



{**PROGRAMMING**}



**UNITED  
MODELING  
LANGUAGE**

# Урок №4

Діаграма послідовності, діаграма кооперації, діаграма компонентів і діаграма розгортання

## ЗМІСТ

|   |           |
|---|-----------|
| <b>1. Діаграма послідовності.....</b>                       | <b>4</b>  |
| 1.1. Цілі даного типу діаграм.....                          | 4         |
| 1.2. Базові поняття.....                                    | 4         |
| 1.2.1. Об'єкти.....   | 4         |
| 1.2.2. Повідомлення.....                                    | 11        |
| 1.3. Практичні приклади побудови діаграм послідовності..... | 19        |
| <b>2. Діаграма кооперації.....</b>                          | <b>22</b> |
| 2.1. Цілі даного типу діаграм.....                          | 22        |
| 2.2. Базові поняття.....                                    | 22        |
| 2.2.1. Кооперація.....                                      | 22        |
| 2.2.2. Діаграма кооперації рівня специфікації... ..         | 23        |
| 2.2.3. Об'єкти.....   | 26        |
| 2.2.4. Зв'язки.....   | 31        |

|  |           |
|--|-----------|
| 2.2.5. Стереотипи зв'язків .....                                 | 32        |
| 2.2.6. Повідомлення .....  | 33        |
| 2.2.7. Формат запису повідомлень .....                           | 35        |
| 2.3. Практичні приклади побудови діаграм<br>кооперації .....     | 36        |
| <b>3. Діаграма компонентів .....</b>                             | <b>38</b> |
| 3.1. Цілі даного типу діаграм .....                              | 38        |
| 3.2. Базові поняття .....  | 38        |
| 3.2.1. Компонент .....   | 39        |
| 3.2.2. Залежності .....  | 42        |
| 3.2.3. Інтерфейс .....   | 44        |
| 3.3. Практичні приклади побудови діаграм<br>компонентів .....    | 46        |
| <b>4. Діаграма розгорткування .....</b>                          | <b>49</b> |
| 4.1. Цілі даного типу діаграм .....                              | 49        |
| 4.2. Базові поняття .....  | 49        |
| 4.3. Практичні приклади побудови діаграм<br>розгорткування ..... | 52        |

# 1. Діаграма послідовності

## 1.1. Цілі даного типу діаграм

Моделювання системи має два основні аспекти: моделювання структури системи (набору складових її компонентних елементів) і визначення зв'язків між компонентами системи.

Говорячи про зв'язок, ми розуміємо, що компоненти системи певним чином взаємодіють один з одним. Один з основних принципів об'єктно-орієнтованого аналізу та проектування наголошує, що будь-які способи інформаційного обміну між елементами системи мають виражатися у надсиланні та отриманні ними повідомлень один від одного.

Для представлення специфіки взаємодії компонентів системи один з одним, в UML використовуються два типи діаграм, які розглянемо в цьому уроці. Це діаграми послідовності і кооперації. Якщо діаграми кооперації відбивають структурну сторону взаємодії елементів системи, то діаграми послідовності покликані показувати тимчасову сторону цієї взаємодії, у чому полягає основна мета їх використання.

## 1.2. Базові поняття

### 1.2.1. Об'єкти

Для визначення складових елементів системи до цього ми використовували терміни «компонент» та «елемент». Але, коли йде мова про реальну взаємодію, то ми, як правило, вже маємо дію з реальними об'єктами (*objects*)

(будь то програмний об'єкт — діючий екземпляр деякого класу або об'єкт предметної дійсності). Тому, в діаграмах послідовності об'єкт є ключовим поняттям, що описує джерело активності.

Взаємодія передбачає наявність кількох учасників, тому на діаграмах послідовності ми матимемо справу з кількома об'єктами. Варто згадати, що діаграма послідовності містить ті об'єкти, які беруть участь в описуваній взаємодії.

Об'єкт на діаграмах послідовності зображується у вигляді прямокутника, розташованого на початку своєї лінії життя (поняття лінії життя буде розглянуто далі у відповідному розділі), з написаними у ньому назвою об'єкта і класом, до якого об'єкт належить (рис. 1).

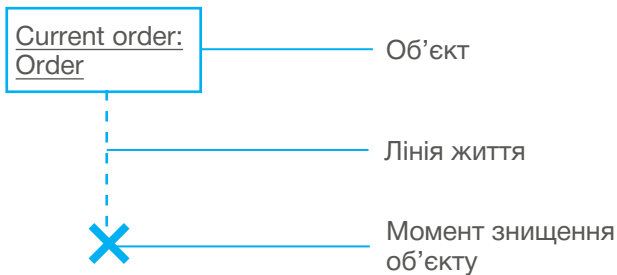


Рисунок 1.

Назва об'єкта від назви класу відокремлюється двокрапкою, як це показано на рисунку. Однак, там, де вказувати назву класу не обов'язково, можна вказати тільки назву об'єкта.

Об'єкти на діаграмі мають розташовуватися зліва направо таким чином, щоб крайнім зліва був той об'єкт, який ініціює взаємодію (рис. 2).

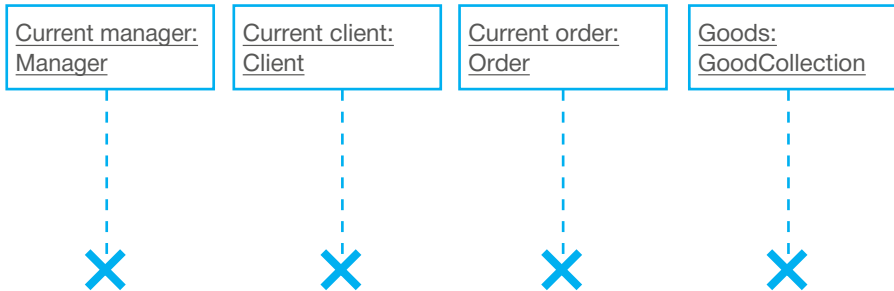


Рисунок 2.

Тимчасова шкала у діаграмі направлена зверху вниз. Тому, початковому моменту взаємодії відповідає верхня частина діаграми.

### Лінія життя об'єкта

Лінія життя об'єкта (*object lifeline*) використовується для того, щоб продемонструвати, який період часу існує об'єкт в системі і, відповідно, може брати участь у взаємодії.

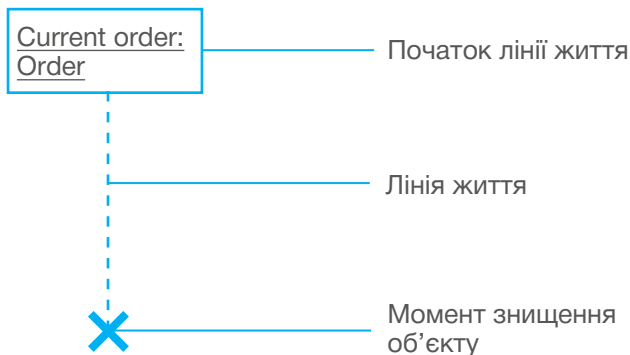


Рисунок 3.

Лінія життя, як було показано на рисунку 3, зображується у вигляді вертикальної пунктирної лінії, спрямованої

від позначення об'єкта зверху вниз до моменту «знищення» об'єкта, який також можна назвати точкою завершення лінії життя. Момент знищення об'єкта зображується у вигляді латинської літери «X», яка розташована в нижній частині лінії життя об'єкта.

Необхідно пам'ятати, що далеко не всі об'єкти існують з початку здійснення взаємодії, описаної в діаграмі. Деякі об'єкти можуть використовуватися і як допоміжні. Вони створюються при необхідності і знищуються, щойно потреба і їх існуванні зникає. Подібне використання об'єктів, які мають ситуативне значення, як правило, пов'язане з раціональною витратою ресурсів системи з метою підвищення її ефективності. Тимчасові об'єкти зображуються таким чином, аби прямокутник, який позначає об'єкт, знаходився не у верхній частині діаграми, а був зміщений по вертикалі до того місця, яке відповідає моменту його створення, як це показано на рисунку 4.

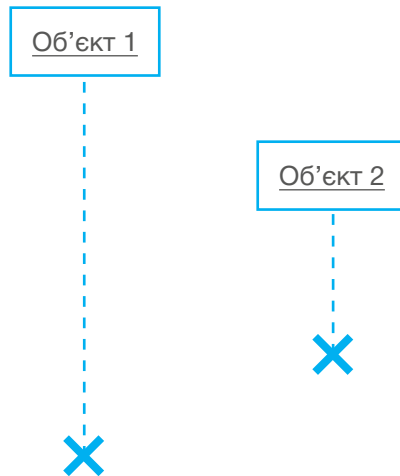


Рисунок 4.

### Фокус керування

Фокус управління (*focus of control*) — це графічне відображення активного стану об'єкта у межах його лінії життя на діаграмі.

Протягом усієї лінії життя об'єкта його активний стан може змінюватися пасивним станом. Під активністю розуміється виконання об'єктом деяких дій. Однак, далеко не кожен об'єкт протягом усієї своєї лінії життя постійно перебуває в активному стані. Тому, в мові моделювання UML вводять поняття «фокус управління», яке графічно зображується у вигляді витягнутого прямокутника, розташованого на лінії життя об'єкта, і відповідного активному стану об'єкта. Під «пунктиром лінії життя» розуміється стан очікування об'єктом повідомлень, які ініціюють його активність. На запропонованому рисунку 5 представлено графічне уявлення активного стану лінії життя об'єкта.

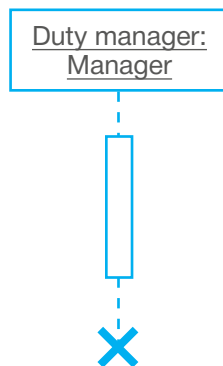


Рисунок 5.

Бувають такі ситуації, коли об'єкт перебуває в активному стані протягом усього існування в системі. У таких



випадках його лінія життя замінюється зображенням фокусу управління, як показано на рисунку 6.

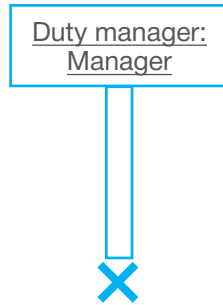


Рисунок 6.

Бувають такі ситуації, в яких джерелом активності виступає користувач або стороння, по відношенню до системи «дійова особа». Тоді, у якості ініціатора активності (крайнього об'єкта зліва) зображують «дійову особу» (*actor*). При цьому, лінія життя «дійової особи» не має символу знищення, що підкреслює характерну особливість користувача системи бути присутнім на всьому періоді активності системи. Дійова особа може мати ім'я, а може виступати як аноніма. Приклад графічного зображення анонімної дійової особи наведено на рисунку 7.



Рисунок 7.

В діаграмах послідовності є спеціальне позначення для зображення рекурсивного виклику. У таких випадках прийнято казати, що об'єкт ініціює взаємодію із самим собою. Рекурсія має вигляд вертикального витягнутого прямокутника, прикріпленого з правого боку фокусу управління об'єкта, який здійснює рекурсивну взаємодію.

На рисунку 8 наведено приклад графічного зображення рекурсивної взаємодії.

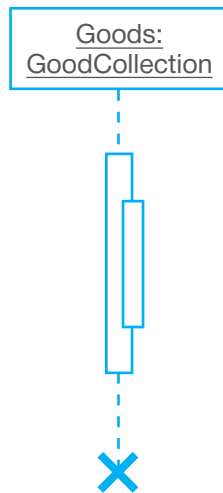


Рисунок 8.

