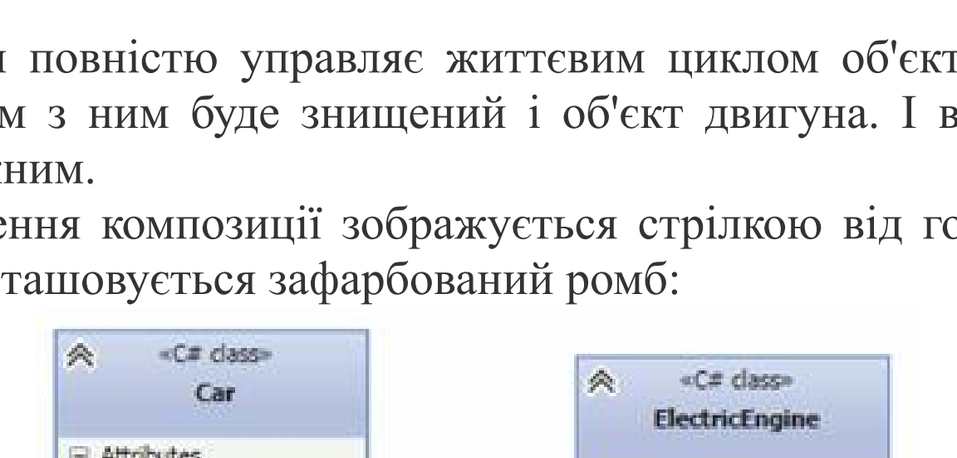
Композиція визначає відношення HAS A, "має". Наприклад, клас автомобіля містить об'єкт класу електричного двигуна:

```
public class ElectricEngine
{ }

public class Car
{
    ElectricEngine engine;
    public Car()
    {
        engine = new ElectricEngine();
    }
}
```

При цьому клас автомобіля повністю управляє життєвим циклом об'єкта двигуна. При знищенні об'єкта автомобіля в області пам'яті разом з ним буде знищений і об'єкт двигуна. І в цьому плані об'єкт автомобіля є головним, а об'єкт двигуна – залежним.

На діаграмах UML відношення композиції зображується стрілкою від головної сутності до залежної, при цьому з боку головної сутності розташовується зафарбований ромб:



Агрегація

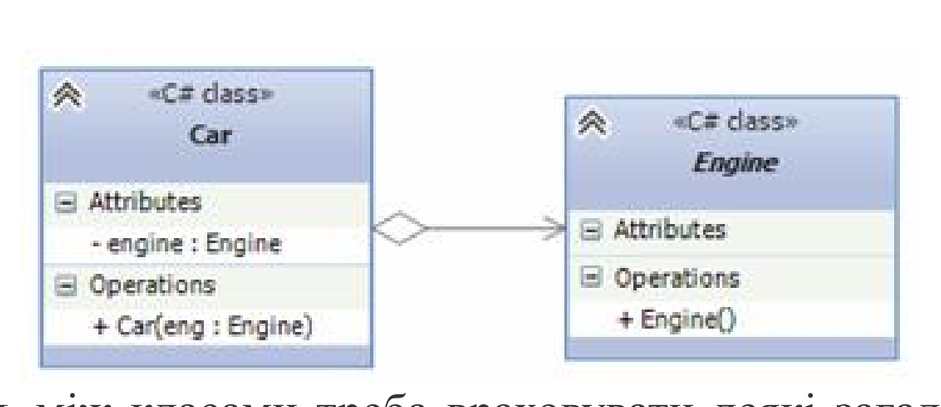
Агрегація також представляє відношення HAS A, але реалізується вона дещо по іншому:

```
public abstract class Engine
{ }

public class Car
{
    Engine engine;
    public Car(Engine eng)
    {
        engine = eng;
    }
}
```

При агрегації реалізується слабкий зв'язок, тобто в даному випадку об'єкти Car і Engine будуть рівноправними. У конструктор Car передається посилання на вже наявний об'єкт Engine. І, як правило, визначається посилання не на конкретний клас, а на абстрактний клас або інтерфейс, що збільшує гнучкість програми.

