**Лекція 5. Засоби попереднього моделювання (UML моделювання)**

**Загальні відомості**

**Статичні та динамічні моделі**

В *UML* *інтегровані* різноманітні відомі засоби візуального моделювання, які добре зарекомендували себе на практиці, зокрема, забезпечується можливість опису двох визначальних видів об'єктних моделей:

* + *структурних* (або *статичних*) *моделей* – описується структура сутностей системи, включаючи класи, інтерфейси, відношення, атрибути;
  + моделей *поведінки* (або *динамічних моделей*) – описується поведінка (функціонування) об'єктів системи, включаючи методи, взаємодію, процес зміни станів окремих компонент чи всієї системи.

**Призначення *UML***

* + Надати користувачу *засоби візуального моделювання* систем різного призначення з акцентацією на можливості їх розробки та отримання документації. (*UML* містить як абстрактні конструкції для представлення моделей, так і цілком конкретні, які дозволяють описувати деталі реалізації програмних систем).
  + Забезпечити користувачів *засобами розширення та специфікації* з метою більш точного опису конкретних предметних областей. (Хоча у більшості випадків для побудови моделей цілком достатньо базових конструкцій *UML,* все ж в *UML* уведено механізм *розширення* базових понять. Крім того, можлива *спеціалізація* базових понять, шляхом доповнення останніх новими атрибутами чи властивостями).
  + Підтримувати таку специфікацію моделей, яка, з одного боку, була б *незалежною* від конкретних мов програмування і, з іншого боку, забезпечувала б потенційні *можливості реалізації* у таких мовах.

**Діаграми *UML***

Графічні зображення моделей системи в UML називаються ***діаграмами***.

Діаграма *UML* – це зображення у вигляді *графа з вершинами (сутностями) і ребрами (відношеннями)*.

Основна *мета* діаграм – *візуалізація архітектури* розроблюваної системи з різних точок зору.

*Діаграма* – *деякий зріз системи.* Звичайно діаграми дають згорнуте представлення елементів, із яких складається розроблювана система. При цьому один і той самий елемент може бути присутнім у декількох (а іноді й в усіх) діаграмах.

При візуальному моделюванні з *UML* використовуються *вісім видів діаграм*, кожна з яких може містити елементи певного типу. Типи можливих елементів і відношень між ними залежать від виду діаграми.

Конструкції UML створюються з багатьох модельних елементів, які позначають різні частини системи програмного забезпечення. Елементи UML використовуються для побудови діаграм, які відповідають певній частині системи або точці зору на систему. У Umbrello UML Modeller реалізовано підтримку таких типів діаграм:

* *Діаграма* *варіантів / випадків використання або прецедентів* (use case diagram) показує дієвих осіб (людей або інших користувачів системи), випадки використання (сценарії використання системи) та їх взаємодію (їх частково розглядали у ЛР№2). Такі діаграми описують ***функціональність***, яка буде надаватись користувачам системи, котра проектується. Представляються шляхом використання ***прецедентів*** та ***акторів***, а також ***відношень між ними***. Набір усіх прецедентів діаграми фактично визначає ***функціональні вимоги***, за допомогою яких може бути сформульоване ***технічне завдання***.
* *Діаграми класів (class diagram)*, на яких показано класи та зв’язки між ними. Ці діаграми описують статичну структуру класів. Дозволяють (на концептуальному рівні) формувати "словник предметної області" та (на рівні специфікацій і рівні реалізацій) визначати структуру класів у програмній реалізації системи. Можуть використовуватись для генерації каркасного програмного коду (в реальній мові програмування).

Для опису динаміки використовуються ***діаграми поведінки*** ***(behavior diagrams)***, що підрозділяються на

* + ***діаграми станів (statechart diagrams);***
  + ***діаграми діяльності (активності) (activity diagrams);***
  + ***діаграми взаємодії (interaction diagrams),*** що у свою чергу підрозділяються на
    - ***діаграм послідовності (sequence diagrams);***
    - ***діаграм кооперації (співробітництва) (collaboration diagrams).***
* *Діаграми стану (statechart diagram)*, на яких показано стани, зміну станів і події у об’єкті або частині системи.
* *Діаграми діяльності (activity diagram)*, на яких показано дії та зміни однієї дії іншою, які є наслідком подій, що сталися у певній частині системи.
* *Діаграми послідовності (sequence diagram)*, на яких показано об’єкти і послідовність методів, якими ці об’єкти викликають інші об’єкти.
* *Діаграми співпраці / кооперації (collaboration diagram)*, на яких показано об’єкти та їх взаємозв’язок з наголосом на об’єкти, які беруть участь у обміні повідомленнями

Ще одна група - ***діаграми реалізації (implementation diagrams)*** поділяються на

* *Діаграми компонентів (component diagram)*, на яких показано програмні компоненти високого рівня.
* *Діаграми впровадження / розгортання (deployment diagram)*, на яких показано екземпляри компонентів та їх взаємодію.

