МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

**О.І. Марченко**
**І.І.Вітковська**

# Компоненти програмної інженерії

## Частина 1. Вступ до програмної інженерії

## Лабораторний практикум

Електронне мережеве видання
*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського*
*як навчальний посібник для здобувачів ступеня бакалавра*
*за спеціальністю 121 «Інженерія програмного забезпечення»*

Київ
КПІ ім. Ігоря Сікорського
2022

Компоненти програмної інженерії. Частина 1. Вступ до програмної інженерії: Лабораторний практикум [Електронний ресурс]: навч. посіб. для студ. освітньої програми «Інженерія програмного забезпечення інформаційних систем» спеціальності 121 «Інженерія прогрманого забезпечення» / Марченко О.І., Вітковська І.І. – Електронні текстові дані (1 файл: 4,5 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2022. – 50 с.

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського*

*(протокол №4 від 19.01.2023 р.) за поданням Вченої ради факультету Інформатики та обчислювальної техніки (протокол №6 від 28.11.2022 р.)*

Електронне мережне навчальне видання

Марченко Олена Іванівна старший викладач

Вітковська Ірина Іванівна старший викладач

Відповідальний редактор:

Рецензент: Волокита А. М., к.т.н., доцент, КПІ ім. Ігоря Сікорського, факультет інформатики та обчислювальної техніки, кафедра обчислювальної техніки

Навчальний посібник охоплює теоретичний матеріал та практичні завдання, які необхідні для виконання лабораторних робіт з дисципліни «Компоненти програмної інженерії Частина 1. Вступ до програмної інженерії» для студентів, які навчаються за спеціальністю 121 «Інженерія програмного забезпечення».

Навчальний посібник може бути корисним студентам відповідних спеціальностей при вивченні дисциплін, пов’язаних із аналізом, проектуванням та моделюванням в розробці програмного забезпечення, а також при виконанні бакалаврських робіт, курсових проектів, в яких використовується відповідна специфікація.

Реєстр. №П 22/23-320. Обсяг авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua

# ЗМІСТ

# ВСТУП

Дисципліна “Компоненти програмної інженерії. Частина 1. Вступ до програмної інженерії” вивчається на етапі підготовки фахівців освітньо-кваліфікаційного рівня «бакалавр» має на меті ознайомлення студентів з сучасними методами та засобами, які застосовуються у процесах життєвого циклу програмного забезпечення, ознайомлення з практичними навиками накопиченими у галузі програмної інженерії; вивчення шаблонного підходу до проектування програмних систем та мови UML; вивчення методів та засобів розробки програмного забезпечення; вивчення об’єктно-орієнтованого підходу до розробки програмного забезпечення.

Завдання вивчення дисципліни обумовлене вимогами підготовки бакалаврів відповідно до кваліфікаційної характеристики зі спеціальності 121 «Інженерія програмного забезпечення».

По закінченню вивчення дисципліни студент набуває такі навики та вміння:

- володіння основною термінологією дисципліни, вміння пояснити зміст базових понять і розділів інженерії програмного забезпечення;
- володіти навиками визначення етапів, методів та стандартів розробки програмного забезпечення;
- знання принципів використання базових шаблонів проектування програмного забезпечення та принципів їх побудови та застосування,
- пошук та визначення джерела вимог до програмного забезпечення;
- аналіз та розробка специфікації вимог користувачів;
- знання теоретичних та практичних основ моделювання систем програмного забезпечення;
- проектувати компоненти архітектурного рішення.

**КОМПЕТЕНТНОСТІ**

Загальні компетентності:

ЗК-1. Здатність до абстрактного мислення, аналізу та синтезу.

ЗК-2. Здатність застосовувати знання у практичних ситуаціях

ЗК-6. Здатність до пошуку, оброблення та аналізу інформації з різних джерел.

Спеціальні (фахові, предметні) компетентності

ФК-1. Здатність ідентифікувати, класифікувати та формулювати вимоги до програмного забезпечення.

ФК-2. Здатність брати участь у проектуванні програмного забезпечення, включаючи проведення моделювання (формальний опис) його структури, поведінки та процесів функціонування.

ФК-5. Здатність дотримуватися специфікацій, стандартів, правил і рекомендацій в професійній галузі при реалізації процесів життєвого циклу

ФК-8. Здатність застосовувати фундаментальні і міждисциплінарні знання для успішного розв’язання завдань інженерії програмного забезпечення.

ФК-13. Здатність проводити обчислювальні експерименти, порівнювати результати експериментальних даних і отриманих рішень.

**ПРОГРАМНІ РЕЗУЛЬТАТИ НАВЧАННЯ**

ПРН-1. Аналізувати, цілеспрямовано шукати і вибирати необхідні для вирішення професійних завдань інформаційно-довідникові ресурси і знання з урахуванням сучасних досягнень науки і техніки.

ПРН-3. Знати основні процеси, фази та ітерації життєвого циклу програмного забезпечення.

ПРН-10. Проводити передпроектне обстеження предметної області, системний аналіз об'єкта проектування.

ПРН-11. Вибирати вихідні дані для проектування, керуючись формальними методами опису вимог та моделювання.

ПРН-12. Застосовувати на практиці ефективні підходи щодо проектування програмного забезпечення.

**ПРЕРЕКВІЗИТИ ТА ПОСТРЕКВІЗИТИ ДИСЦИПЛІНИ (МІСЦЕ СТРУКТУРНО-ЛОГІЧНІЙ СХЕМІ НАВЧАННЯ ЗА ВІДПОВІДНОЮ ОСВІТНЬОЮ ПРОГРАМОЮ)**

Пререквізити:

Основи програмування

Постреквізити:

Компоненти програмної інженерії. Частина 2. Моделювання та аналіз вимог до програмного забезпечення

Навчальний посібник призначений для студентів спеціальності 121 «Інженерія програмного забезпечення» кафедри інформатики та програмної інженерії всіх освітніх програм та всіх форм навчання.

У посібнику наведено систематизоване викладення основних відомостей з використання сучасного інженерного підходу щодо розробки програмного забезпечення шляхом розглядання складових фаз аналізу та проектування; ознайомлення з практичними навиками у цій галузі.

# ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ

Посібник призначений для виконання лабораторних робіт з дисципліни “Компоненти програмної інженерії. Частина 1. Вступ до програмної інженерії”. Він складається з п’яти тем. Для кожної з тем викладений систематизований теоретичний матеріал, наведені приклади виконання діаграм, завдання, контрольні питання та рекомендована література.

В процесі виконання лабораторних робіт студенти вивчають UML (англ. Unified Modeling Language – уніфікована мова моделювання) – мову графічного опису для об'єктного моделювання в галузі розробки програмного забезпечення, для моделювання бізнес-процесів, системного проектування та відображення організаційних структур.

UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, званої UML-моделлю. UML було створено для визначення, візуалізації, проектування та введення документації, в основному програмних систем. UML не є мовою програмування, але на підставі моделей UML можлива генерація коду. Актуальна версія стандарту 2.5.1.