Є також ситуації, у яких об'єкт ініціює звернення до себе не рекурсивного характеру (наприклад, це може інтерпретуватися на рівні специфікації як виклик об'єктом власного методу). Тоді говорять про рефлексивне повідомлення, яке графічно зображується як стрілка, спрямована з деякого фокусу управління на нього. Приклад рефлексивного повідомлення показаний на рисунку 9.

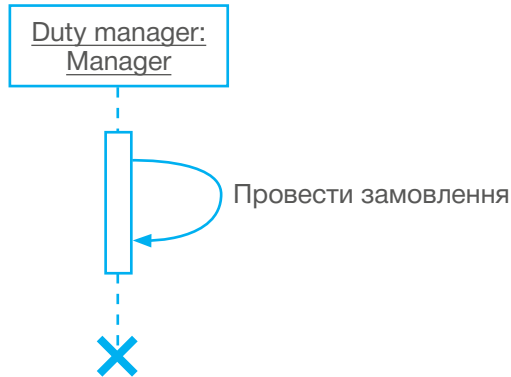


Рисунок 9.

### 1.2.2. Повідомлення

Як згадувалося на початку розділу, мета діаграм послідовності полягає в тому, щоб відобразити тимчасовий аспект взаємодії об'єктів системи. Основним напрямом у межах цієї мети є зображення самої взаємодії.

Як вже зазначалося, що згідно з принципами об'єктно-орієнтованого аналізу та проектування, будь-яка взаємодія об'єктів певної системи може здійснюватися лише за допомогою надсилання повідомлень між об'єктами. У такому контексті, повідомлення (*message*) — це закінчений фрагмент інформації, який один об'єкт надсилає іншому. Також передбачається, що факт прийому об'єктом повідомлення ініціює деякі дії, які повинні бути виконані об'єктом, який отримав повідомлення.

Зазвичай говорять, що повідомлення ініціює виконання деяких операцій, а аргументи (параметри) цих операцій передаються в «тілі» повідомлення. При цьому, повідомлення мають бути упорядковані за часом їх виникнення в системі.

Повідомлення на діаграмах послідовності графічно зображуються у вигляді горизонтальних стрілок, які з'єднують лінії життя або фокуси керування двох об'єктів. Стрілки повідомлень мають бути спрямованими від об'єктів, що надсилають повідомлення (їх називають клієнтами), до об'єктів, які їх отримують (такі об'єкти називають серверами).

Прийнято вважати, що повідомлення має форму запити, а реакцією сервера з його виконання є «відповідь», тобто, зворотне повідомлення клієнту про те, що запитовані дії були виконані. Відповідь також передається у формі повідомлення. Відповідь, зазвичай, має вигляд горизонтальної пунктирної стрілки, спрямованої від сервера до клієнта (рис. 10).

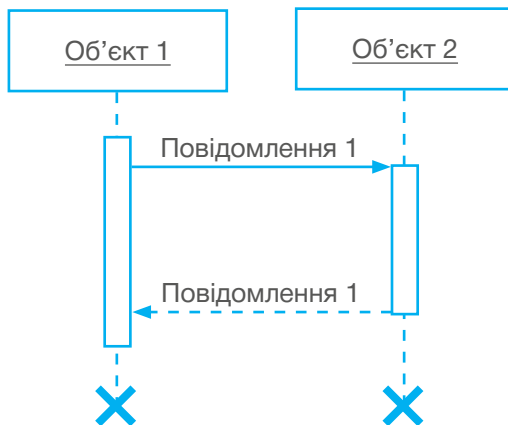


Рисунок 10.

Приймається припущення про те, що часу на передачу повідомлення відносно мало у порівнянні з часом, який витрачається на виконання запитованої дії об'єктом-сервером. Також, прийнято вважати, що за час передачі

повідомлення з відповідним об'єктом не може відбутися жодних змін.

У мові моделювання UML зустрічаються різні види повідомлень, для кожного з яких визначене своє графічне позначення:

- **Синхронне повідомлення** — найпоширеніший вид повідомлень. Він використовується для виклику процедур, виконання операцій та позначення вкладених потоків керування. Початок такої стрілки завжди сти-кається з лінією життя об'єкта або фокусом управління об'єкта-клієнта, а кінець — з лінією життя об'єкта-сервера. При цьому типі з'єднання, приймаючий об'єкт зазвичай отримує фокус управління, стаючи активним (рис. 11).

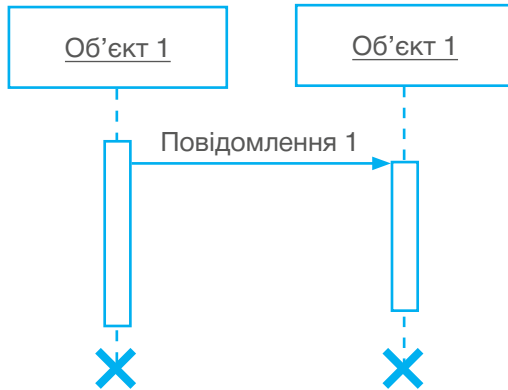


Рисунок 11.

- **Повідомлення виклику** — використовується для позначення невикладеного потоку керування. Така стрілка, зазвичай, зображується спрямованою від лінії життя (фокус управління) одного об'єкта до

фокус управління об'єкта-сервера і вказує на один крок потоку управління. Подібні виклики, зазвичай, асинхронні (рис. 12).

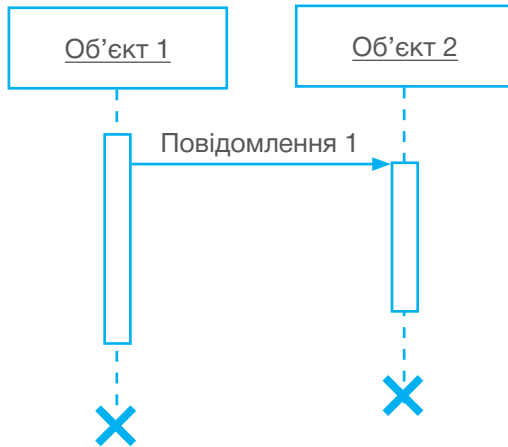


Рисунок 12.

- **Асинхронне повідомлення** — визначає повідомлення між двома об'єктами у певній послідовності (рис. 13).

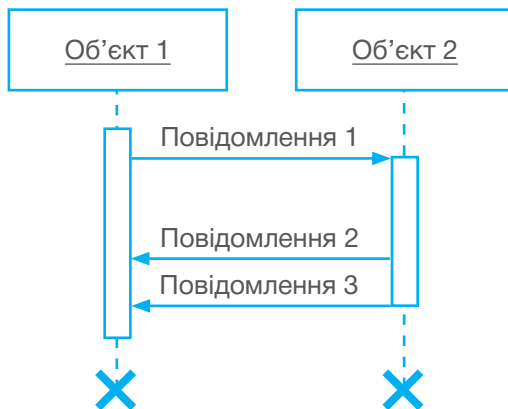


Рисунок 13.

Прикладом у чистому вигляді асинхронного повідомлення є надсилання об'єктом-сервером повідомлень про те, що процедура переходить на наступний етап. Або час повідомлення, коли виникла помилка.

- **Повідомлення повернення** — визначає, що сервер по виконанні дії поверне у викликаний об'єкт певний результат. Повідомлення повернення графічно зображується горизонтальною пунктирною стрілкою, спрямованою від об'єкта-сервера до об'єкта-клієнта. У процедурних потоках управління, повідомлення повернення може бути вилучене через те, що з контексту стає зрозумілим, що сервер має повертати результат. Також вважається, що кожен виклик процедури має парне повернення. Для асинхронних викликів, стрілка повернення має бути вказана явно, оскільки з контексту не відомо, чи передбачає кожен конкретний асинхронний виклик повернення результату (рис. 14).

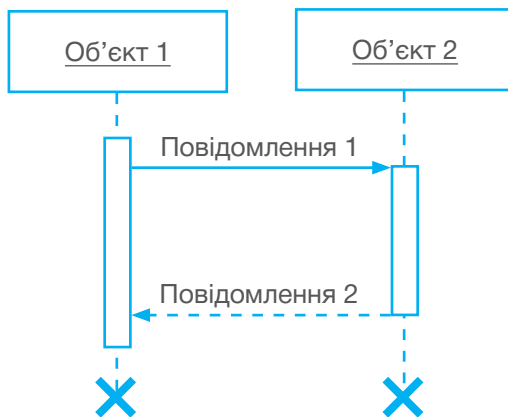


Рисунок 14.

### Розгалуження потоку керування

Розгалуження потоку керування графічно зображується кількома стрілками, що виходять з одного фокусу управління клієнтського об'єкта до кількох об'єктів-серверів. При цьому, біля кожної стрілки у квадратних дужках вказується умова надсилання відповідного повідомлення. Передбачається, що умови, використовувані при розгалуженні, мають взаємно виключати одна одну.

Нижче на рисунку 15 наведено приклад графічного зображення розгалуження у діаграмах послідовності мови моделювання UML.

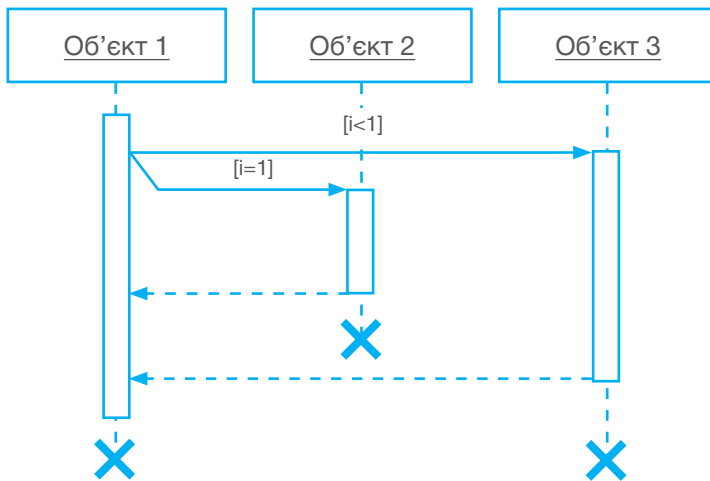


Рисунок 15.

### Стереотипи повідомлень

Стереотип повідомлення — це стандартна дія, яка виконується у відповідь на повідомлення. Стереотип може бути вказаний у подвійних лапках біля повідомлення (як правило, над з'єднанням з вирівнюванням по центру).



На рисунку 16 наведено приклад графічного оформлення використання стереотипу повідомлення.

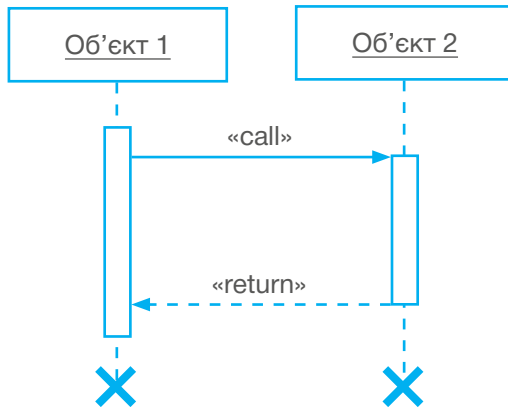


Рисунок 16.