Окремо виділемо

* *Діаграми взаємозв’язку сутностей взаємодії (interaction diagrams)*, на яких показано дані, взаємозв’язки і умови обмеження зв’язків між даними.

Кожна з цих діаграм конкретизує різні уявлення про моделі системи. При цьому, діаграма варіантів використання представляє концептуальну модель системи, яка є вихідною для побудови всіх інших діаграм. Діаграма класів є логічною моделлю, що відбиває статичні аспекти структурної побудови системи, а діаграми поведінки, які також є різновидами логічної моделі, відображають динамічні аспекти її функціонування. Діаграми реалізації слугують для подання компонентів системи і відносяться до її фізичної моделі.

З перерахованих вище діаграм деякі служать для позначення двох і більше підвидів. У якості ж самостійних уявлень використовуються наступні діаграми: варіантів використання, класів, станів, діяльності, послідовності, кооперації, компонентів і розгортання .

Для діаграм мови UML існують три типи візуальних позначень, які є важливими з точки зору укладеної в них інформації:

* зв'язку , Які надаються різними лініями на площині;
* текст , Що міститься всередині меж окремих геометричних фігур;
* графічні символи , Зображувані поблизу візуальних елементів діаграм.

При графічному зображенні діаграм рекомендується дотримуватися наступних правил:

* кожна діаграма повинна бути закінченим поданням деякого фрагмента модельованої предметної області;
* представлені на діаграмі сутності моделі повинні бути одного концептуального рівня;
* вся інформація про сутності повинна бути явно представлена на діаграмі;
* діаграми не повинні містити суперечливої інформації;
* діаграми не слід перевантажувати текстовою інформацією;
* кожна діаграма повинна бути самодостатньою для правильної інтерпретації всіх її елементів;
* кількість типів діаграм, необхідних для опису конкретної системи, не є строго фіксованим і визначається розробником;
* моделі системи повинні містити тільки ті елементи, які визначено.

**Спрощена стратегія використання *UML*-діаграм при моделюванні програмних систем**

Спочатку для програмної системи серією *діаграм прецедентів* визначається її *зовнішня функціональність* (виділяються всі прецеденти та актори, а також відношення між ними).

Уся подальша робота над проектом має здійснюватись на основі прецедентів: для кожного прецеденту формується опис його динаміки у вигляді серії *діаграм взаємодії* та *діаграм діяльності*.

З отриманих описів шляхом виявлення об'єктів, що задіяні в реалізації прецедентів, будуються *діаграми класів*.

Для визначення поведінки класів із складною динамікою реагування на події можуть формуватись *діаграми станів*.

Розміщення об'єктів по програмних модулях описується в *компонентних діаграмах*, а розміщення програмних модулів по вузлах комп'ютерам мережі – у *діаграмах розгортання*.

Кожна діаграма має бути *завершеним* представленням фрагмента предметної області, який моделюється, тобто має враховувати всі істотні сутності обраного рівня абстрагування (*повнота моделі*).

Усі сутності діаграми мають бути *одного концептуального рівня* (відповідно до обраного рівня абстрагування).

Діаграми не повинні містити *суперечливу інформацію* (наприклад, не повинно бути елементів з однаковими іменами, але з різними атрибутами – омонімія, так само не повинно бути циклів у графах, що представляють відношення узагальнення чи агрегування).

Діаграми, в сукупності, мають бути *самодостатніми*, тобто не повинні вимагати додаткового текстового пояснення стосовно семантики чи інтерпретації окремих елементів діаграм. (Важлива роль при цьому покладається на побудову *ієрархії моделей*).

Не обов'язково у проектуванні використовувати *всі види* діаграм (наприклад, для локальних програмних систем не доцільно розробляти діаграму розгортання).

**Діаграма варіантів / випадків використання або прецедентів**

Розробка діаграми випадку використання призначена для:

* + Визначення загальних меж та контексту домену (предметної області), що моделюється, на початкових етапах проектування системи.
  + Формулювання загальних вимог до функціональної поведінки системи.
  + Розробки початкової концептуальної моделі системи для її подальшої деталізації у вигляді логічної та фізичної моделей.
  + Підготовки вихідної документації для взаємодії розробників системи зі своїми замовниками та користувачами.

Суть цієї діаграми полягає в тому, що проектована система представлена як сукупність сутностей або суб'єктів, що взаємодіють із системою, використовуючи так звані випадки використання. У цьому випадку Актор - це будь-яка сутність, яка взаємодіє із системою зовні. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка може служити джерелом впливу на модельовану систему, як визначено розробником. У свою чергу, випадок використання використовується для опису послуг, які система надає актору. Іншими словами, кожен випадок використання визначає певний набір дій, які виконує система під час діалогу з актором. У цьому випадку нічого не сказано про те, як буде реалізована взаємодія акторів із системою.