Перед початком виконання студентам слід ознайомитися з теоретичною частиною роботи. Після цього кожен студент отримує індивідуальне завдання до виконання. Лабораторні роботи слід виконувати послідовно, оскільки роботи побудовані у логічній послідовності вивчення функціональних характеристик системи що розглядається; її структурного аналізу; дослідження поведінки і закінчується визначенням загальної схеми архітектури системи.

Загальні рекомендації до виконання лабораторного практикуму:

– кожна лабораторна робота виконується та оформлюється у вигляді окремого документу з назвою ПрізвищеІніціали_ГрупаАА_ЛРВ_ВаріантСС, де АА – номер групи, В – номер лабораторної роботи, СС – номер варіанту;

– кожна лабораторна робота починається з титульного аркушу, на якому вказують тему лабораторної роботи, номер варіанту, прізвище та ініціали виконавця (приклад Додаток А);

– кожна лабораторна робота повинна містити зміст;

– наводиться опис варіанту завдання;

– діаграми, схеми та інший графічний матеріал лабораторних робіт виконати за допомогою спеціалізованого програмного забезпечення (Rational Rose, PowerDesigner, Enterprise Architect, Microsoft Visio) або веб-застоснку для побудови діаграм і схем (Draw.io);

– до усіх діаграм обов’язкове обґрунтування використання визначеного типу діаграм для завдання лабораторної роботи.

# ЛАБОРАТОРНА РОБОТА №1

# Встановлення вимог до функціональності ПЗ засобами мови UML

Мета роботи: отримати навички специфікування вимог до ПЗ

## Короткі теоретичні відомості

Діаграми варіантів використання або прецедентів (Use Case Diagram), відноситься до класу поведінкових діаграм та застосовуються для відображення функціональності програмного забезпечення, границь та контексту предметної області, для якої призначено програмне забезпечення та взаємодії програмного забезпечення із зовнішнім середовищем (зокрема користувачами).

Графічними компонентами діаграми варіантів використання є дійові особи (актори), варіанти використання та зв’язки між ними.

Діаграми варіантів використання супроводжуються вербальними описами (специфікаціями). Тобто кожний прецедент може бути описаний за допомогою фіксованого потоку подій.

Діаграма будується за допомогою графічних елементів, які наведені у таблиці 1.

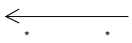
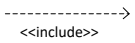
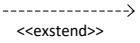
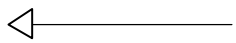
Побудова діаграми прецедентів відбувається у такій послідовності:

– визначення діючих осіб (actors);

– визначення варіантів використання системи, виходячи з потреб діючих осіб;

– встановлення зв'язків між суб'єктами та варіантами використання. Зв'язки включення та розширення обов'язково підписуються позначеннями «include» та «extend».

Таблиця 1 – Основні графічні елементи діаграми діяльності

| Класифікатор | Функція класифікатору | Нотація |
|---|---|---|
| Суб'єкт (Actor) | Користувач системи |   |
| Варіант використання (Use case) | Специфікація поведінки системи чи її частини при взаємодії з суб'єктами |   |
| Комунікативний зв'язок (асоціація) | Взаємодія суб'єкта з системою |  |
| Включення «include» | Варіант використання включає в себе обов'язкові варіанти |   |
| Разширення «extend» | Варіант розширюється деякими необов'язковими варіантами |   |
| Успадкування (узагальнення) | Суб'єкт чи варіант використання успадковує структуру та поведінку предка, визначаючи при цьому власні властивості та поведінку |  |

Для прикладу розглянемо модель прецедентів Банкомат.

**Опис предметної області**

Банкомат – це автомат для видачі готівки за кредитними пластиковими картками. До його складу входять такі пристрої: дисплей, панель управління з кнопками, приймач кредитних карток, сховище готівки та лоток для видачі, принтер для друку довідок.

Банкомат зв’язаний з комп’ютером банку, де зберігаються відомості про рахунки клієнтів. Обслуговування клієнта розпочинається з моменту розміщення пластикової картки в банкомат. Після розпізнання типу пластикової картки банкомат видає на дисплей запрошення ввести персональний код – чотиризначне число (ПІН код). Банкомат перевіряє

правильність введеного коду і дозволяє клієнту вибрати операцію або зняття готівки, або перевірки залишку рахунка.

Якщо клієнт обирає зняття готівки, він повинен вказати суму (10, 50, 100, 200, 500, 1000 грн.). На дисплеї формується повідомлення про друк довідки, а банкомат посилає запит головному комп'ютерові банку. Якщо дозвіл на операцію отримано, банкомат перевіряє наявність коштів у своєму сховищі. На дисплей виводиться повідомлення «Вийміть картку». Після виймання картки банкомат видає вказану суму готівки до лотка. Банкомат друкує довідку про операцію, якщо вона була затребувана клієнтом. Коли ж клієнт бажає отримати інформацію про залишок на рахунку, він обирає відповідний пункт меню, банкомат посилає запит до головного комп'ютера банку і виводить суму на екран, або друкує довідку за бажанням клієнта.

**Глосарій**

Банкомат – термінал, який дає можливість клієнту здійснити транзакцію, використовуючи для ідентифікації свою пластикову картку. Банкомат взаємодіє з клієнтом для отримання необхідної інформації для транзакції та з головним комп'ютером банку, який одержує інформацію, перевіряє та видає дозвіл або заперечує проведення транзакції.

Картка – електронна пластикова картка клієнта. Кожна картка містить код банку, номеркартки, персональний код клієнта.

Клієнт – власник одного або декількох рахунків у банку.

Транзакція – одиничний інтегрований запит на виконання деякої послідовності операцій над рахунками одного клієнта.

Рахунок – одиничний банківський рахунок, над яким виконуються транзакції. Клієнт може мати декілька рахунків.

Предметна область може бути представлена у вигляді описів користувача (user story).

Як [роль користувача] я хочу [діяльність], щоб я міг [вигода].

Як [роль користувача] я можу [діяльність], щоб [вигода].

Роль користувача - хто? (новий користувач, гість, шукач роботи) Діяльність - що? (функціональність, дія системи).

Вигода - чому? (цінність для кінцевого користувача) Наприклад, «Як користувач я хочу, щоб веб-сторінки завантажувались протягом 2 або3 секунд, щоб я міг швидко працювати».

**Побудова діаграми варіантів використання**

Для побудови діаграми варіантів використання моделі банкомату необхідно спочатку виділити акторів і прецеденти:

Актори:

- актор на ім’я Банк (це зовнішня система);
- актор на ім’я Клієнт банкомату.

Прецеденти:

- отримання довідки про стан рахунку;
- зняття готівки;
- перевірка ПІН коду;
- блокування кредитної картки.

Залишається побудувати діаграму варіантів використання (рисунок 1).