У мові моделювання UML передбачено набір стандартних стереотипів:

- «call» (викликати) — повідомлення, яке передбачає виклик процедури об'єкта-сервера;
- «return» (повернути) — повідомлення, яке повертає значення завершеної процедури;
- «create» (створити) — повідомлення, яке вимагає створення нового об'єкта, для виконання певних операцій;
- «destroy» (знищити) — повідомлення, яке вимагає знищення відповідного об'єкта.
- «send» (надіслати) — означає надсилання певного сигналу приймаючому об'єкту. Відмінність сигналу від повідомлення полягає в тому, що сигнал повинен бути явно визначений у класі, об'єкт якого є відправником.

Повідомлення можуть мати також вказівки конкретних операцій, які вони ініціюють у приймаючих об'єктах. Для специфікації виклику операції в приймаючому об'єкті, поруч зі стрілкою повідомлення необхідно вказати назву операції, виклик якої здійснюється, і круглі дужки після неї, в яких можна вказати аргументи цієї операції.

### *Тимчасові обмеження на діаграмах послідовності*

Бувають такі випадки, коли виникає потреба вказати обмеження часу під час надсилання повідомлення. Обмеження можуть накладатися як на процес передачі повідомлення, так і на час виконання викликаної операції.

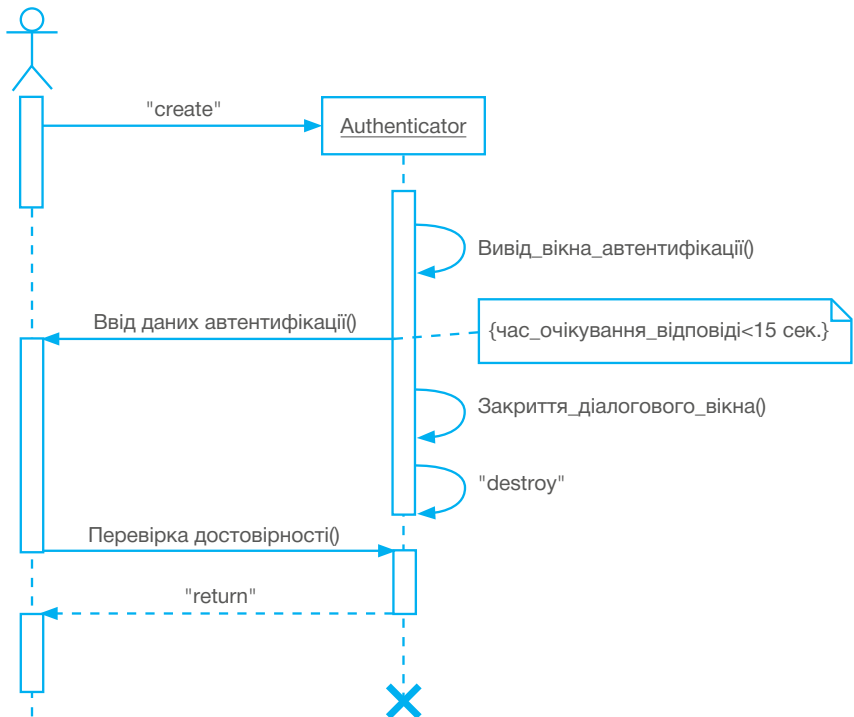


Рисунок 17.

Тимчасове обмеження записується в такому ж контейнері (як і коментарі) з тією відмінністю, що для запису тимчасових обмежень у діаграмах послідовності використовують фігурні дужки. Тимчасові обмеження можуть бути вказані на початку стрілки повідомлення. Хоча, зазвичай, їх вказують ліворуч від стрілки відповідного повідомлення. Якщо тимчасове обмеження накладається на конкретний об'єкт, спочатку вказується ім'я об'єкта, потім ставимо крапку, а далі специфікується тимчасове обмеження на об'єкт.

Важливо пам'ятати, що тимчасові обмеження — це не умови, вони мають характер директив. На рисунку 17 наведено приклад використання тимчасових обмежень у діаграмах послідовності мови моделювання UML.

### *Коментарі або примітки*

Коментарі (примітки) — важлива особливість будь-якої формальної мови, зокрема UML. Їх важливість у тому, що вони дозволяють деталізувати і пояснити неочевидні і двозначні формулювання (в даному випадку — моделях). Як вже зазначалося, коментарі вказують у прямокутному контейнері із заломленим верхнім правим кутом, а від самого контейнера йде пунктирне лінійне з'єднання до коментованого формалізму.

## **1.3. Практичні приклади побудови діаграм послідовності**

Як приклад ми розглянемо побудову діаграми послідовності роботи об'єкта класу DBAdapter, який є адаптером для підключення і виконання запитів до серверів баз даних SQL Server і Oracle.

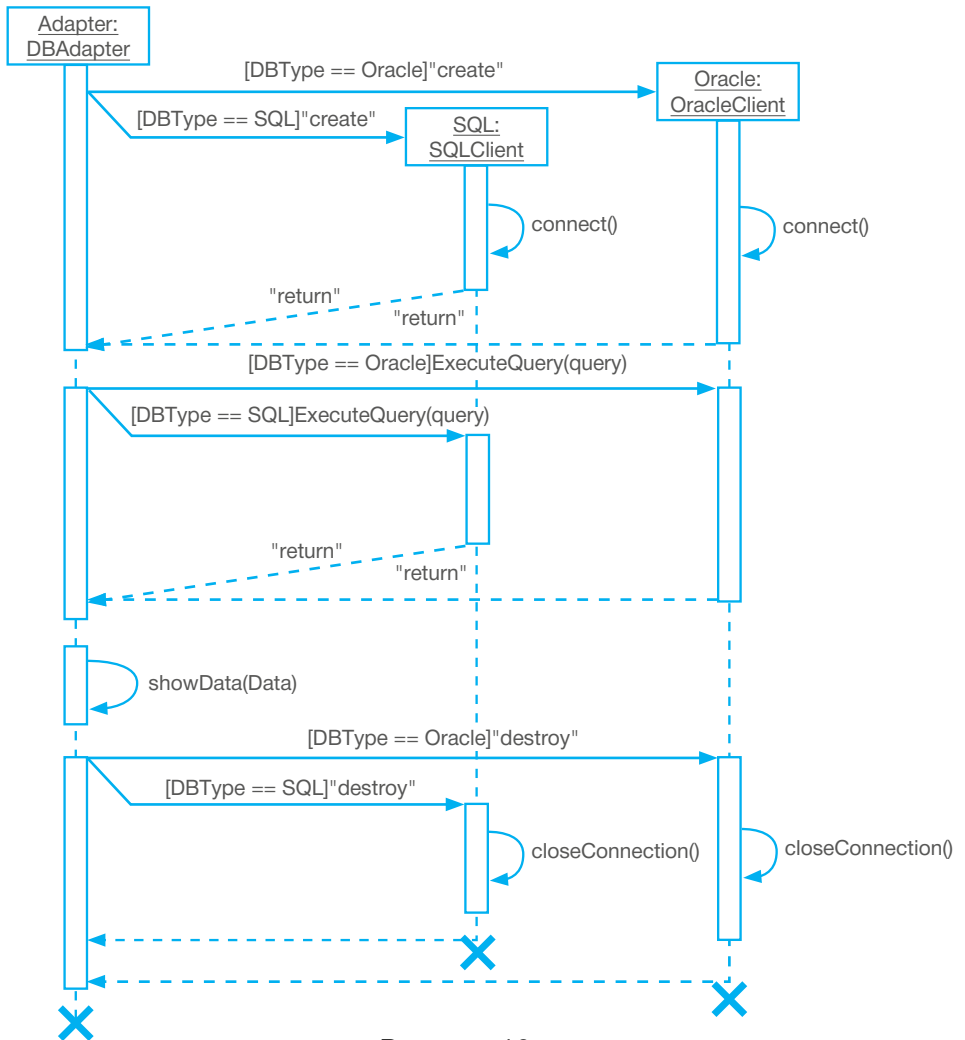


Рисунок 18

Умовою використання класу є завдання прапору, що визначає з яким СУБД здійснюватиметься взаємодія. Беручи за основу цей прапорець, адаптер створює необхідний об'єкт клієнта баз даних і згодом використовує його для виконання запитів до необхідної бази даних. Після

того, як дані отримані і виведені користувачу, адаптер знищує об'єкт клієнта, який реалізує взаємодію, а той, в свою чергу, закриває з'єднання з СУБД перед знищенням (рис. 18).

## 2. Діаграма кооперації

### 2.1. Цілі даного типу діаграм

Як і діаграми послідовності, діаграми кооперації використовуються для зображення особливостей взаємодії об'єктів модельованої системи. Але, на відміну від діаграм послідовності, діаграми кооперацій призначені для відображення структурного аспекту описуваної в них взаємодії.

Діаграми кооперації відображають лише відносини між об'єктами, які мають певний сенс із погляду взаємодії, а не їх послідовності. З іншого боку, послідовність викликів на діаграмах кооперації може бути відображена за допомогою специфікації порядкових номерів в об'єктах. Але цей спосіб, все одно, повною мірою не відображає часовий аспект взаємодії. Тому, важливо розуміти, що для комплексного відображення моделі мало використовувати якийсь один спосіб уявлення.

Проте, тимчасовий аспект важливий не на всіх рівнях абстракції. Наприклад, з погляду аналітики або архітектури важливий скоріш структурний опис моделі, який висвітлює все багатство складових системи об'єктів і зв'язків, існуючих між ними. Таке структурне уявлення і є метою використання діаграм кооперації.

### 2.2. Базові поняття

#### 2.2.1. Кооперація

Кооперація (*collaboration*) — формалізм мови моделювання UML, який слугує для відображення об'єктів, що

взаємодіють з певною метою. Мета кооперації полягає в тому, щоб проілюструвати особливості основних операцій системи (тих операцій, які мають найбільше значення з точки зору системи).

Кооперація має два рівні відображення:

- **рівень специфікації**, на якому відображаються ролі класифікаторів та асоціацій, існуючих у рамках певної взаємодії;
- **рівень прикладів** ілюструє екземпляри (об'єкти) і зв'язки, які утворюють «ролі» в межах кооперації.

Діаграма кооперацій рівня специфікації відображає ролі, виконувані елементами у взаємодії. Під елементами на цьому рівні маються на увазі класи і асоціації.

У діаграмі рівня прикладів посідають важливе місце не класи та асоціації, а їх екземпляри, — об'єкти і зв'язки (зв'язки на цьому рівні доповнені стрілками повідомлень, які вказують напрямом передачі повідомлення).

### 2.2.2. Діаграма кооперації рівня специфікації

Кооперацію рівня специфікації зображують пунктирним еліпсом із назвою кооперації всередині нього. Важливо пам'ятати, що кооперація рівня специфікації має бути пов'язана з конкретним варіантом використання і покликана описати деталі реалізації, оскільки, залежно від різних варіантів використання, особливості кооперації можуть різко відрізнятися.

Учасники кооперації поєднуються з нею пунктирними лініями. Учасникам кооперації можуть бути об'єкти або класи. При цьому, біля кожної сполучної лінії необхідно

вказати роль (*role*) учасника у межах кооперації (ролі, зазвичай, відповідають назвам елементів в контексті взаємодії).

На зображенні (рис. 19) показано, який графічний вид мають кооперації та з'єднання на діаграмах кооперації.