Основними елементами діаграми випадків використання є випадок використання та діюча особа - Актор.

На ЛР№2 було розглянуто випадки використання та акторів, розглянемо відношення між ними на діаграмі варіантів використання.

***Значення.*** Діаграми варіантів використання є *первісним,* *концептуальним* *представленням* (*концептуальною моделлю*) ПС в процесі її проектування й розробки. Вони виступають основою *подальшої деталізації* системи у формі логічних і фізичних моделей.

Розробка діаграм варіантів використання переслідує такі конкретні цілі:

* + визначити загальні *межі* та *контекст* системи;
  + сформулювати загальні *вимоги* до *функціональної поведінки* системи;
  + отримати *первісну* документацію для предметної взаємодії розроблювачів системи з її замовниками й користувачами.

**Діаграма класів**

Діаграмою класів у термінології UML називається діаграма, на якій зображений набір класів (і деяких інших сутностей, які не мають явного відношення до проектування БД при використанні UML в проектуванні БД), а також зв'язків між цими класами. Крім того, діаграма класів може включати коментарі та обмеження. Обмеження можуть неформально задаватися на природній мові або формулюватися мовою об'єктних обмежень OCL (Object Constraints Language).

На діаграмах класів показано різноманітні класи, які утворюють систему і їх взаємозв’язки. Діаграми класів називають “статичними діаграмами”, оскільки на них показано класи разом з методами і атрибутами, а також статичний взаємозв’язок між ними: те, яким класам “відомо” про існування яких класів, і те, які класи “є частиною” інших класів, — але не показано методи, які при цьому викликаються.

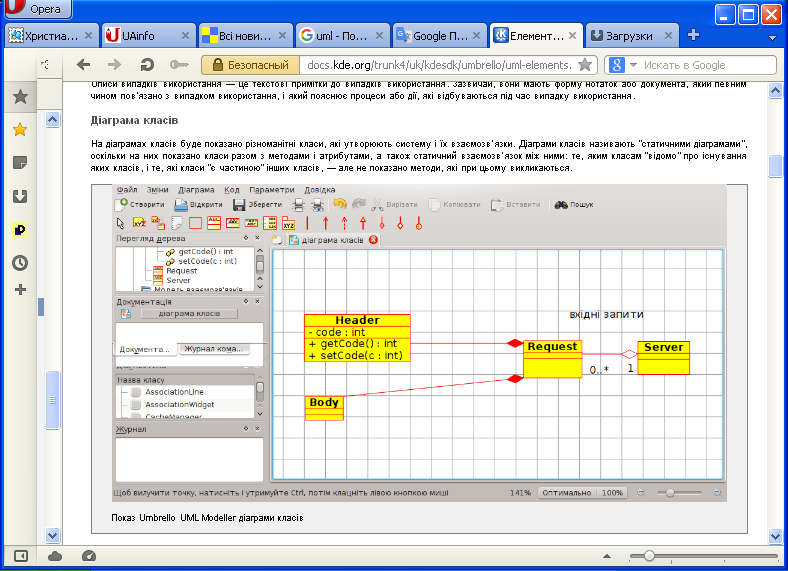


Рис.1 Показ Umbrello UML Modeller діаграми класів

***Клас***

Класом називається іменований опис сукупності об'єктів із загальними атрибутами, операціями, зв'язками і семантикою. Графічно клас зображується у вигляді прямокутника. У кожного класу має бути ім'я (текстовий рядок), що відрізняє його від всіх інших класів. При формуванні імен класів у UML допускається використання довільної комбінації літер, цифр і навіть розділових знаків. Однак на практиці рекомендується використовувати як імена класів короткі і осмислені прикметники та іменники, кожне з яких починається з великої літери.

Клас визначає атрибути і методи набору об’єктів. Всі об’єкти цього класу (примірники/екземпляри цього класу) мають спільну поведінку і однаковий набір атрибутів (кожен з об’єктів має свій власний набір значень). Іноді замість назви «клас» використовують назву “тип”, але, потрібно зауважити, що ці назви описують різні речі: тип є загальнішим визначенням.

У UML класи позначаються прямокутниками з назвою класу, у цих прямокутниках у вигляді двох “відсіків” може бути показано атрибути і операції класу.