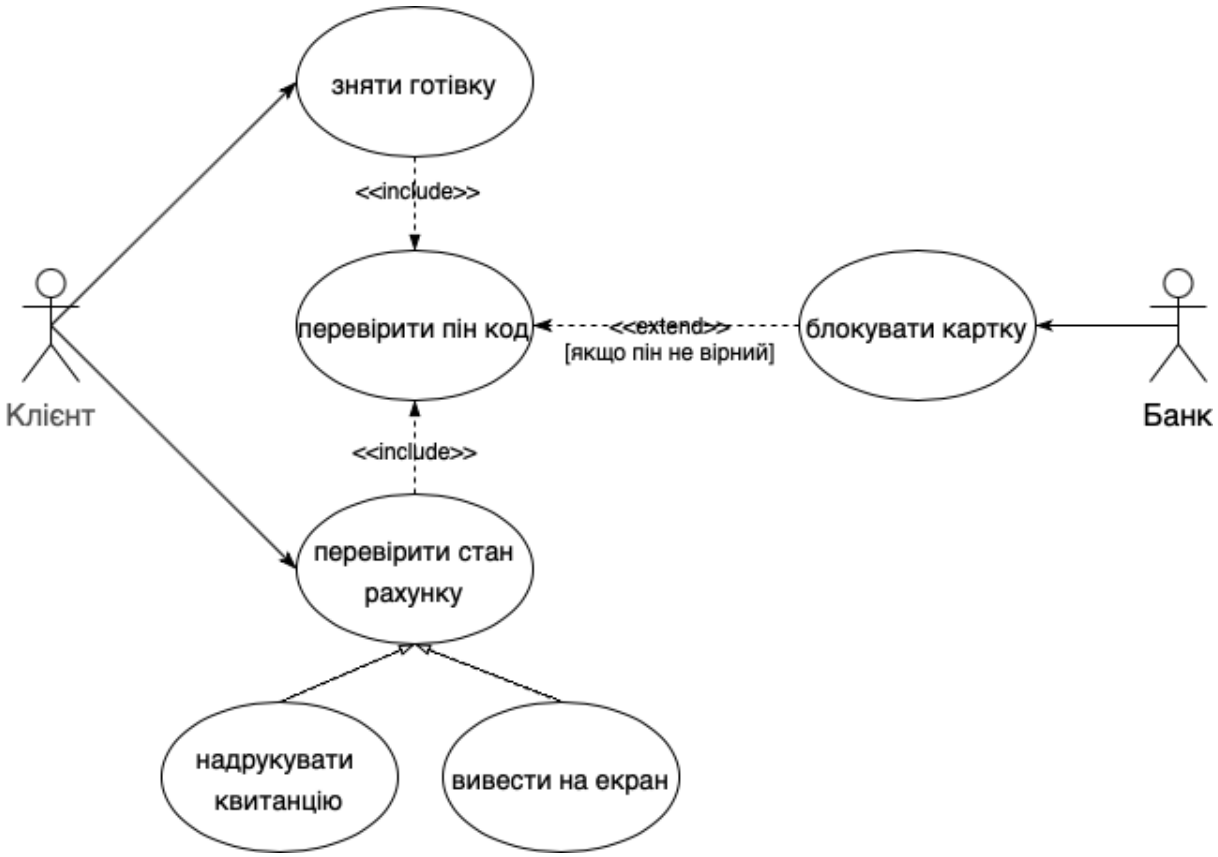



Рисунок 1 – Приклад діаграми варіантів використання банкомату з наведеного опису

Кожен прецедент в діаграмі має бути описаний для подальшого використання зі специфікування вимог[1]. Приклад опису наведений в таблиці 2.

Таблиця 2 – Приклад опису варіанту використання

| | |
|---|---|
| Use case name | Назва варіанту використання (наприклад: Створити обліковий запис) |
| Use case ID | Унікальний ідентифікатор варіанту використання (наприклад: UC-01) |
| Goals | Ціль варіанту використання (наприклад: Створення облікового запису користувача Адміністратором) |
| Actors | Актор, що працює з варіантом використання (наприклад: Адміністратор) |

| | |
|---|---|
| Trigger | Що ініціює виконання варіанту використання (наприклад: Перехід у розділ редагування та створення облікових записів чи надходження запиту на створення облікового запису) |
| Pre-conditions | Передумова, стан програмного забезпечення перед початком роботи з варіантом використання (наприклад:<br>1) Адміністратор увійшов у систему<br>2) В Адміністратора достатньо прав для створення облікового запису) |
| Flow of Events | Основні дії, що виконуються у варіанті використання та події, що відбуваються під час цього (наприклад:<br>1) Адміністратор викликає функцію створення облікового запису<br>2) Система пропонує ввести інформацію про обліковий запис<br>3) Адміністратор заповнює дані<br>4) Система перевіряє коректність введеної інформації<br>5) Система створює обліковий запис<br>6) Система повідомляє користувача (для кого створений обліковий запис) про створення облікового запису<br>7) Система повідомляє Адміністратора про успішність створення облікового запису для певного користувача) |
| Extension | Розширення можливостей та функціоналу варіанту використання або альтернативні дії (наприклад: Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним фоном; Система відображає попередження) |
| Post-Condition | Постумова, стан програмного забезпечення після закінчення роботи з варіантом використання (наприклад:<br>1) Обліковий запис створений<br>2) Користувач, для якого створювали запис, повідомлений) |

Особливості використання відношень у діаграмі варіантів використання

Відношення (relationship) [1] – семантичний зв'язок між окремими елементами моделі. Один актор може взаємодіяти з кількома варіантами використання. В цьому випадку цей актор звертається до кількох сервісів даної системи. У свою чергу, один варіант використання може взаємодіяти з декількома акторами, надаючи для них свій сервіс.

При побудові діаграми були враховані наступні особливості використання відношення асоціації:

– один варіант використання може мати кілька асоціацій із різними акторами;

– два варіанти використання, що відносяться до одного актора, не можуть бути асоційовані, тому що кожен із них визначає закінчений фрагмент функціональності актора;

– два варіанти використання, що відносяться до різних акторів не асоціюються між собою, а можуть лише узагальнюватись чи залежати один від одного;

– асоціації можуть бути кратними. Кратність (multiplicity) асоціації вказується поруч із позначенням компонента діаграми, який є учасником цієї асоціації. Кратність характеризує загальну кількість конкретних екземплярів даного варіанту використання, які можуть виступати як елементи даної асоціації. Кратність має спеціальне позначення у формі однієї або кількох цифр і, можливо, спеціального символу "*" (зірочка означає умовно необмежену кількість) (приклад наведений на рисунок 2).

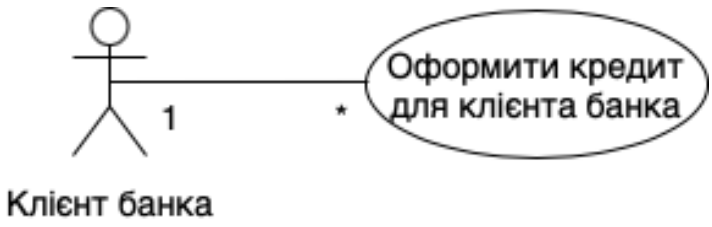



Рисунок 2 – Приклад кратної асоціації

Також використання узагальнення включає наступне:

– головною особливістю узагальнення є те, що воно може пов'язувати між собою лише елементи одного типу;

– один варіант використання може мати кілька батьківських варіантів використання (множинне успадкування);

– один варіант використання може бути предком кількох дочірніх варіантів використання (таксономічний характер відносин);

– одні актори можуть узагальнювати інших.