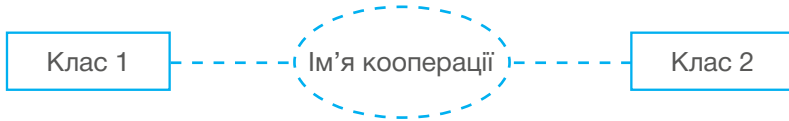


Рисунок 18.

З наведеного малюнка ми бачимо, що клас графічно зображується у вигляді прямокутника із записаним рядком тексту у ньому, який називається роллю класифікатора (*classifier role*). Роль класифікатора має показати особливість використання об'єкта у взаємодії. Роль класифікатора, зазвичай, містить тільки вказівку назви класу. Хоча, можливо також вказувати секції атрибутів і операцій. Назва ролі класифікатора має зазначатися в наступному форматі:

```
назва_ролі_класифікатора: назва_класифікатора
```

Назва класифікатора — це, як правило, назва класу (типу даних), але також можна вказати повний шлях до назви з повною специфікацією всіх пакетів (якщо пакети вказуються, необхідно специфікувати всі пакети в зазначеному шляху). При цьому, назви пакетів відокремлюються один від одного подвійною двокрапкою («::»).

Якщо кооперація має на увазі узагальнене уявлення, тобто одна кооперація є окремим випадком іншої кооперації, то на діаграмі можуть вказуватися відносини



узагальнення між відповідними коопераціями. Для цього використовують звичайний зв'язок узагальнення, який є суцільною лінією з не зафарбованою стрілкою на кінці. Ставлення узагальнення має бути спрямоване від кооперації-нащадка до кооперації-предка, як це показано на наведеному на рисунку 19 прикладі.

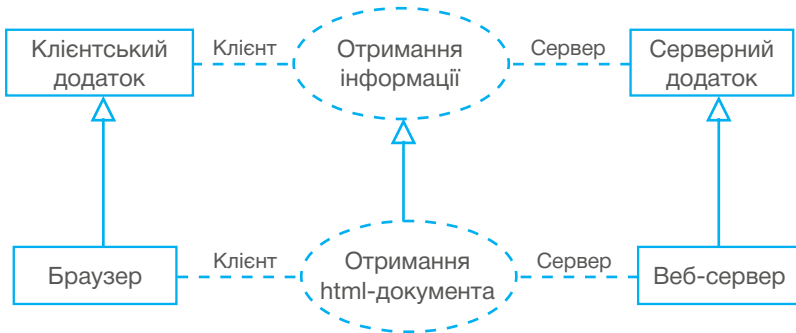


Рисунок 19.

Іноді виникає потреба явно специфікувати, що кооперація є реалізацією певної операції чи класифікатора. В такому випадку використовують два способи графічного відображення:

- **перший спосіб** полягає в тому, щоб поєднати кооперацію з класом пунктирною лінією, яка закінчується стрілкою узагальнення (рис. 20);



Рисунок 20.

- **другий спосіб** полягає в тому, щоб вказати через дві точки після назви кооперації специфікацію того, реа-

лізацією якої операції і якого класу є така кооперація (рис. 21).

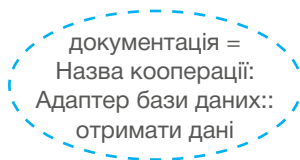


Рисунок 21.

Специфікація операції вказується у такому форматі:

```
назва_кооперації: назва_класифікатора::  
назва_операції_класифікатора
```

Опис кооперацій починається з рівня специфікації, на якому, власне, і вказуються усі узагальнення та реалізації. Після цього кожену кооперацію потрібно описати на рівні прикладів для того, щоб розкрити особливості внутрішньої структури взаємодії з описом характеру зв'язків і ролі, в яких виступають об'єкти в рамках взаємодії.

На діаграмі кооперації рівня специфікації можуть бути іменовані класи із зазначенням ролі класу, а також і анонімні класи, але тільки із зазначенням ролі.

### 2.2.3. Об'єкти

Нагадаємо, що під об'єктом (*object*) розуміється окремий екземпляр класу, який створюється на етапі виконання програми. Він наділений власним ім'ям і конкретними значеннями атрибутів.

Графічно, об'єкт зображується у вигляді прямокутника з підкресленим текстом всередині (саме підкреслення

дає можливість швидко, візуально відрізнити об'єкти від класів), записаний у наступному форматі:

назва\_об'єкта/назва\_ролі\_класифікатора:назва\_класифікатора

Назва ролі класифікатора може не вказуватися (в такому випадку назву необхідно опустити разом із двокрапкою). Назва ролі опускається у разі, якщо є лише одна роль, у якій можуть виступати об'єкти даного класифікатора, а отже, досить буде вказати назву класифікатора або ролі. Далі ми надаємо можливі варіанти різних сигнатур назв об'єкта:

Таблиця 1

| Шаблони назв | Опис  |
|--------------|---|
| :A           | Анонімний об'єкт — екземпляр класу A                      |
| /Role        | Анонімний об'єкт у ролі Role                              |
| /Role:A      | Анонімний об'єкт — екземпляр класу A у ролі Role          |
| A/Role       | Об'єкт з назвою A у ролі Role                             |
| A:C          | Об'єкт з назвою A — екземпляр класу C                     |
| A/Role:B     | Об'єкт з назвою A, який є екземпляром класу B у ролі Role |
| A            | Об'єкт з назвою A   |
| A:           | Об'єкт невизначеного класу з назвою A                     |

На рисунку 22 наведено приклад того, як виглядатимуть об'єкти з різною сигнатурою назви на діаграмі.

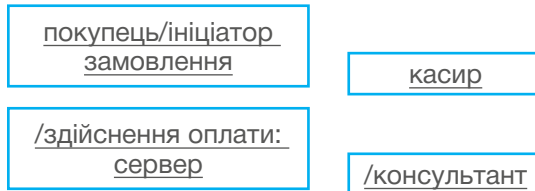


Рисунок 22.

## Мультиоб'єкт

Мультиоб'єкт (*multiobject*) — це спосіб зобразити безліч об'єктів на одному з кінців асоціації у межах діаграми кооперації. Мультиоб'єкт використовують для того, аби показати, що сигнал відправляється відразу всім елементам певної множини об'єктів (рис. 23).

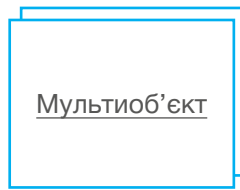


Рисунок 23.

Графічно, мультиоб'єкт зображується двома прямокутниками, накладеними один на одного, при цьому один із них виступає через верхню праву вершину іншого.

Також, може бути явно специфіковано зв'язок композиції між об'єктом і мультиоб'єктом, до якого він належить (рис. 24).

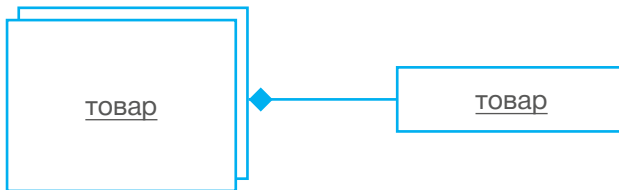


Рисунок 24.

## Активний об'єкт

Усі об'єкти можна умовно поділити на два типи, які у термінах мови моделювання UML мають назву «активні» і «пасивні».

Пасивні об'єкти виконують операції лише над даними і не можуть керувати іншими об'єктами. Однак, пасивні об'єкти, в межах їх запитів, мають можливість відправляти сигнали іншим об'єктам.

Кожен активний об'єкт (*active object*) має ряд або потік (*thread*) управління, і може керувати іншими об'єктами. При цьому передбачається, що потік управління може виконуватись паралельно з іншими потоками управління в рамках одного обчислювального процесу, який називають процесом управління (при цьому поняття процесу і потоку співвідносні з однойменними категоріями операційної системи, і розрізняються за принципом виділення ресурсів; таким чином, під процесом управління можна розуміти потік управління, який монополізує ресурси і включає інші потоки управління).

Активні об'єкти зображуються як звичайні об'єкти, прямокутник яких має ширші сторони. Також можливий варіант, коли активний об'єкт маркується ключовим словом «`{active}`». Активний об'єкт може ініціювати один потік управління і є вихідною точкою даного потоку управління.

На рисунку 25 показан приклад, у якому об'єкт касир ініціює створення об'єкта класу замовлення з однойменною роллю.

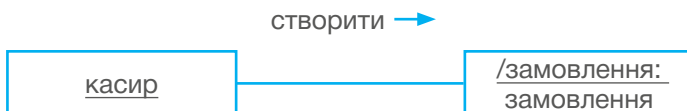


Рисунок 25.

Також, потоки можуть бути пронумеровані, щоб вказати, якою послідовністю вони створюються.

На рисунку 26 наведено приклад, в якому об'єкт «касир» створює об'єкт «поточне замовлення», який потім модифікується викликом відповідної дії. В даному випадку, на діаграмі вказаний лише один активний об'єкт — «касир».

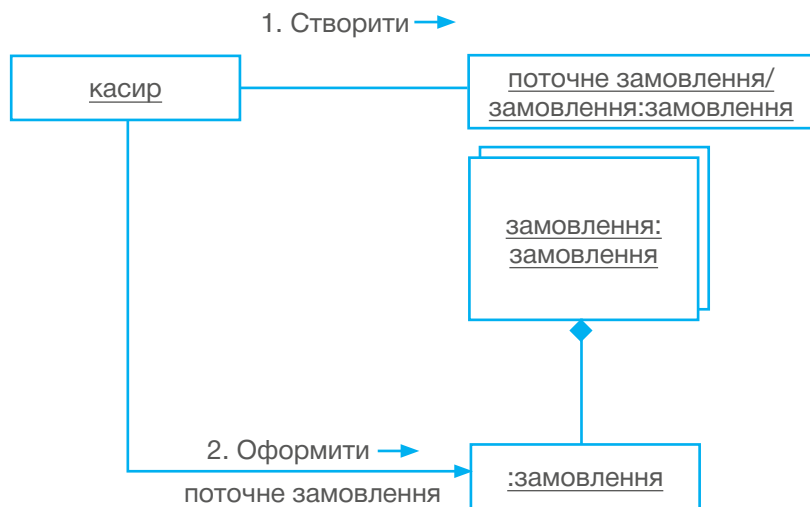


Рисунок 26.

### Складовий об'єкт

Складовий об'єкт (*composite object*), який також називають об'єктом-контейнером і призначений для графічного зображення об'єкта, що має власну структуру і власні внутрішні потоки управління. Складовий об'єкт — це екземпляр складового класу (або класу-контейнера), тобто класу, пов'язаного відносинами агрегації або композиції зі своїми компонентними частинами. Очевидно, що відповідні зв'язки пов'язують і екземпляри цих класів.

Графічно, складовий об'єкт зображується як звичайний об'єкт, замість атрибутів якого вказані прямокутники його компонентних класів.

На рисунку 27 наведено приклад з використанням складового об'єкта. На прикладі показано структуру об'єкта класу Window (вікно) з компонентними об'єктами, що забезпечують відповідний вікну функціонал (а саме: горизонтальну і вертикальну прокрутку, робочу область і рядок стану).

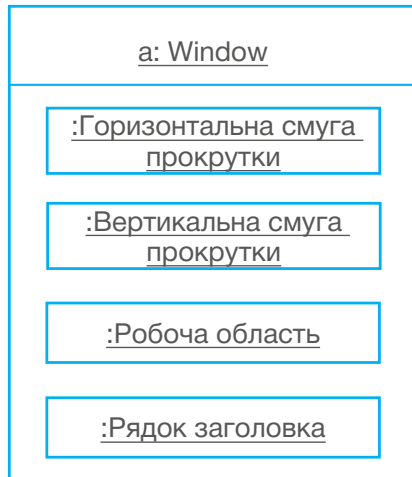


Рисунок 27.