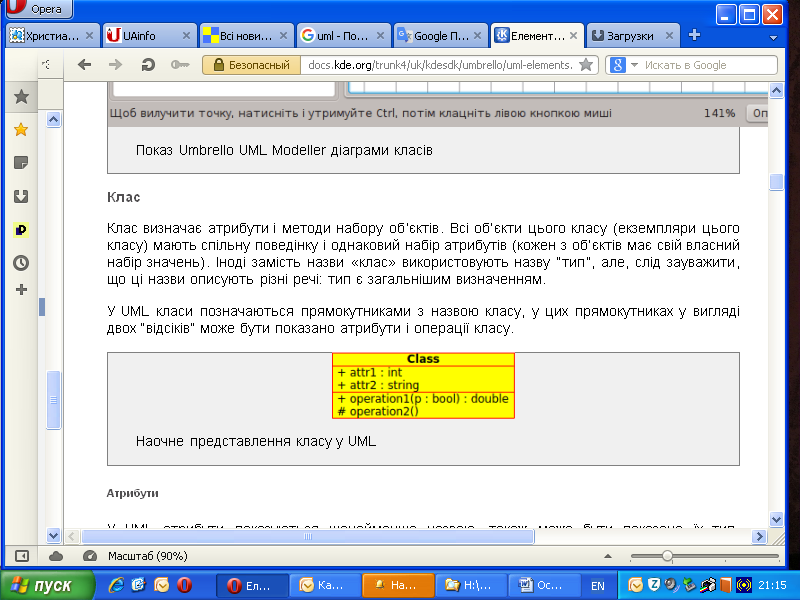


Рис.2 Наочне представлення класу у UML

***Атрибути***

Атрибутом класу називається іменовану властивість класу, що описує безліч значень, які можуть приймати екземпляри цієї властивості. Клас може мати будь-яке число атрибутів (зокрема, не мати жодного атрибута). Властивість, що виражається атрибутом, є властивістю модельованої сутності, загальною для всіх об'єктів даного класу. Таким чином, атрибут є абстракцією стану об'єкта. Будь-який атрибут будь-якого об'єкта класу повинен мати деяке значення.

Імена атрибутів представляються у розділі класу, розташованому під ім'ям класу. Хоча UML НЕ накладає обмежень на способи створення імен атрибутів (ім'я атрибута може бути довільною текстової рядком), на практиці рекомендується використовувати короткі прикметники та іменники, що відображають смисл відповідної властивості класу. Перше слово в назві рекомендується писати з великої літери, а всі інші слова - з прописної.

У UML атрибути показуються щонайменше назвою, також може бути показано їх тип, початкове значення і інші властивості. Крім того, атрибути може бути показано з областю видимості атрибута:

+ відповідає *публічним* (public) атрибутам

# відповідає *захищеним* (protected) атрибутам

- відповідає *приватним* (private) атрибутам

***Операції***

Операцією класу називається іменована послуга, яку можна запросити у будь-якого об'єкта цього класу. Операція - це абстракція того, що можна робити з об'єктом. Клас може містити будь-яке число операцій (зокрема, не містити жодної операції). Набір операцій класу є загальним для всіх об'єктів даного класу.

Операції класу визначаються у розділі, розташованому нижче розділу з атрибутами. При цьому можна обмежитися тільки зазначенням імен операцій, залишивши детальну специфікацію виконання операцій на більш пізні етапи моделювання. Для іменування операцій рекомендується використовувати дієслівні форми, відповідні очікуваній поведінці об'єктів даного класу. Опис операції може також містити її сигнатуру, тобто імена і типи всіх параметрів, а якщо операція є функцією, то і тип її значення.

Операції (методи) також показуються принаймні назвою, крім того, може бути показано їх параметри і типи значень, які буде повернуто. Операції, як і атрибути, може бути показано з областю видимості:

+ відповідає *публічним* (public) операціям

# відповідає *захищеним* (protected) операціям

- відповідає *приватним* (private) операціям

***Шаблони***

Серед класів можуть бути шаблони, значення, які використовуються для невизначеного класу або типу. Тип шаблону визначається під час ініціалізації класу (тобто, під час створення об’єкта). Шаблони існують у мові програмування C.

***Асоціації класів (Категорії зв'язків)***

Класи можна співвіднести (пов’язати) один з одним у декілька способів.

***Зв'язок-залежність (dependency)***

Залежністю називають зв'язок, коли зміна в специфікації одного класу може вплинути на поведінку іншого класу, що використовує перший клас. Найчастіше залежності застосовують в діаграмах класів, щоб відобразити в сигнатурі операції одного класу той факт, що параметром цієї операції можуть бути об'єкти іншого класу. Відповідно що, якщо інтерфейс другого класу змінюється, це впливає на поведінку об'єктів першого класу.

Залежність показується переривчастою лінією зі стрілкою, спрямованої до класу, від якого є залежність (див.Рис3).

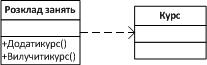


Рис.3 Приклад зв’язку залежності

***Узагальнення (generalization***)

Наслідування є однією з фундаментальних основ об’єктно-орієнтованого програмування, у якому клас “отримує” всі атрибути і операції класу, нащадком якого він є, і може перевизначати або змінювати деякі з них, а також додавати власні атрибути і операції.