Особливості використання відношення включення:

– один базовий варіант використання може бути пов'язаний відношенням включення з декількома варіантами використання, що включаються;

– один варіант використання може бути включений до інших варіантів використання;

– на одній діаграмі варіантів використання не може бути замкнутого шляху стосовно включення.

Особливості використання відношення розширення:

– один базовий варіант використання може мати кілька точок розширення, з кожною з яких повинен бути пов'язаний розширюючий варіант використання;

– один розширюючий варіант використання може бути пов'язаний з відношенням розширення з декількома базовими варіантами використання;

– розширюючий варіант використання може мати власні розширюючі варіанти використання;

– на одній діаграмі варіантів використання не може бути замкнутого шляху щодо розширення.

Приклад використання точка розширення – що є іменованим маркером, який посилається на місце або декілька місць у послідовності поведінки варіанта

використання (там де можна внести додаткову поведінку) наведений на рисунку 3.

Синтаксис точки розширення можливо представляти двома[5] способами:

– під стрілкою розширення у [ .. ] написати умову;
– прописати точки розширення у базовому варіанті використання.

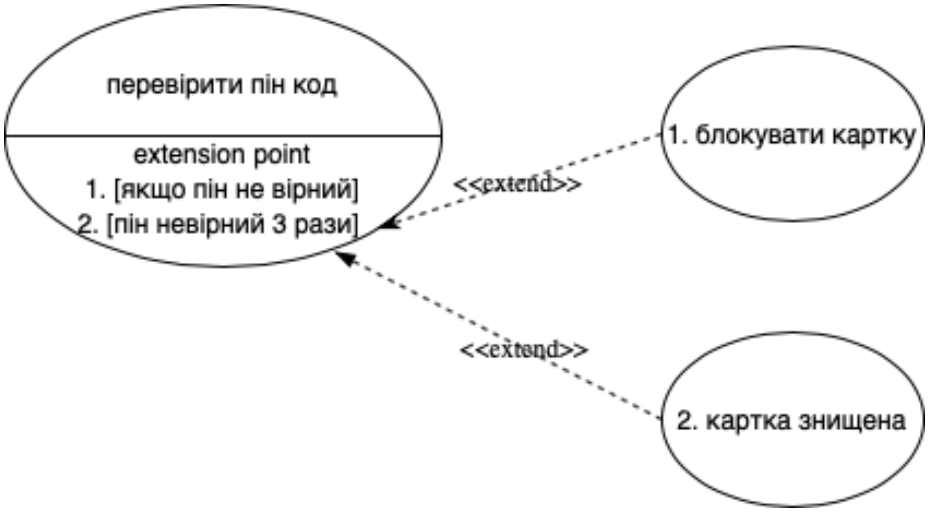



Рисунок 3 – Приклад точки розширення

Інші приклади діаграм варіантів використання наведено на рисунку 4 (адміністрування даних) та рисунку 5 (реєстрація користувача).

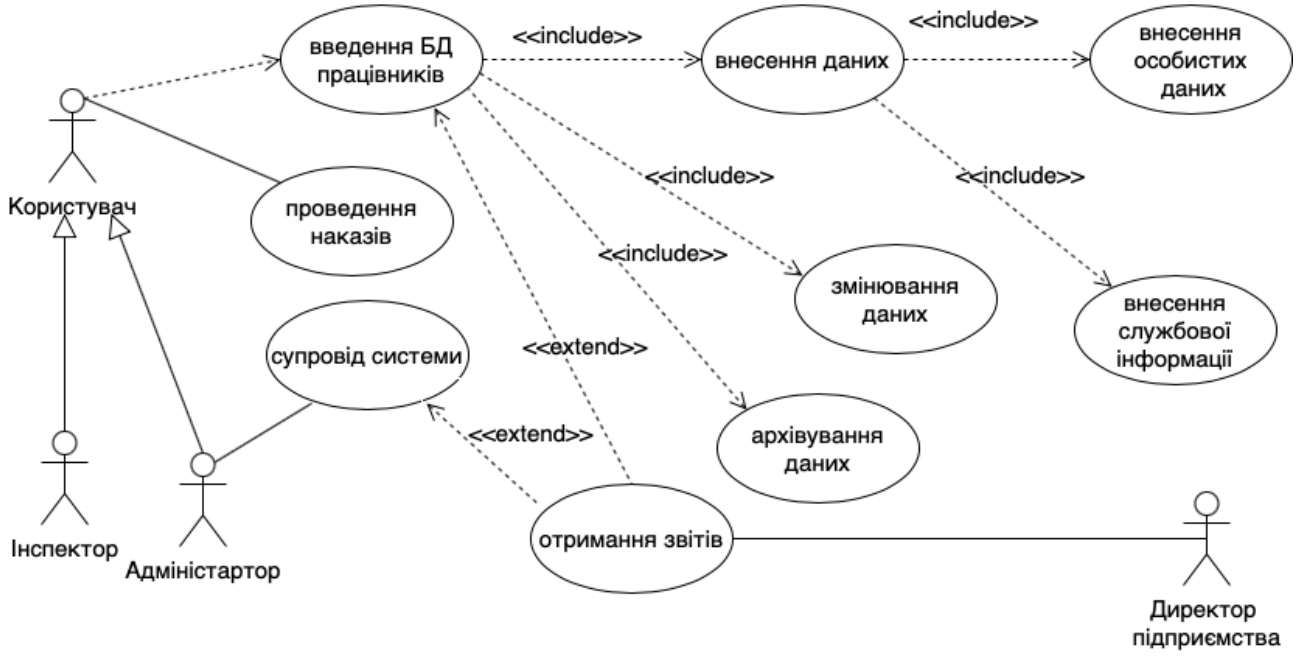



Рисунок 4 – Діаграма варіантів використання (адміністрування даних)