#### 2.2.4. Зв'язки

Як згадувалося раніше, асоціації, як і класифікатори, теж можуть мати екземпляр. Екземпляр асоціації називають зв'язком (*link*). Зв'язок може існувати між двома і більше об'єктами. Бінарний зв'язок (зв'язок двох об'єктів) у діаграмах кооперації графічно зображується у вигляді суцільної лінії, яка проходить від прямокутника одного

об'єкта до прямокутника іншого об'єкта. На обох клонах зв'язку можуть бути явно вказані ролі відповідних об'єктів, а поруч із лінією, посередині, може бути вказана назва цієї асоціації.

Зв'язки не мають назв і для них не вказується кратність. Хоча, в окремих випадках асоціації (таких як композиція та агрегація) можуть бути зазначені релевантні їм позначення.

### 2.2.5. Стереотипи зв'язків

У мові UML передбачені стереотипи зв'язків, які вказують на особливість реалізації зв'язку, для якого вони вказані. Нижче надається список канонічних стереотипів зв'язків:

- “association” (асоціація) — цей стереотип можна не вказувати, оскільки за замовчуванням передбачається, що зв'язок є екземпляром асоціації.
- “parameter” (параметр методу) — відповідний об'єкт такого зв'язку може бути лише ролі параметра певного методу.
- “local” (локальна змінна методу) — область видимості змінної обмежена лише сусіднім об'єктом.
- “global” (глобальна змінна) — область видимості поширюється на всю діаграму кооперації.
- “self” (рефлексивний зв'язок об'єкта) — об'єкт пов'язаний сам із собою. Такий зв'язок припускає, що об'єкт може передавати повідомлення самому собі. Графічно, цей зв'язок зображується петлею, розташованою у верхній частині об'єкта.



На наведеному зображенні (рис. 28) показаний приклад використання зв'язків у діаграмах кооперацій.

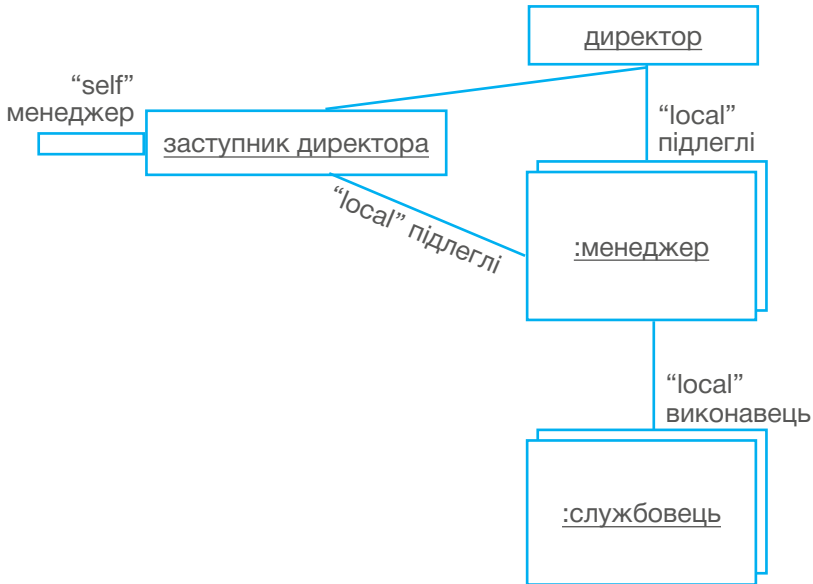


Рисунок 28.

### 2.2.6. Повідомлення





Ми вже розглядали повідомлення при вивченні діаграм послідовності. Проте, при побудові діаграм кооперації вони мають декілька додаткових особливостей.

На діаграмі кооперації повідомлення вказують, що певний об'єкт передає іншому об'єкту якусь інформацію. Тобто, за допомогою повідомлень ми вказуємо особливості комунікації об'єктів нашої моделі. При цьому передбачається, що після отримання повідомлення об'єкт виконає певну дію (це повелося називати стимулюючим змістом повідомлень).

Зв'язок у такому контексті слугує каналом передачі повідомлень від об'єкта-джерела (*source-object*) до об'єкта-одержувача, який також називають об'єктом призначення (*destination-object*).

Графічно, повідомлення зображуються поміченими стрілками поруч із зв'язком (вище або нижче лінії зв'язку). Напрямок стрілки вказує від об'єкта-відправника до об'єкта-одержувача.

Існує чотири види стрілок для позначення повідомлень:

1.  (суцільна лінія із трикутною стрілкою) — використовується тоді, коли повідомлення ініціює виклик процедури або іншого потоку виконання. Як правило, усі повідомлення такого типу синхронні.
2.  (суцільна лінія з V-подібною стрілкою) — порожній потік управління; зображує один окремий «крок» у послідовності потоку управління. Найчастіше, усі такі повідомлення асинхронні.
3.  (суцільна лінія з напівстрілкою) — використовується при ініціюванні асинхронного потоку управління. Повідомлення такого типу відправляється в довільний момент часу (який, спочатку, не відомий) і найчастіше — активними об'єктами. Зазвичай, повідомлення цього типу ініціюються «акторами» і є стартовими у всьому потоці управління.
4.  (пунктирна лінія з V-подібною стрілкою) — використовується для позначення повернення із виклику процедури. Як правило, повідомлення такого типу відсутні на діаграмах взагалі, оскільки за умовчанням передбачається наявність такого повідомлення після закінчення будь-якої процедури.

### 2.2.7. Формат запису повідомлень

Повідомлення можуть мати підпис у вигляді рядка тексту у такому форматі:

```
попередні_повідомлення [сторожова_умова]
вираз_послідовності
повернене_значення назва_повідомлення список_аргументів
```

- Попередні повідомлення мають вигляд списку з номерами повідомлень, розділених комами і зазначених перед слешем — «/».
- Список попередніх повідомлень вказується з метою специфікувати, що це повідомлення не може бути передано доки не будуть виконані повідомлення, зазначені у списку.
- Цей вираз може і не вказуватися. Але, у такому разі, його не вказують і разом із роздільною рисою.
- Сторожова умова — логічний вираз, який вказують у квадратних дужках; призначений для того, щоб синхронізувати окремі потоки управління. Цей вираз може і не вказуватися.
- Вираз послідовності — це перелік термінів (англ. *term*) послідовності, розділений точками, після якого вказана двокрапка.
- Кожен із термінів є окремим рівнем вкладеності послідовності процедур у формі закінченої ітерації. Перший термін відповідає початку процедурної послідовності.

Нижче наведено форму запису кожного окремого терміну:

```
[ціле_число|назва] [символ_рекурентності]
```

Ціле число визначає порядковий номер терміну у процедурній послідовності. Назва використовується для того, щоб вказати паралельні потоки управління (повідомлення, які відрізняються назвою — паралельні на даному рівні вкладеності). А символ рекурентності використовується для вказівки умовного або ітеративного виконання.

- Повернене значення — список назв значень, які повертаються після закінчення комунікації або взаємодії в повній ітерації зазначеної послідовності процедур.
- Назва повідомлення — це назва події, яке ініціюється в об'єкті одержувача після того, як він його прийняв. Найчастіше, для назви використовують назву операції об'єкта-одержувача.
- Список аргументів — список дійсних параметрів викликаної операції, розділеною комами і взяту у дужки.

### **2.3. Практичні приклади побудови діаграм кооперації**

У якості прикладу складання діаграм кооперації, наведемо складання загальної структури взаємодії розподіленого додатку, який виконує віддалене адміністрування робочих станцій. В якості панелі адміністратора виступає додаток, наділений інтерфейсом адміністрування системи з можливістю встановлення завдання до виконання будь-якому зареєстрованому клієнтському комп'ютеру.

Після встановлення, завдання надходять на сервер, який, залежно від того, коли робоча станція буде доступною, надсилає до клієнтського додатка повідомлення про те, яке завдання потрібно виконати.

Після виконання завдання, клієнтська програма звітує серверу, а він, у свою чергу, сигналізує панелі адміністратора про подачу зміненої інформації.

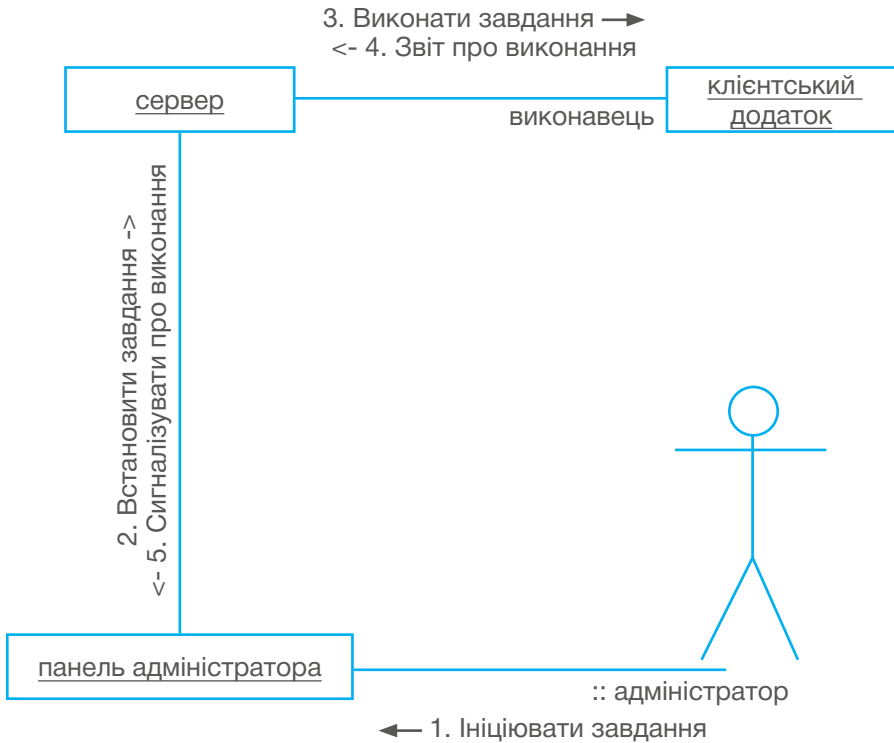


Рисунок 29.

## 3. Діаграма компонентів

Діаграма компонентів використовується для відображення компонентів додатка. Побудувавши таку діаграму, ви зможете графічно відобразити всю структуру додатка, починаючи від об'єктів і класів, закінчуючи виконуваними файлами і бібліотеками.

Діаграми компонентів складаються з компонентів, інтерфейсів і зв'язків між ними.

### 3.1. Цілі даного типу діаграм

Багаторазове використання коду — коли будується діаграма компонентів, досить легко помітити схожі частини (компоненти) додатку. Тому, варто задуматися, чи є можливість із двох компонентів зробити один універсальний? Діаграма компонентів допоможе графічно зобразити багаторазове використання коду.

Візуалізація загальної структури програми — діаграма компонентів дуже добре відображає взаємозв'язок об'єктів. Це допоможе програмістам детальніше розібратися зі структурою об'єктів і їх взаємозв'язком.

Уявлення структури даних — за допомогою діаграми компонентів можна представити схему зберігання даних або схему роботи будь-якої програми.