Зв'язком-узагальненням називається зв'язок між загальною сутністю, названою суперкласом, або батьком, і більш спеціалізованої різновидом цієї сутності, названим підкласом, або нащадком. Узагальнення іноді називають зв'язками «is a», маючи на увазі, що клас-нащадок є окремим випадком класу-предка. Клас-нащадок успадковує всі атрибути і операції класу-предка, але в ньому можуть бути визначені додаткові атрибути та операції.

Об'єкти класу-нащадка можуть використовуватися скрізь, де можуть використовуватися об'єкти класу-предка. Це властивість називають поліморфізмом по включенню, маючи на увазі, що об'єкти нащадка можна вважати включеними в безліч об'єктів класу-предка. Графічно узагальнення зображуються у вигляді суцільної лінії з великою незафарбовані стрілкою, спрямованої до суперкласу.

У UML зв‘язок Узагальнення між двома класами розташовує їх у вузлах ієрархії, яка відповідає концепції успадкування класу-нащадка від базового класу. У UML узагальнення буде показано у вигляді лінії, яка поєднує два класи, зі стрілкою, яку спрямовано від базового класу.

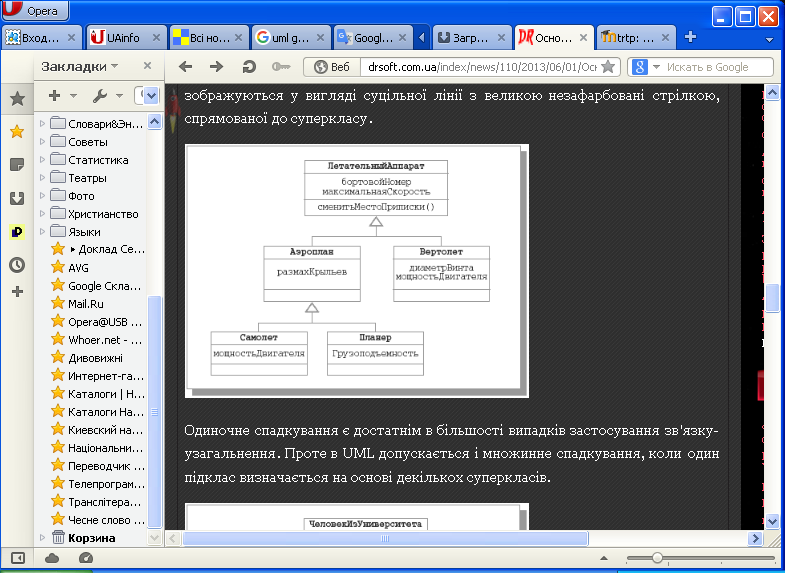
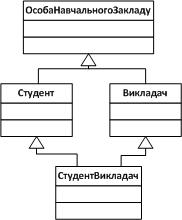


Рис.4 Наочний показ узагальнення у UML

Одиночне спадкування є достатнім в більшості випадків застосування зв'язку-узагальнення. Проте в UML допускається і множинне спадкування, коли один підклас визначається на основі декількох суперкласів.

← Рис.5 Наочний показ узагальнення у UML

На цій діаграмі класи "Студент" та "Викладач" породжені з одного суперкласу "ОсобаНавчальногоЗакладу". Взагалі кажучи, до класу "Студент" відносяться ті об'єкти класу " ОсобаНавчальногоЗакладу", які відповідають студентам, а до класу "Викладач" - об'єкти класу "ОсобаНавчальногоЗакладу", відповідні викладачам. Але, як це часто трапляється, багато студентів вже в студентські роки починають викладати, так що можуть існувати такі два об'єкти класів "Студент" і "Викладач", яким відповідає один об'єкт класу " ОсобаНавчальногоЗакладу ". Отже, серед об'єктів класу "Студент" можуть бути викладачі, а деякі викладачі можуть бути студентами. Тоді ми можемо визначити клас "СтудентВикладач" шляхом множинного наслідування від суперкласів "Студент" та "Викладач". Об'єкт класу "СтудентВикладач" має всі властивості та операціями класів "Студент" і "Викладач" та може бути використаний скрізь, де можуть застосовуватися об'єкти цих класів. Тут працює поліморфізм по включенню. Потрібно зазначити, що множинне спадкування, крім того що не надто часто потрібно на практиці, породжує ряд проблем, з яких однією з найбільш відомих є проблема іменування атрибутів і операцій у підкласі, отриманому шляхом множинного успадкування. Наприклад, припустимо, що при утворенні підкласів "Студент" та "Викладач" в них обох був визначений атрибут з ім'ям "номерКімнати". Дуже ймовірно, що для об'єктів класу "Студент" значеннями цього атрибута будуть номери кімнат у студентському гуртожитку, а для об'єктів класу "Викладач" - номери службових кабінетів. Як бути з об'єктами класу " СтудентВикладач ", для яких істотні обидва однойменних атрибуту (у студента-викладача можуть матися і кімната в гуртожитку, і службовий кабінет)? На практиці застосовується одне з таких рішень:

1. заборонити створення підкласу "СтудентВикладач", поки в одному з суперкласів не буде зроблена перейменування атрибута "номерКімнати";
2. успадковувати це властивість лише від одного з суперкласів, так що, наприклад, значенням атрибута "номерКімнати" у об'єктів класу "СтудентВикладач" завжди будуть номери службових кабінетів;
3. успадкувати у підкласі обидві властивості, але автоматично перейменувати обидва атрибуту, щоб прояснити їх зміст; назвати їх, наприклад, "номерКімнатиСтудента" і "номерКімнатиВикладача".

Жодне з рішень не є повністю задовільним. Перше рішення вимагає повернення до раніше певного класу, імена атрибутів і операцій якого, можливо, вже використовуються в додатках. Друге рішення порушує логіку спадкування, не даючи можливості на рівні підкласу використовувати всі властивості суперкласів. Нарешті, третє рішення змушує використовувати довгі імена атрибутів і операцій, які можуть стати неприпустимо довгими, якщо процес множинного спадкоємства триватиме від отриманого підкласу.

Але, звичайно, складність проблеми іменування атрибутів і операцій незрівнянно менша складності реалізації множинного спадкоємства в реляційних БД. Тому при використанні UML для проектування реляційних БД потрібно дуже обережно використовувати наслідування класів взагалі і намагатися уникати множинного успадкування.

***Асоціації (association)***

Асоціацією називається структурний зв'язок, який показує, що об'єкти одного класу деяким чином пов'язані з об'єктами іншого чи того ж самого класу. Допускається, щоб обидва кінці асоціації ставилися до одного класу. В асоціації можуть зв'язуватися два класи, і тоді вона називається бінарною. Допускається створення асоціацій, що пов'язують одразу n класів (вони називаються n-арнимі асоціаціями).) Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам із собою або з іншими класами.

З поняттям асоціації пов'язані чотири важливих додаткових поняття: ім'я, роль, кратність і агрегація. По-перше, асоціації може бути присвоєно ім'я, характеризує природу зв'язку. Смисл імені уточнюється за допомогою чорного трикутника, який розташовується над лінією зв'язку праворуч або ліворуч від імені асоціації.

****

Рис.6 Зв’язок асоціації

Асоціація означає взаємозв’язок між класами, вона є базовим семантичним елементом і структурою для багатьох типів “з’єднань” між об’єктами.

Асоціації є тим механізмом, який надає об’єктам можливість обмінюватися даними між собою. Асоціація описує з’єднання між різними класами (з’єднання між дійсними об’єктами називається об’єктним з’єднанням, або зв’язком).

Асоціації можуть виконувати роль, яка визначає призначення асоціації і може бути одно- чи двосторонньою (другий варіант означає, що у межах зв’язку кожен з об’єктів може надсилати повідомлення іншому, перший же — варіанту, коли лише один з об’єктів знає про існування іншого). Крім того, кожен з кінців асоціації має значення Кратності (multiplicity) / численності, яке визначає кількість об’єктів на відповідному кінці асоціації, які можуть мати зв’язок з одним з об’єктів на іншому кінці асоціації.

Кратністю (multiplicity) ролі асоціації називається характеристика, яка вказує, скільки об'єктів класу з даної роллю може або повинні брати участь у кожному примірнику асоціації (в UML примірник асоціації називається з'єднанням - link, але тут не будемо використовувати цей термін, щоб не створювати плутанину, оскільки важко одночасно говорити про зв'язки, асоціації та з'єднання, маючи на увазі різні поняття). Найбільш поширеним способом завдання кратності ролі асоціацію є вказівка конкретного числа або діапазону. Наприклад, вказівка «1» говорить про те, що кожен об'єкт класу з даної роллю повинен брати участь в деякому екземплярі даної асоціації, причому в кожному примірнику асоціації може брати участь рівно один об'єкт класу з даної роллю. Вказівка діапазону «0 .. 1» говорить про те, що не всі об'єкти класу з даної роллю зобов'язані брати участь в будь-якому примірнику даної асоціації, але в кожному примірнику асоціації може брати участь тільки один об'єкт. Аналогічно, вказівка діапазону «1 .. \*» говорить про те, що всі об'єкти класу з даної роллю повинні брати участь в деякому екземплярі даної асоціації, і в кожному примірнику асоціації повинен брати участь хоча б один об'єкт (верхня границя не задана). Тлумачення діапазону «0 .. \*» є очевидним розширенням випадку «0 .. 1».