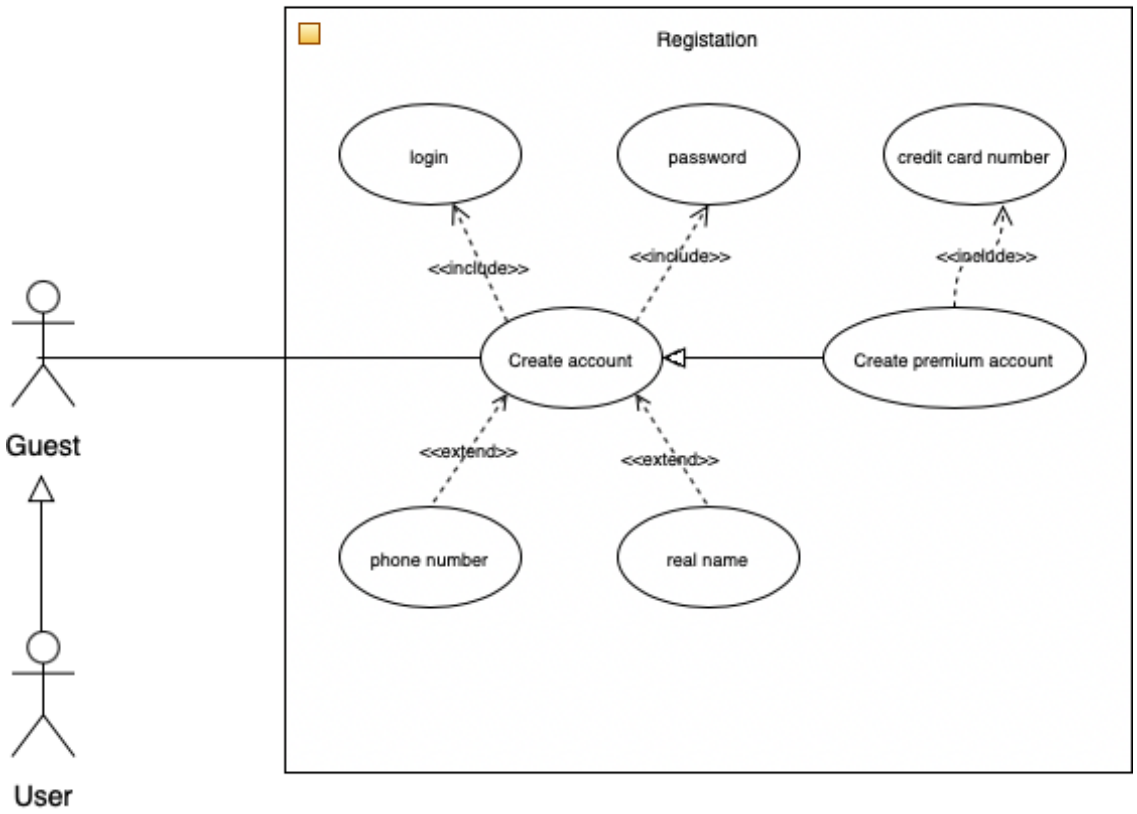



Рисунок 5 – Діаграма варіантів використання (реєстрація користувача)

*Примітки* (notes) [1] у мові UML призначені для включення до моделі довільної текстової інформації, що має безпосереднє відношення до контексту проекту, що розробляється. В якості такої інформації можуть бути коментарі розробника (наприклад, дата та версія розробки діаграми або її окремих компонентів), обмеження (наприклад, значення окремих зв'язків або екземпляри сутностей) і помічені значення. Стосовно діаграм варіантів використання примітка може мати найзагальнішу інформацію, що відноситься до загального контексту системи.

Графічно примітки позначаються прямокутником із загнутим верхнім правим куточком (приклад наведений на рисунку 6). Усередині прямокутника міститься текст примітки. Примітка може ставитись до будь-якого елемента діаграми, в цьому випадку їх з'єднує пунктирна лінія. Якщо примітка стосується кількох елементів, то від нього проводяться, відповідно, кілька

ліній. Зрозуміло, що примітки можуть бути присутніми не лише на діаграмі варіантів використання, але й на інших канонічних діаграмах.

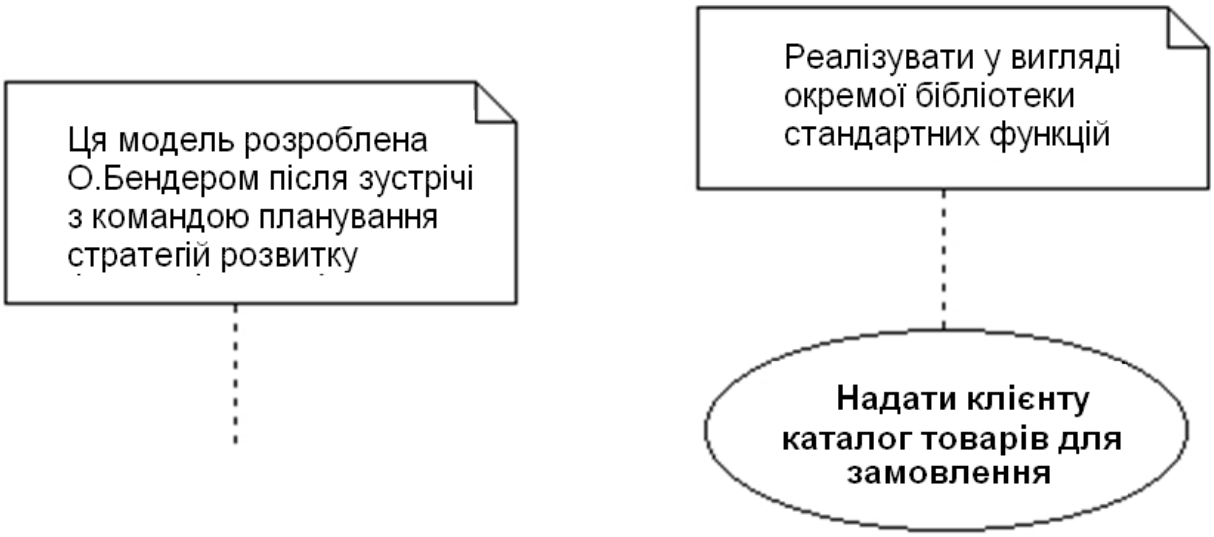



Рисунок 6 – Приклади приміток

**Завдання на лабораторну роботу**

1. Ознайомитись за стандартом UML, вивчити структуру та елементи уніфікованої мови моделювання UML.
2. Ознайомитися з інтерфейсом та можливостями CASE-інструменту Rational Rose (PowerDesigner, Enterprise Architect, Microsoft Visio, Draw.io).
3. За узгодженням з викладачем обрати варіант завдання (таблиця 1.3) для виконання лабораторної робіти.
4. Провести аналіз предметної області та зробити її короткий опис;
5. Визначити не менше двох Акторів, та не менше десяти Варіантів використання.
6. Побудувати діаграму варіантів використання на основі проведеного попереднього аналізу.
7. Провести опис основних варіантів використання, включаючи розширення та включення.

Таблиця 3 – Варіанти завдань

| *№* | *Назва* | *Короткий опис для реалізації* |
|---|---|---|
| 1 | Система лікувального закладу | Дозволяє лікарям вносити пацієнтів, їх діагнози та призначення, а медсестрам виконувати їх |
| 2 | Інформаційна система приймальної комісії ВНЗ | Дозволяє вносити та переглядати дані про вступників, врахувати розподіл балів за різними предметами та проводити розподіл за спеціальностями |
| 3 | Інформаційна система деканату | Дозволяє приймати й відраховувати студентів, вести облікуспішності за підсумками сесії, переводити студентів із групи в групу чи з курсу на курс |
| 4 | Система клімат контролю будинка | Дозволяє встановлювати і підтримувати температуру, вологість повітря окремо у різних приміщеннях будинку. (можливо використання різних приладів) |
| 5 | Інформаційна система готелю | Дозволяє вносити відвідувачів за різними типами номерів, а також надавати додаткові послуги. В залежності від типу номера можливі додаткові знижки |
| 6 | Інформаційна система поліклініки | Дозволяє ставити і знімати хворих з обліку, записувати хворих наприйом до лікарів, враховувати факт прийому, надавате направлення на ліки |
| 7 | Інформаційна система ресторану | Дозволяє вибирати різні страви, фіксувати їх подачу за столики, та формувати загальний чек по столику |
| 8 | Система курєрської доставки | Дозволяє фіксувати прийом вантажу, вираховувати вартість доставки, проводити доставку за адресою |
| 9 | Інформаційна система екзаменаційної сесії | Дозволяє вносити предмети, оцінки, студентів викладачів. Формувати відомості по групах та предметах |
| 10 | Система електронної бібліотеки | Дозволяє додавати різну літературу та авторів, проводити пошук, купувати. В залежності від здійснених покупок нараховувати знижку |
| 11 | Система кадрової агенції | Дозволяє вносити вакансії, та проводити відбір персоналу для підприємств за визначеними запитами |
| 12 | Інформаційна система аптеки | Дозволяє реєструвати та додавати ліки до каталогу за спеціалізацією, шукати ліки, продавати їх |