### 3.2. Базові поняття

Як ми вже згадували, діаграми компонентів складаються з трьох базових явищ:

- **Компоненти** — основний будівельний блок у діаграмі компонентів, за допомогою яких позначаються об'єкти класів, класи, функціональні точки, екземпляри програм.
- **Інтерфейси** — діаграма відображає інтерфейси доступу до компонентів. Також на діаграмі можна визначити ситуацію, коли для використання об'єкта вам необхідний клас, який підтримує певний інтерфейс.
- **Залежності** — за допомогою залежностей можна показати, що один об'єкт використовує інший. Залежності іноді називають зв'язком.

### 3.2.1. Компонент

Компонент відображається на діаграмі компонентів як прямокутник, в якому зліва знаходяться два прямокутники:

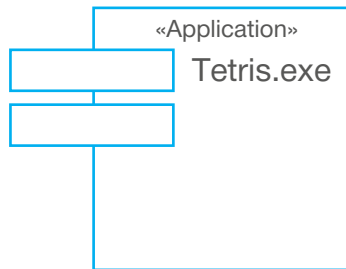


Рисунок 30.

Даний вид компонента не є стандартним. Часто компонент помічають у вигляді простого прямокутника, як правило, додатку, в якому створюються UML діаграми, на компоненті малюють таку фігуру для позначення, що даний прямокутник є компонентом.

Для позначення компонентів варто використовувати Microsoft Visual Studio 2010 (рис. 31):



Рисунок 31.

Як ви помітили, у правому верхньому куті прямокутника зображена фігура, яка вказує на даний прямокутник, який є компонентом. На компоненті, як правило, пишуть два записи: тип об'єкта і його назва. Іноді, замість типу компонента нічого не пишуть або просто зазначають «component». Тип об'єкта можна ігнорувати, оскільки є стандарт написання назв, які мають на увазі тип об'єкта:

- **Скомпільований код** — коли компонент виступає, як готовий додаток, поряд з його назвою вказують розширення використовуваного файлу, наприклад: Tetris.exe, або Tetris.jar. Такий підхід використовується і до бібліотек, наприклад: MessageWindows.dll
- **Файли** — коли програма використовує зовнішнє сховище даних, наприклад XML або TXT, назва для такого компонента дається із розширенням файлу. Наприклад: Settings.xml
- **Сторінки** — розширення сторінок на діаграмі компонентів теж необхідно вказувати. Наприклад: Index.php



- **Класи** — коли йдеться про класи, можна писати просто назву класу, але іноді пишуть слово «клас» перед назвою класу, наприклад: `class Program`
- **Об'єкти класів** — об'єкти називають наступним чином: спочатку пишеться назва об'єкта, далі пишуть двокрапку і тип даних цього об'єкта. Наприклад: `LWin:LoginWindow`

Рекомендується вказувати стереотип об'єкта. Незважаючи на те, що його іноді ігнорують, це внесе ясність у вашу діаграму.

Компоненти можуть бути зображені на діаграмі окремо, а можуть розташовуватися один в одному (рис. 32).

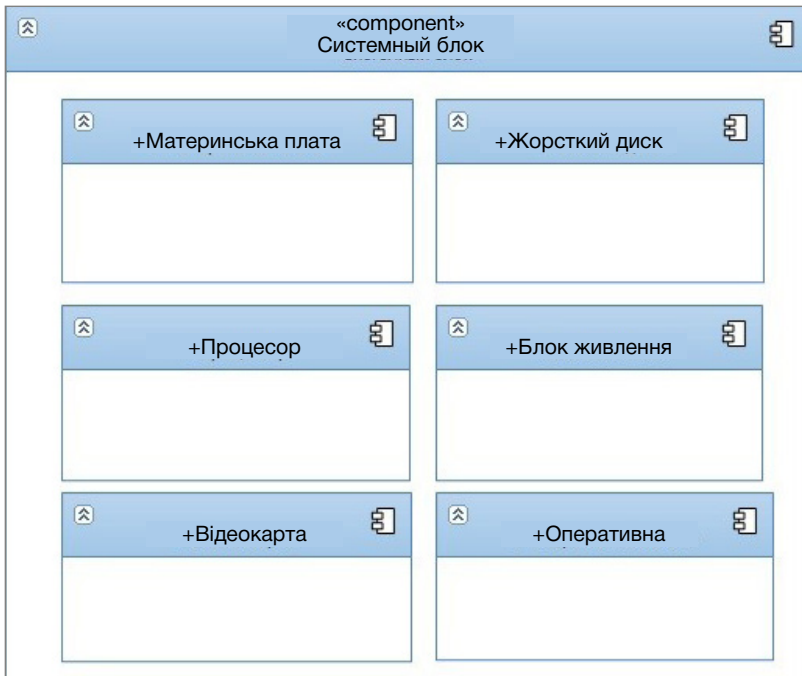


Рисунок 32.

Наприклад, можна сказати що є загальне поняття системного блоку комп'ютера; це компонент робочої станції, але всередині системного блоку знаходяться інші компоненти: жорсткий диск, материнська плата, процесор, блок живлення.

Але ця діаграма має незакінчений вид, тому що на ній відсутні зв'язки та інтерфейси.

Слід зазначити, що вкладати один компонент в інший можна тільки один раз. Інакше кажучи, я не можу створити «на словах» компонент системного блоку, в якому буде знаходитися компонент оперативної пам'яті, всередині якої будуть інші компоненти (чіпи та ін.). Таке обмеження пов'язане зі складнощами при читанні таких діаграм.

Дизайнер в MS Visual Studio 2010 дозволяє змінювати основний колір компонента. Така властивість має суто естетичний характер.

### 3.2.2. Залежності

Залежність — це дуже простий спосіб показати на діаграмі, що один компонент використовує інший компонент. Зазвичай, залежність позначається пунктирною стрілкою від залежного об'єкта до незалежного. Наприклад: материнська плата в комп'ютері залежить від відеокарти, а відеокарта — незалежний компонент. Тому, стрілка має вказуватися від материнської плати до відеокарти.

На діаграмах допускається ситуація, коли один компонент залежить від двох або більше інших компонентів. Допускається взаємозалежність компонентів, коли обидва компоненти залежні один від одного.

Лінія залежності не показується, якщо незалежний компонент є частиною залежного компонента:

Часто дизайнери діаграм підписують залежності для внесення більш чіткого розуміння діаграми.

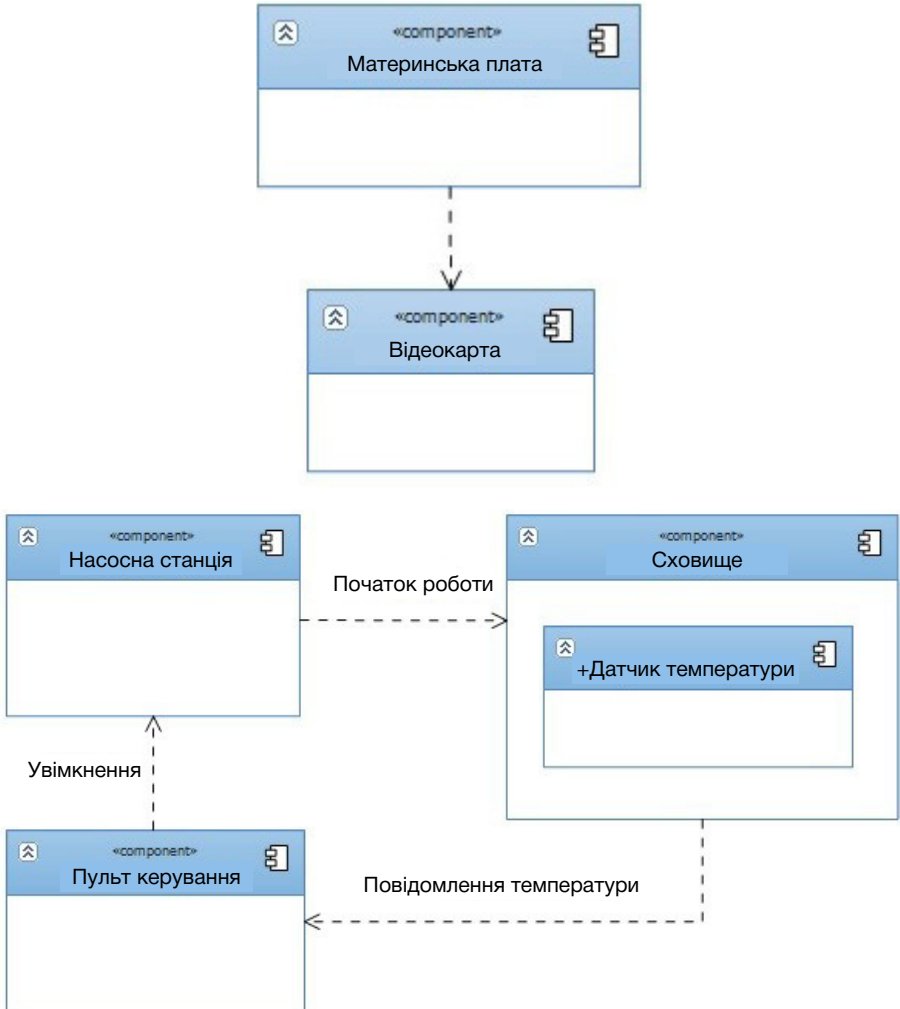


Рисунок 33.

### 3.2.3. Інтерфейс

Інтерфейси — важлива частина діаграми компонентів. Часто буває, що для доступу до певного компонента потрібен інтерфейс.

Наприклад: бібліотека для MP3 медіа-файлів надає інтерфейс, який показує її налаштування (інформацію про методи). Відображається ця ситуація наступним чином:

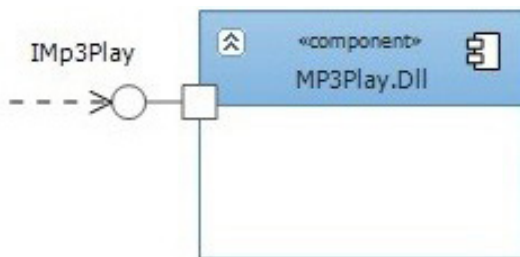


Рисунок 34.

Другий випадок, коли об'єкт для своєї роботи вимагає об'єкт, що підтримує інтерфейс. Наприклад: компонент відображення на екрані 3D об'єктів вимагає для своєї роботи компонент з підтримкою інтерфейсу “I3dDraw”. Тоді діаграма буде мати вигляд:

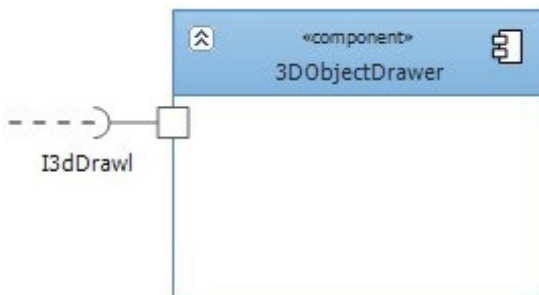


Рисунок 35.