У складніших (які вкрай рідко зустрічаються на практиці) випадках визначення кратності можна використовувати списки діапазонів. Наприклад, список «2, 4 .. 6, 8 .. \*» говорить про те, що всі об'єкти класу з вказаною роллю повинні брати участь в деякому екземплярі даної асоціації, і в кожному примірнику асоціації повинні брати участь дві, від чотирьох до шести або більше семи об'єктів класу з даної роллю.

У UML асоціації позначаються лініями, що з’єднують класи, які беруть участь у зв’язку, крім того, може бути показано роль і численність кожного з учасників зв’язку. Численність буде показано у вигляді діапазону [мін..макс] невід’ємних чисел, зірочка (\*) на боці максимального значення позначає нескінченність.

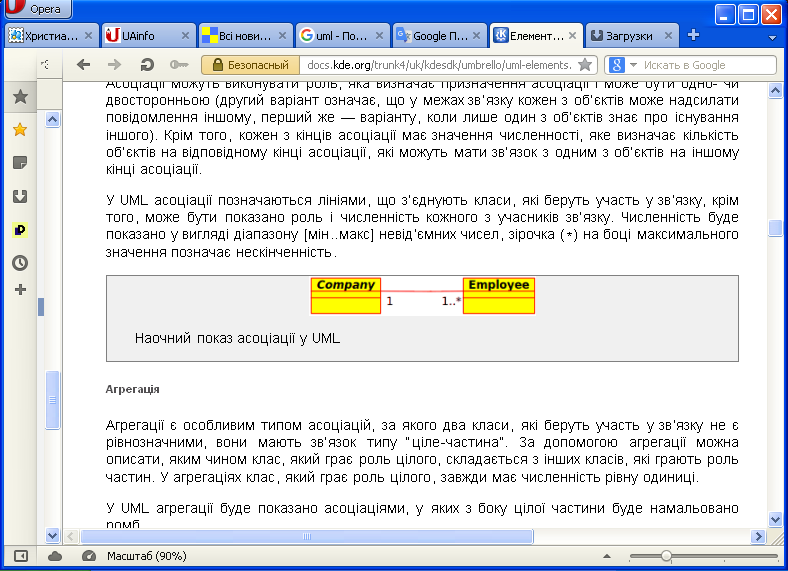


Рис.7 Наочний показ асоціації в Umbrello UML Modeller

***Агрегація***

Агрегації є особливим типом асоціацій, за якого два класи, які беруть участь у зв’язку не є рівнозначними, вони мають зв’язок типу “ціле-частина”. За допомогою агрегації можна описати, яким чином клас, який грає роль цілого, складається з інших класів, які грають роль частин. У агрегаціях клас, який грає роль цілого, завжди має численність рівну одиниці.

У UML агрегації буде показано асоціаціями, у яких з боку цілої частини буде намальовано ромб.

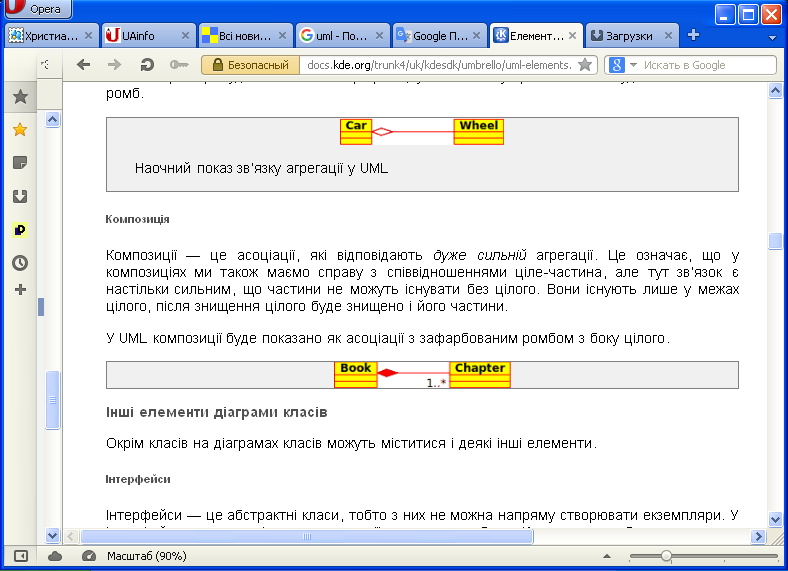


Рис.8 Наочний показ зв’язку агрегації в Umbrello UML Modeller

***Композиція***

Композиції — це асоціації, які відповідають дуже сильній агрегації. Це означає, що у композиціях також маємо справу з співвідношеннями ціле-частина, але тут зв’язок є настільки сильним, що частини не можуть існувати без цілого. Вони існують лише у межах цілого, після знищення цілого буде знищено і його частини.