| № | Назва | Короткий опис для реалізації |
|---|---|---|
| 13 | Система діяльності рекламного агентства | Дозволяє створювати рекламу різних видів стосовно запитів покупців |
| 14 | Інформаційна система школи | Дозволяє проводити прийом школярів, вчителів. Розподіл їх по класам та предметам (без врахування оцінок за предмети) |
| 15 | Інформаційна система перукарні | Дозволяє враховувати прийом відвідувачів за різними типами послуг, призначати на ці послуги різних виконавців |
| 16 | Система житлового агентства | Дозволяє квартиронаймачам вибрати й зняти житло, а власникам житла - запропонувати й здати житло. |
| 17 | Система бронювання квитків у театрах | Дозволяє вибирати вистави, час та місце проведення, також є можливість врахування постійної чи гастролюючої трупи |
| 18 | Інформаційна система ЗНО | Дозволяє проводити реєстрацію абітурієнтів на певний вид екзаменів, проводити їх |
| 19 | Система прокату автомобіля | Дозволяє реєструвати авто, його стан та переміщення при здачі в оренду, враховуючи дані користувача |
| 20 | Система обліку вантажних перевезень однієї компанії | Дозволяє приймати різні типи вантажу, і в залежності від цього реалізовувати його перевезення за певною адресою |
| 21 | Система автобусного депо | Дозволяє реєструвати автобуси, їх переміщення, призначення водіїв, можливість їх заміни |
| 22 | Система фіксації правопорушень | Дозволяє фіксувати правопорушення на вулицях міста, класифікувати та відправляти листи порушникам |
| 23 | Система обліку робочогочасу | Дозволяє керівникам видавати завдання й відслідковувати хід їхнього виконання, а виконавцям - вести облік робочого часу, витраченого на виконання кожного завдання. |
| 24 | Система обліку у гуртожитку | Дозволяє проводити облік кімнат, додаткових послуг, та реалізувати заповнення студентами гуртожитку |
| 25 | Інформаційна система ДАІ (1) | Дозволяє проводити реєстрацію та перереєстрацію транспортних засобів |
| 26 | Інформаційна системапаркування автомобілів | Дозволяє визначати наявність вільних місць для паркування за маршрутом і оповістити водія, коли знайдено досить велике місце для паркування |

| № | Назва | Короткий опис для реалізації |
|---|---|---|
| 27 | Інформаційна система ДАІ (2) | Дозволяє обрати місце навчання, здійснити реєстрацію, оплатити навчання, отримати посвідчення водія на право керування транспортними засобами. |
| 28 | Система маршрутів громадського транспорту | Дозволяє контролювати переміщення різного транспорту в місті за певними маршрутами |
| 29 | Інформаційна системаскладу | Дозволяє враховувати надходження й вихід товарів зі складу, а також визначати місце зберігання товарів на складі в залежності від термінів та умов зберігання |
| 30 | Система діловодства організації | Дозволяє реєструвати вхідні та вихідні документи різного призначення, та контролювати їх переміщення, враховуючи виконавців |
| 31 | Система здачі торгівельної площі в оренду | Дозволяє контролювати розподіл приміщень у великому супермаркеті і здавати їх в оренду, в залежності від площі-вартості-особливих умов оренди |
| 32 | Система обліку товарів у супермаркеті | Дозволяє контролювати наявність, ціни, терміни та умови зберігання товарів |
| 33 | Інформаційна система ЖЕКу | Дозволяє формувати платіжні документи з оплати житлово-комунальних послуг, оплату послуг, контролювати надані послуги. |
| 34 | Система спортивних гуртків | Дозволяє реєструвати студентів, та розподіляти за різними гуртками в залежності від уподобань, створення груп у відповідності до розподілу по факультетах |
| 35 | Система продажу в автосалоні | Дозволяє реєструвати в салоні нові машини, проводити продаж, проводити розподіл авто за певними характеристиками |

## Контрольні питання

1. Поясніть сутність об’єктно-орієнтованого підходу до розробки інформаційних систем та його відмінність від структурного підходу.

2. Наведіть визначення мови UML та розкрийте її склад. Наведіть складові моделі UML.

3. Охарактеризуйте відношення залежності та розкрийте його значення на діаграмах UML.

4. Охарактеризуйте відношення асоціації та розкрийте його значення на діаграмах UML. Наведіть доповнення, які використовуються до асоціацій. Охарактеризуйте відношення узагальнення та розкрийте його значення на діаграмах UML.

5. Дайте визначення діаграми прецедентів та її основне призначення.

6. Наведіть поняття прецеденту та актора. Охарактеризуйте та наведіть приклади відношень між ними.

7. Охарактеризуйте відношення між прецедентами на діаграмі прецедентів

# ЛАБОРАТОРНА РОБОТА №2

## Специфікування предметної галузі проекту засобами мови UML

Мета роботи: дослідження класів та отримання навиків у побудові діаграми класів UML для специфікування предметної галузі, використанні стереотипів UML та структуруванні моделі UML за допомогою пакетів.

## Короткі теоретичні відомості

Діаграма класів (Class diagram) [1] відноситься до класу структурних діаграм та призначена для відображення статичної структури предметної галузі проекту або системи, що проектується. Діаграма містить класи і взаємозв'язки між ними та дозволяє описати їх структуру та типи відношень.

Клас – це група об'єктів із спільними властивостями (атрибутами), функціями та відношеннями з іншими об'єктами. Графічна нотація класу розділяється на три сектори: ім'я класу, атрибути та операції.

Ім'я класу – унікальне ім'я, що записується з великої букви без пропусків.

Атрибут є властивістю класу, якому надається ім'я, тип та інші характеристики згідно з наступним синтаксисом:

< специфікатор_видимості > < ім'я > :< тип > = <значення за умовчанням>

Операція – це опис способу виконання дій з об'єктом (екземпляром класу), наприклад: читання або змінення значень атрибутів, обчислення нових значень за інформацією, яка знаходиться в об'єкті і т. ін.

Синтаксис опису операцій:

<специфікатор_видимості> < ім'я > (< список-параметрів >) : < тип_виразу,_який_повертає_значення > = <строка_властивостей>

У розділі атрибутів класу можуть вказуватися додаткові характеристики класу за допомогою наступних позначень (таблиця 4).