Найчастіше зустрічається поєднання цих двох способів опису інтерфейсу:

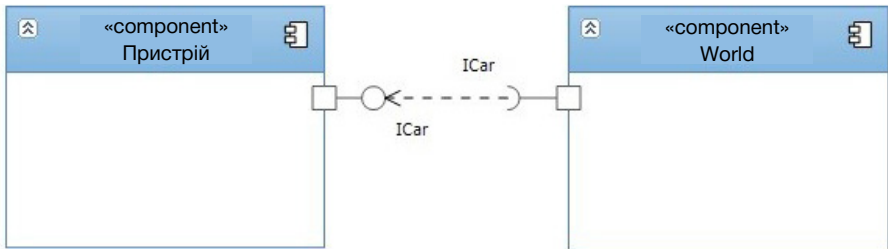


Рисунок 36.

Діаграма показує, що компонентіві «World» потрібен для роботи компонент з підтримкою інтерфейсу «ICar». Компонент «Car» підтримує інтерфейс «ICar», що дозволяє компонентіві «World» використовувати компонент «Car».

Більш наочним прикладом може бути комп'ютер і USB накопичувальний пристрій:

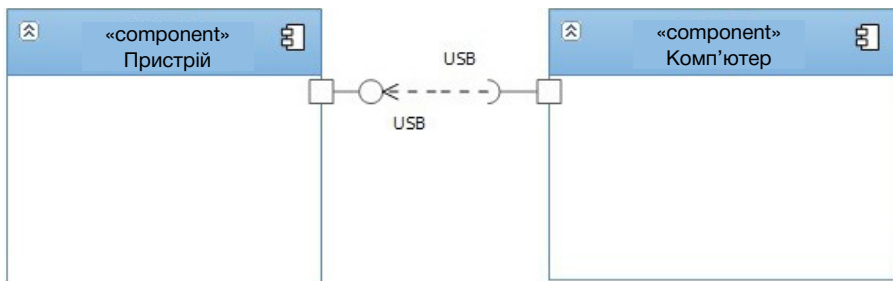


Рисунок 37.

Зверніть увагу на залежності. Коли об'єкт для своєї роботи вимагає клас, який підтримує певний інтерфейс, це означає, що він залежить від класу, який підтримує інтерфейс.

### 3.3. Практичні приклади побудови діаграм компонентів

Для прикладу, розглянемо додаток: Медіа програвач. Ознайомимосся з усіма компонентами цього додатку:

- **MainWindow** — інтерфейс програми користувача, і вся її основна функціональність, цей клас використовує об'єкти, успадкованими від інтерфейсу «UseAudioDll». Також, цей компонент використовує клас «TrackCollection» і вміє керувати ним.
- **TrackCollection** — колекція, яка зберігає об'єкти, які підтримують інтерфейс «ITrack». На діаграмі видно, що даний об'єкт потребує об'єктів, які підтримують цей інтерфейс.
- **Track** — клас, який містить інформацію про медіа трек; об'єкти даного класу підтримують відразу три інтерфейси: «ITrack», «IMp3File», «IOggFile».
- **Mp3Play.Dll** — бібліотека для відтворення файлів формату MP3. Для керування такою бібліотекою, надається доступ через інтерфейс UseAudioDll. Але для коректної роботи, бібліотеці потрібен компонент, підтримуючий інтерфейс «IMp3File».
- **OggPlay.Dll** — бібліотека має подібні функції до Mp3Play.Dll, але для її роботи потрібен компонент, який працює за допомогою інтерфейсу IOggFile.

Можливо, вам здається, що цю діаграму краще було б зобразити у вигляді діаграми класів, але це не так.

Діаграма кооперацій краще показує, як відбувається взаємодія між компонентами додатку (рис. 38).

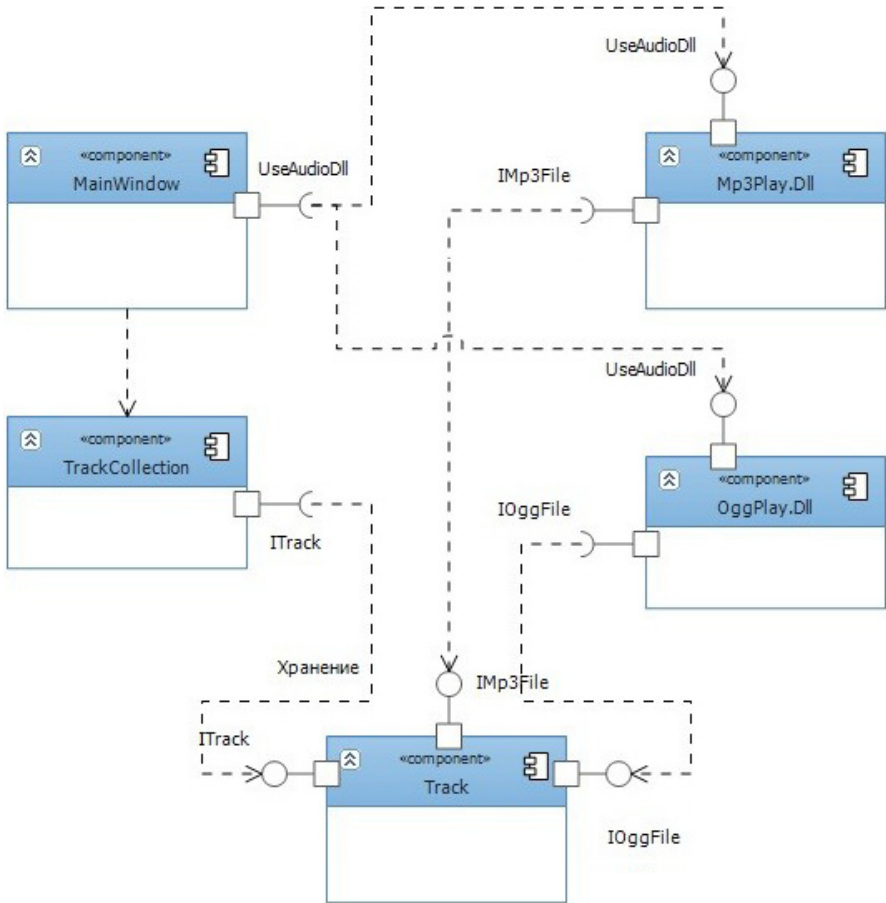


Рисунок 38.

Далі ми розглянемо діаграму компонентів для корпоративного сайту (рис. 39 см. на наступній стр.).

Коли будується діаграма компонентів для сайту, варто багато уваги приділяти назвам залежностей, за допомогою яких можна вказати, яким способом з однієї сторінки на іншу будуть передаватися дані, або у який спосіб користувач переходитиме між ними.

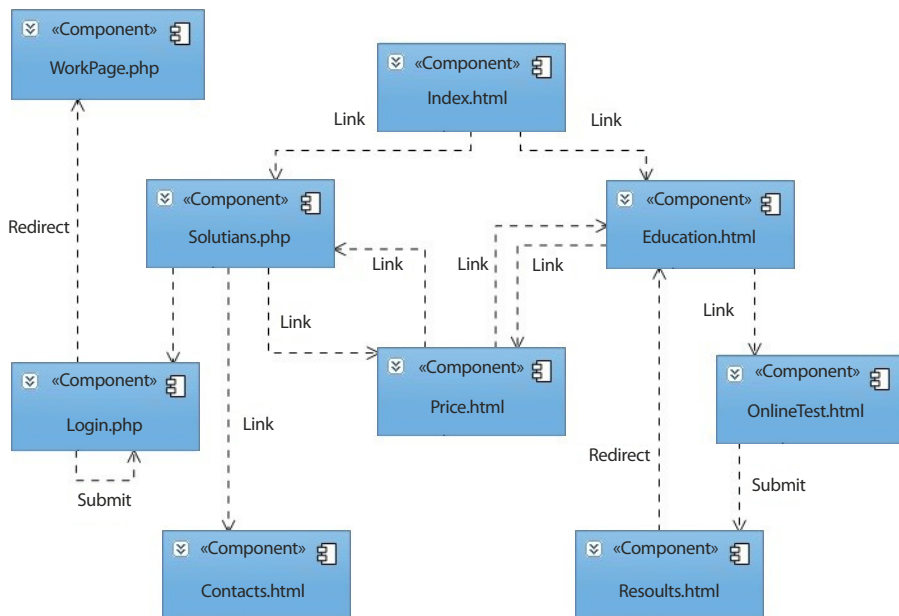


Рисунок 39.



## 4. Діаграма розгорткування

Діаграма розгорткування — використовується для фізичного уявлення системи, яка розгортається. Простіше кажучи, діаграма розгорткувань показує, які фізичні пристрої та зв'язки між ними необхідні для коректної роботи додатку, що розробляється.

Варто зазначити, що діаграма розгорткування іноді замінюється на діаграму компонентів.

### 4.1. Цілі даного типу діаграм

При складанні діаграми розгорткування можна знайти слабкі або незахищені місця в системі. Наприклад: створивши діаграму розгорткування для онлайн-ігри, стає зрозумілим, що сервер буде публічним, має бути добре захищеним від атак хакерів і мати достатню кількість ресурсів для обслуговування великої кількості клієнтів.

Побудувавши діаграму розгорткування, можна сказати, які пристрої будуть необхідними для роботи додатків. Це дозволить більш конкретно зрозуміти суть продукту, що розробляється.

### 4.2. Базові поняття

Діаграма розгорткування складається з двох базових понять: «Вузли» та «З'єднання».

Вузол на діаграмі розгорткування — це якась обчислювальна одиниця, як правило, це пристрій з процесором та (або) електронною, магнітною пам'яттю. Вузлом

можна вважати все, починаючи від супер комп'ютера до USB-накопичувача. Вузол зображується на діаграмі розгортвання у вигляді кубу:

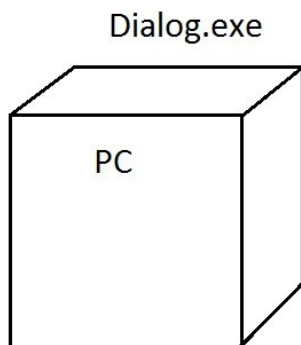


Рисунок 40.

Так можна позначити будь-який вузол. Необхідно написати на кубі тип даного пристрою (можливо, додаткову інформацію, наприклад частоту процесора). Зверху над кубом, зазвичай, відображається назва компонента програми, яка працює на цьому пристрої. Іноді для позначення «простих» пристроїв (наприклад, принтер) використовується куб із чорними краями:

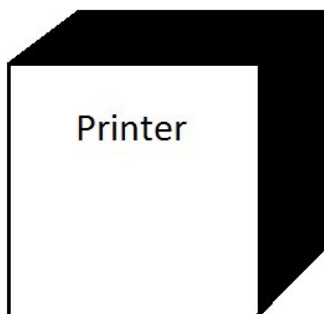


Рисунок 41.

Також можна з'єднати діаграму компонентів з діаграмою розгортання, просто відобразивши компоненти всередині пристрою, наприклад, так:

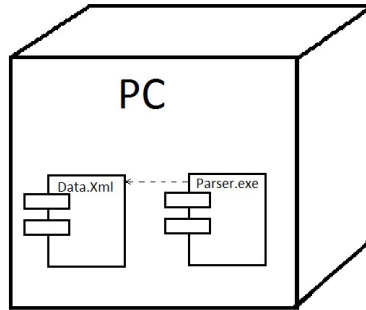


Рисунок 42.

Для побудови діаграм розгортання можна використовувати Microsoft Visio, завдяки якій можна зображати вузли та з'єднання.