Представлення агрегації на діаграмах UML таке ж, як і представлення композиції, тільки тепер ромб не замальований:



При проектуванні відношень між класами треба враховувати деякі загальні рекомендації. Зокрема, замість наслідування слід надавати перевагу композиції. При наслідування весь функціонал класу-спадкоємця жорстко визначено на етапі компіляції. І під час виконання програми ми не можемо його динамічно перевизначити. А клас-спадкоємець не завжди може перевизначити код, який визначений в батьківському класі. Композиція ж дозволяє динамічно визначати поведінку об'єкта під час виконання, і тому є більш гнучкою.

Замість композиції слід надавати перевагу агрегації, як більш гнучкому способу зв'язку компонентів. У той же час не завжди агрегація доречна. Наприклад, нехай у нас є клас людини, який містить об'єкт нервової системи. Зрозуміло, що в реальності, принаймні на поточний момент, неможливо окремо створити нервову систему і потім помістити її в людину. Тобто в даному випадку людина буде головним компонентом, а нервова система – залежним (підлеглим). Їх створення і життєвий цикл будуть відбуватися спільно, тому тут краще вибрати композицію.

2.2 Завдання

Розробити діаграму класів для завдання, виконаного в лабораторній роботі №1. Для виконання завдання рекомендовано використовувати інструментальне середовище VisualStudio будь-якої версії.

2.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

2.4 Контрольні запитання

Що таке UML?
Що таке діаграма компонентів, як вона будується і для чого вона призначена?
Що таке діаграма класів, як вона будується і для чого вона призначена?
Основні відношення між елементами програми: наслідування, реалізація, асоціація, композиція, агрегація. Як вони позначаються на діаграмі класів?

Рекомендовані посилання:

https://ru.wikipedia.org/wiki/%%D0%%94%%D0%%B8%%D0%%B0%%D0%%B3%%D1%%80%%D0%%B0%%D0%

https://ru.wikipedia.org/wiki/%%D0%%94%%D0%%B8%%D0%%B0%%D0%%B3%%D1%%80%%D0%%B0%%D0%

https://www.youtube.com/watch?v=MXkBUgs7z9g

http://posibnyky.vntu.edu.ua/bevz/52.htm

https://www.youtube.com/watch?v=UI6lqHOVHic

https://www.youtube.com/watch?v=KQUGFFN4M90

https://docs.kde.org/trunk4/uk/kdesdk/umbrello/uml-elements.html#class-diagram

Востаннє редаговано: Середа, 23 травня 2018, 15:58. Версія: 4. Опубліковано: Середа, 14 лютого 2018, 12:09.

Навігація по матеріалу
Домашня курсу
Загальні відомості про курс
Модуль 1
Теоретичний навчальний ма...
Практичні, семінарські і...
Завдання для самостійної...
Модульний контроль
Модуль 2
Модуль 3
Підсумкова атестація
Ректорська контрольна робота (гру..
Список питань на РКК
Ректорський контроль (проба)..
Лекція 1. Вступ до крос-платформн..
Лекція 2. Визначення та властивос..
Лекція 3. Стратегії інтеграції пр..
Лекція 4. Основні види архітектур..
Лекція 5. Технологічний стандарт..
Лекція 6. Технологія COM
Лекція 7. Основи ООП
Лекція 8. Розподілені бази даних..
Лекція 9. ORM-технологія
Лекція 10. Виклик віддалених проц..
Лекція 11. Компонентні технології..
Лекція 12. Технологія Entity...
Лабораторна робота №1
Теоретичні відомост
Завдання до лаб. №1
Лабораторна робота №2
Додаток до ЛР №2
Лабораторна робота №3
Лабораторна робота №4
Лабораторна робота №5
Лабораторна робота №6

Про курс

Призначено для груп:
СН 3 курс, СНз 3 курс

Спеціальності:
6.050101 Комп'ютерні науки
122 Комп'ютерні науки та інформаційні технології (бакалавр)

Лектор:
Mykhailovych Taras V
Останній візит: 15.06.2017 14:11
Написати повідомлення

Козак Руслан Орестович
Останній візит: 27.03.2018 12:26
Написати повідомлення

Готович Володимир Анатолійович
Останній візит: 01.06.2018 17:33
Написати повідомлення

Василенко Ярослав Пилипович
Останній візит: 27.03.2018 09:04
Написати повідомлення

Активні користувачі
Тхір Ігор Ігорович
Гості не показані

Веб конференції та вебінари
Нічого не знайдено.

Пошук
Шукати:
всі слова
будь-яке слово
Пошук