Таблиця 4 – Додаткові характеристики класу

| | |
|---|---|
| changeable- | ніяких обмежень на зміну значення атрибута не накладається |
| addOnly- | значення застосовується до атрибутів, кратність яких більше одиниці. При зміні значення атрибута нове значення додається у масив значень, але старі значення не змінюються та не пропадають |
| frozen- | значення атрибута задається при ініціалізації об’єкта та не змінюється. |

Приклади опису операцій наведені у таблиця 5.

Таблиця 5 – Приклади операцій класу

| | |
|---|---|
| move | вказано тільки ім’я операції |
| +move(in from : Dpt, in to : Dpt) | вказані видимість операції, направлення передачі, імена та типи параметрів |
| +setPwd(in pwd : String = "password") | процедура, для якої вказано значення аргументу за умовчанням |

Специфікатор видимості певного класу показує чи має інший клас доступ до атрибутів та операцій цього класу. Основні специфікатори видимості наведені у таблиці 6. Усі додаткові види видимості визначаються розробниками, інструментами моделювання та генератором коду.

Таблиця 6 – Приклади операцій класу

| Значення | Опис |
|---|---|
| public (загальний) | доступний для усіх клієнтів класу |
| protected (захищений) | доступний тільки для підкласів і друзів класу |
| private (секретний) | доступний тільки для друзів класу |

Приклади графічного зображення класів наведені на рисунку 7.

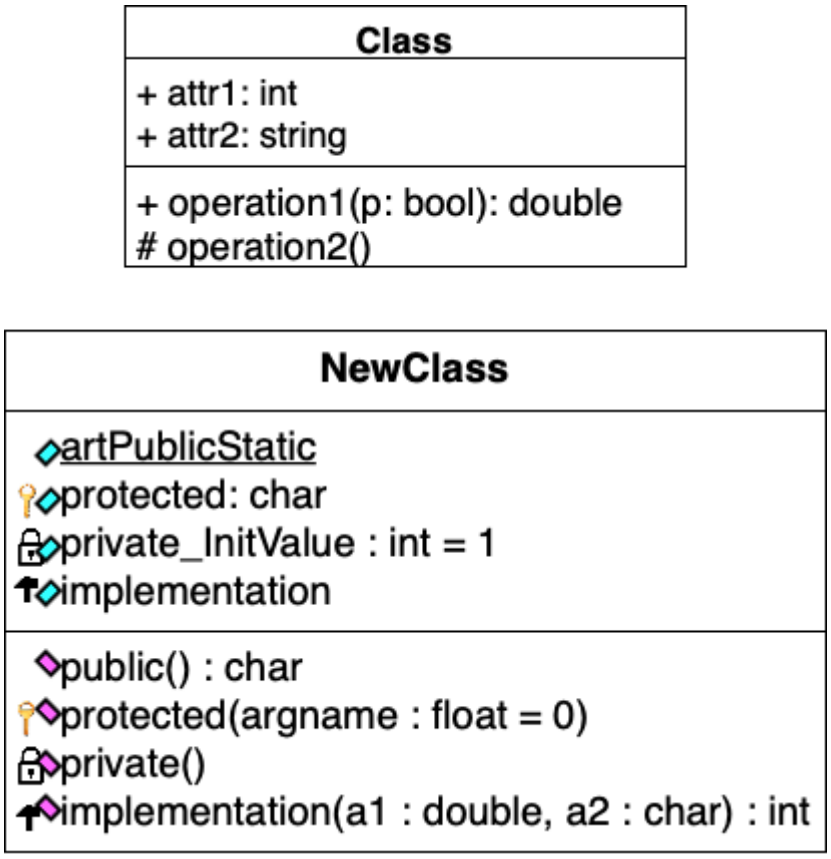



Рисунок 7 – Приклади класів

Між класами можна встановлювати зв’язки наступних видів, які показані на рисунку 8 у наступному порядку:

– *асоціація* – встановлює зв’язок між об’єктами;

– *агрегація* – це відношення виду „частина-ціле” між класом, який є збіркою (агрегатом), що включає до себе інші класи, та цими класами;

– *композиція* – це агрегація, в якій об’єкт, що включається, не може існувати без класу-агрегату;

– *успадкування* – клас-потомок успадковує структуру та поведінку предка, визначаючи при цьому власні атрибути та операції.

*Кратність зв’язків* вказує кількість класів або об’єктів, що можуть взаємодіяти по даному зв’язку та позначається наступним чином:

– 0..1 - жодного або один;

– 0..* - від нуля до багатьох;

– 1..1 - обов’язкова приналежність;

– 1..* - від одного до багатьох.

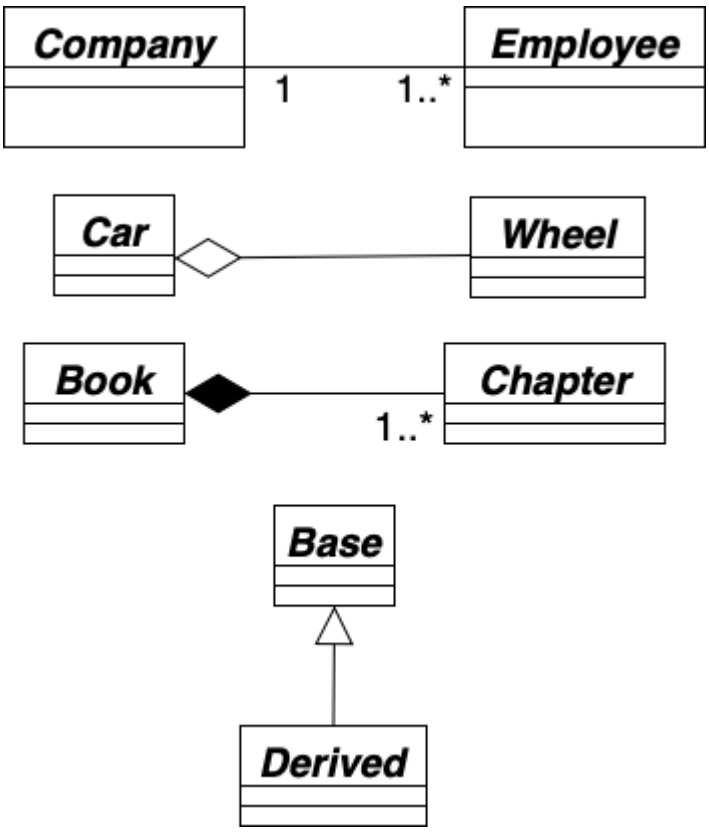



Рисунок 8 – Приклади зв’язків між класами

Окремий випадок відношення асоціації – так звана асоціація, що виключає *(Xor-association).* Семантика даної асоціації вказує на те, що з декількох потенційно можливих варіантів даної асоціації в кожен момент часу може використатися тільки один. На діаграмі класів асоціація, що виключає, зображується пунктирною лінією, що з'єднує дві й більше асоціацій (рисунок 9). Поруч з такою лінією записується обмеження у формі рядка тексту у фігурних дужках: {xor}.