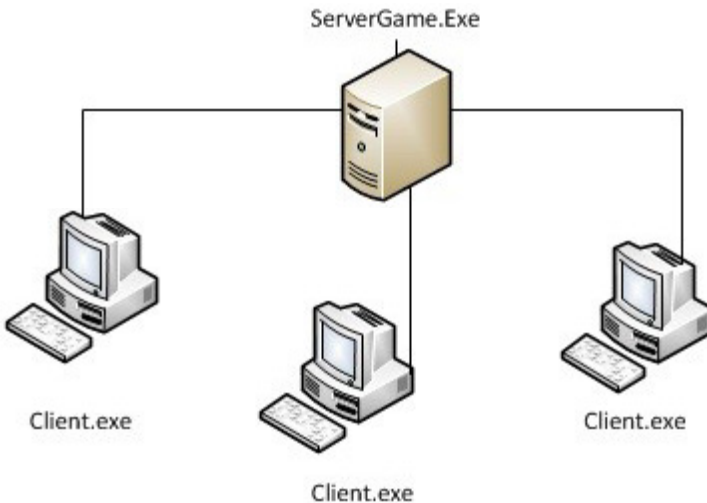


Рисунок 43.

Вузлом також є і людина, яка графічно зображує бізнес-процеси. З'єднання на діаграмі розгортання зображуються простими лініями без стрілок, які, як правило, вказують на те, що даний об'єкт має відношення до об'єкта, до якого веде лінія.

На діаграмі (рис. 44) показано, що принтер пов'язаний лише з одним комп'ютером, тому що лінія веде від принтера до одного з комп'ютерів. Допускається підписування ліній для більш зрозумілішого уявлення діаграми.

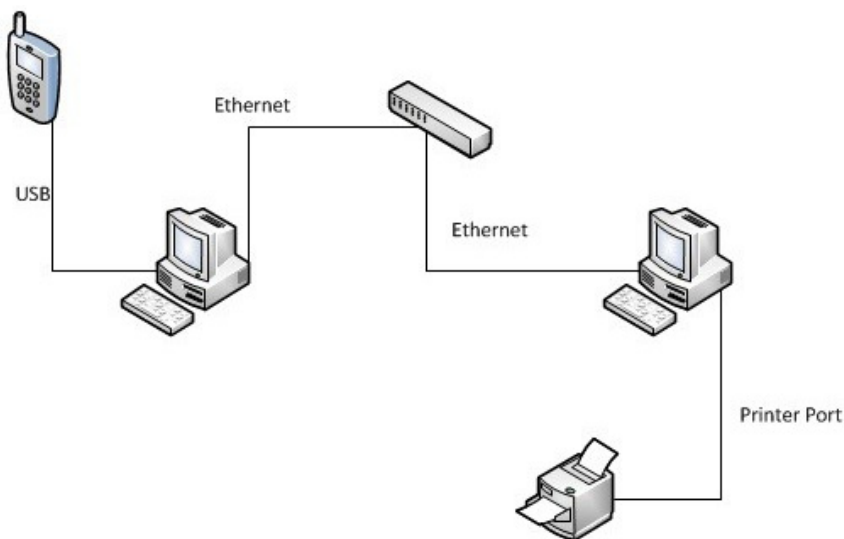


Рисунок 44.

### 4.3. Практичні приклади побудови діаграм розгортання

Розглянемо додаток керування системою «Розумний дім». Розумний дім — житлове приміщення, обладнане високотехнологічними пристроями, що допомагають

користувачеві в господарстві. Яскравим прикладом такого пристрою може бути холодильник, який стежить за кількістю продуктів і, при необхідності, відправляє господареві повідомлення про те, які продукти зіпсувалися і їх потрібно замінити.

Далі представлена діаграма розгортання такої системи:

Електронний замок

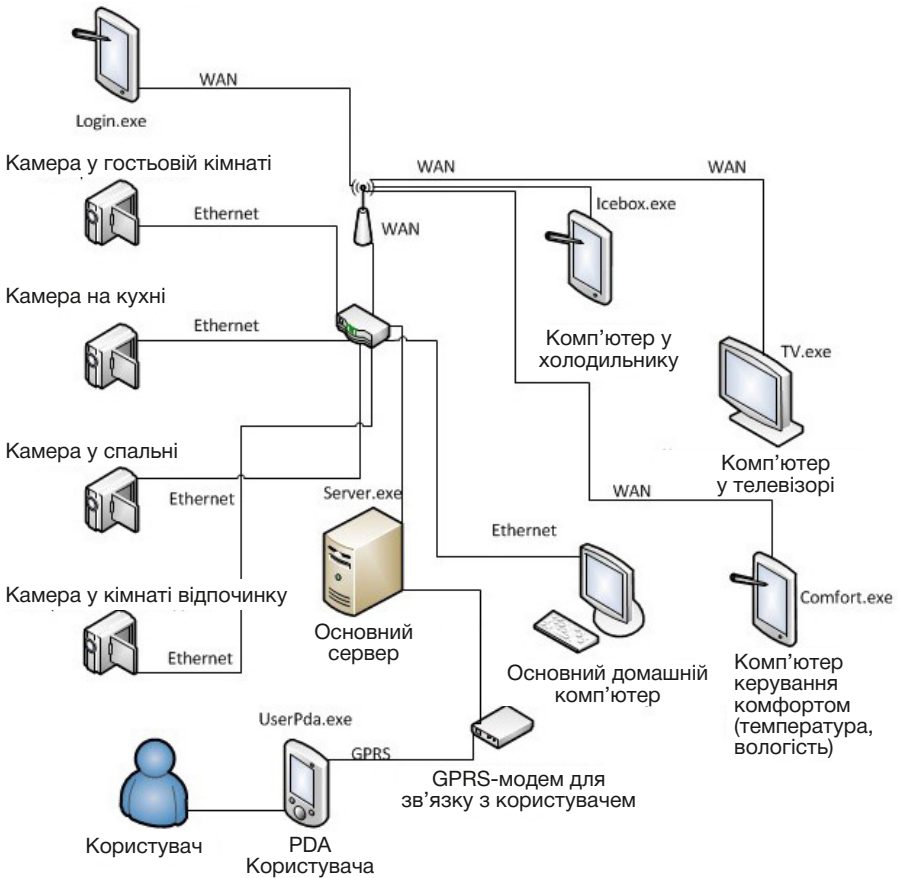


Рисунок 45.

Далі йде перелік додатків з коротким описом до них:

- **Server.exe** — основний додаток, встановлений на вузлі «основний сервер», займається збором і зберіганням інформації від кожного пристрою, пов'язаний з GPRS-модемом і роутером, який транслює.
- **Login.exe** — додаток «Електронний замок» просить користувача ввести пароль, зв'язується з WiFi точкою, яка зв'язується з роутером, який транслює запит на сервер. Якщо сервер відправляє відповідь, що пароль введено вірно, електронний замок відкриє двері.
- **Icebox.exe** — додаток, що керує холодильником, зв'язується з сервером через WiFi, який пускає сигнал транзитом через роутер до основного серверу.
- **Tv.exe** — додаток, що керує телевізором; з'єднується з сервером так само, як і комп'ютер в холодильнику.
- **Comfort.exe** — додаток для налаштування клімату в будинку, який з'єднується з сервером так само, як і комп'ютер в холодильнику.
- **UserPda.exe** — додаток встановлено в портативному комп'ютері у користувача.

Проаналізувавши діаграму, можна дійти невтішного висновку, що роутер біля входу на сервер досить перевантажений, оскільки передаватиметься потокове відео з камер спостереження. Можливо, варто додати комп'ютер, який буде зайнятий зберіганням записів з камер. Також, мережа має WiFi доступ, тому необхідно створити захист для додатків. Це не дозволить зловмиснику поміняти користувачеві температуру в будинку без відома власника.

Наступний додаток, який ми розглянемо, це додаток для контролю часу гри гравців у комп'ютерному клубі.

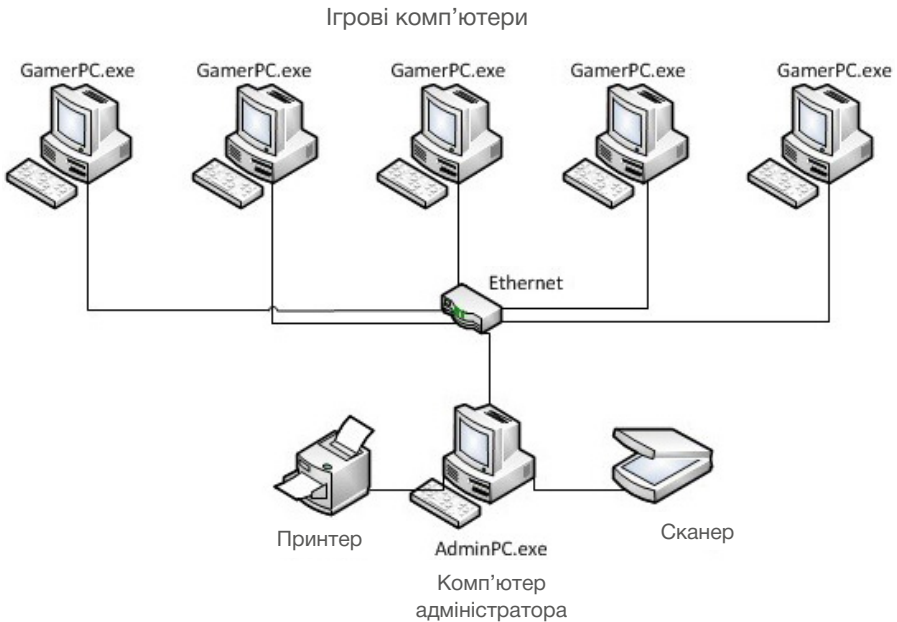


Рисунок 46.

У цьому продукті для комп'ютерних клубів буде міститися лише два продукти:

- **AdminPC.exe** — додаток, встановлений на комп'ютері адміністратора, оприділяє кількість часу до вимкнення користувацьких комп'ютерів.
- **GamerPC.exe** — додаток дозволяє/не дозволяє користувачеві грати; рішення приймається від того, чи сплатив користувач за відведений час гри.

Проаналізувавши схему, стає зрозумілим, що комп'ютер адміністратора буде найбільш схильний до атак хакерів, оскільки саме з нього нараховується час для користувачів. .

Комп'ютер адміністратора має бути стійким проти вірусів.



## Урок №4

Діаграма послідовності, діаграма кооперації, діаграма компонентів і діаграма розгортання

© STEP IT Academy, [www.itstep.org](http://www.itstep.org)

Усі права на фото-, аудіо- і відеотвори, що охороняються авторським правом і фрагменти яких використані в матеріалі, належать їх законним власникам. Фрагменти творів використовуються в ілюстративних цілях в обсязі, виправданому поставленим завданням, у рамках учбового процесу і в учбових цілях, відповідно до законодавства про вільне використання твору без згоди його автора (або іншої особи, яка має авторське право на цей твір). Обсяг і спосіб цитованих творів відповідає прийнятим нормам, не завдає збитку нормальному використанню об'єктів авторського права і не обмежує законні інтереси автора і правовласників. Цитовані фрагменти творів на момент використання не можуть бути замінені альтернативними аналогами, що не охороняються авторським правом, і відповідають критеріям добросовісного використання і чесного використання.

Усі права захищені. Повне або часткове копіювання матеріалів заборонене. Узгодження використання творів або їх фрагментів здійснюється з авторами і правовласниками. Погоджене використання матеріалів можливе тільки якщо вказано джерело.

Відповідальність за несанкціоноване копіювання і комерційне використання матеріалів визначається чинним законодавством.