У UML композиції буде показано як асоціації з зафарбованим ромбом з боку цілого.

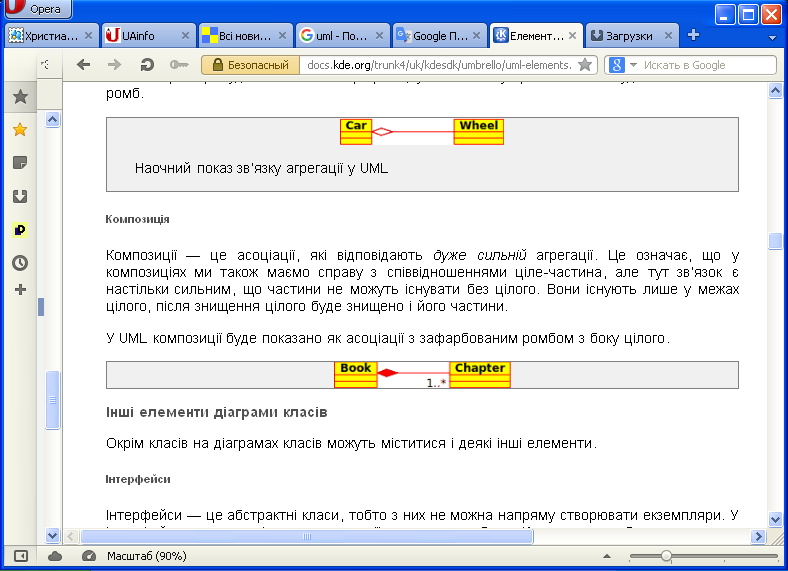


Рис.9 Наочний показ зв’язку композиції в Umbrello UML Modeller

***Інші елементи діаграми класів***

Окрім класів на діаграмах класів можуть міститися і деякі інші елементи.

Інтерфейси

Інтерфейси — це абстрактні класи, тобто з них не можна напряму створювати екземпляри. У інтерфейсах можуть міститися операції, але не атрибути. Класи можуть бути нащадками інтерфейсів (за допомогою асоціації реалізації), а з цих діаграм можна потім створювати сутності.

Типи даних

Типи даних — це базові елементи, з яких типово будується мова програмування. Типовими прикладами є цілі числа і булеві значення. Вони не можуть мати зв’язків з класами, але класи можуть мати зв’язки з ними.

Переліки

Переліки є простими списками значень. Типовим прикладом є перелік днів тижня. Пункти переліків називаються літералами переліків. Подібно до типів даних, переліки не можуть мати зв’язків з класами, але класи можуть мати зв’язки з переліками.

Пакунки

Пакункам відповідають простори назв у мовах програмування. На діаграмі пакунки використовуються для позначення частин системи, у яких міститься декілька класів, може навіть сотні класів.

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

*Література*.

1. Г. Буч, Дж. Рамбо , А. Джекобсон Язьік UML. Руководство пользователя.: Пер. с англ. - М.: ДМК, 2000. - 432с. URL: <http://www.dut.edu.ua/uploads/l_1508_23942623.pdf>, <http://bourabai.kz/dbt/uml/content.html>.
2. Мацяшек, Лешек А. Анализ требований и проектирование систем : Разраб. информ. систем с использованием UML / Лешек А. Мацяшек; [Пер. с англ. и ред. В.М. Неумоина]. - М. [и др.] : Вильямс, 2002. - 428 с. : ил.
3. Вендров A.M. CASE-технологии. Современные методы и средства проектирования информационных систем. - М.: Финансы и статистика, 1998.
4. Леоненков А.В. Самоучитель UML. 2-е издание - СПб.: "БХВ-Петербург", 2004. - 432 с.
5. Буч Г., Рамбо Дж., Якобсон А. UML: специальный справочник - СПб: "Питер", 2001. - 656 с.

*Запитання для самоперевірки.*

1. На які групи можна поділити UML-діаграми?
2. Яка стратегія використання *UML*-діаграм при моделюванні програмних систем?
3. Яке призначення діаграм використання?
4. Яке призначення діаграм класів?
5. Якими відносинами можуть бути пов'язані сутності діаграми?
6. Які типи візуальних позначень існують в UML?
7. Які діаграми відносяться до статичних?
8. Чи може статична діаграма бути і поведінковою?

*Запитання для письмової відповіді.*

1. В ЛР№2 потрібно було створити таблицю з описом вимог до застосунку. Які вимоги можуть бути висунуті (виходячи з постановки задачі), що б вони мали відношення розширення та включення варіантів використання?
2. Які класи потрібно створити для реалізації постановки задачі з ЛР№2.
3. Наведіть власні приклади різних видів зв’язків між класами.