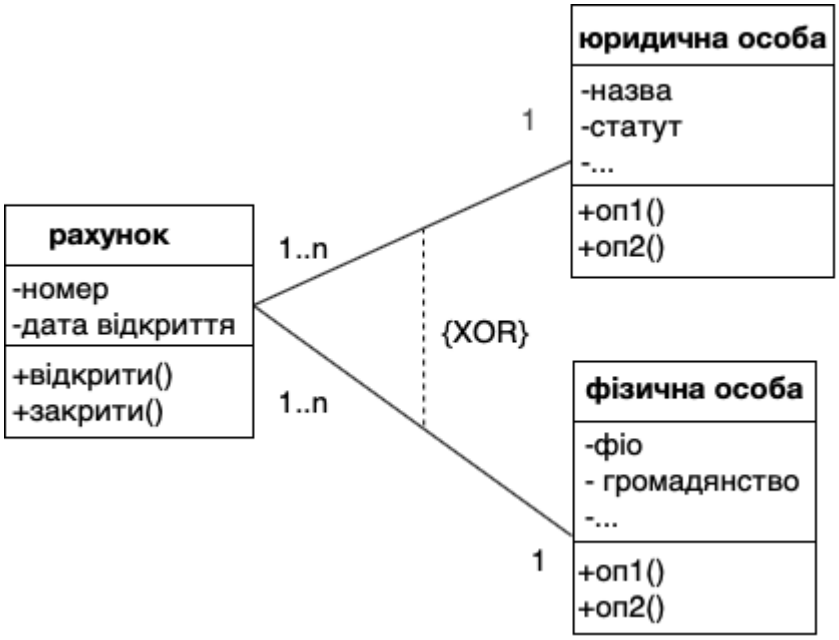



Рисунок 9 – Приклад використання xor-асоціації

Інтерфейс (interface) [1] служить для специфікації параметрів моделі, які видимі ззовні без зазначення їх внутрішньої структури. У мові UML інтерфейс є класифікатором і характеризує лише обмежену частину поведінки сутності, що моделюється. Стосовно діаграм варіантів використання, інтерфейси визначають сукупність операцій, які забезпечують необхідний набір сервісів чи функціональності для акторів. Інтерфейси неспроможні містити ні атрибутів, ні станів, ні направлених асоціацій. Вони містять лише операції без зазначення особливостей їх реалізації. Формально інтерфейс еквівалентний абстрактному класу без атрибутів та методів з наявністю лише абстрактних операцій.

На діаграмі класів інтерфейс зображується у вигляді маленького кола, поруч із яким записується його ім'я (рисунок 10, а). У якості імені може бути іменник, який характеризує відповідну інформацію або сервіс (наприклад, "датчик", "сирена", "відеокамера"), але частіше рядок тексту (наприклад, "запит до бази даних", "форма введення", "пристрій подачі звукового сигналу"). Якщо ім'я записується англійською, воно має починатися з великої літери I, наприклад, ISecurelnformation, ISensor (рисунок 10, б).

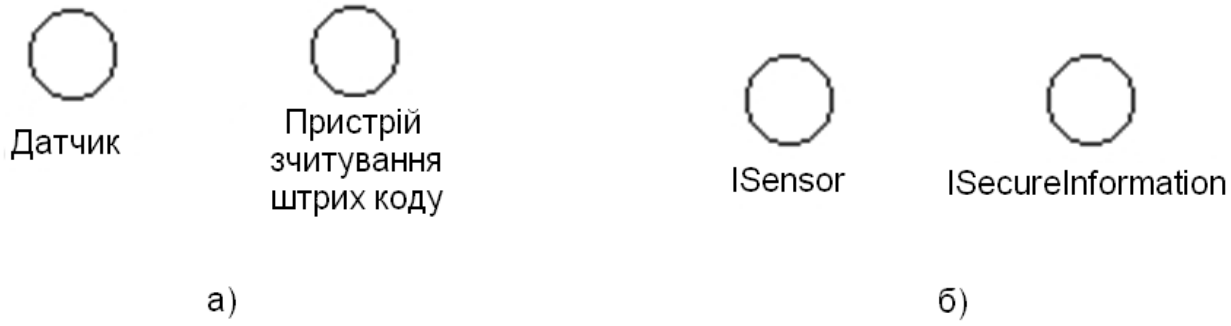



Рисунок 10 – Приклад зображення інтерфейсів

При побудові діаграми студентам рекомендовано при потребі використовувати стереотипи. *Стереотипи* [1] є одним із трьох типів механізмів розширюваності в уніфікованій мові моделювання (UML). Вони дозволяють проектувальникам розширювати словник UML для створення нових елементів моделювання, які отримуються з існуючих, але мають певні властивості, що підходять для конкретної проблеми, предметної області чи для іншого спеціалізованого використання. Термін походить від початкового значення слова "стереотип", яке використовується в друкарстві. Наприклад, при моделюванні мережі вам знадобляться символи для представлення маршрутизаторів та концентраторів. За допомогою стереотипних вузлів ви можете подавати їх у вигляді примітивних будівельних блоків.

Графічно стереотип відображається як ім'я в лапках («», або якщо такі лапки неприпустимі, <<>>) і розташоване над ім'ям іншого елементу. На додачу або в якості альтернативи він може бути позначений відповідною іконкою. Наприклад, стереотипи діаграми класів можуть бути використані для опису методів поведінки, таких як конструктор (рисунок 11).

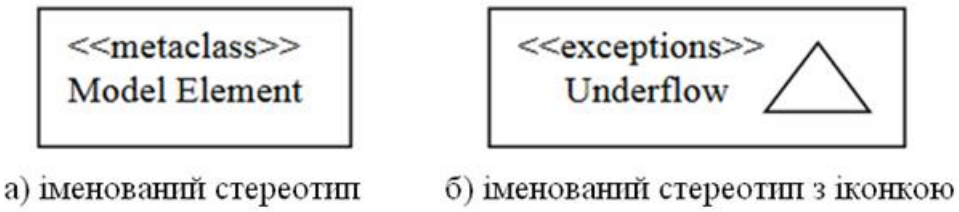
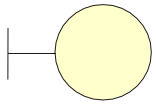
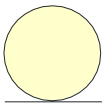
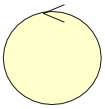



Рисунок 11 – Приклад використання стереотипів

У рамках проектування діаграми класів запропоновано спеціальні графічні примітиви [9], які можуть бути використані для уточнення семантики окремих класів, що наведені в таблиці 7, а також приклади на рисунках 12 та 13.

Таблиця 7 – Приклад стереотипів класів

| Позначення стереотипу | Назва стереотипу | Опис |
|---|---|---|
|  | <<boundary>><br>Граничний клас | клас, що розташовується на границі системи із зовнішнім середовищем і безпосередньо взаємодіє з акторами, але є складовою частиною системи. Виділяється для кожного зовнішнього агента (актора) для взаємодії з ним. |
|  | <<entity>><br>Клас сутність | пасивний клас, що містить інформацію, яка має зберігатися постійно й не повинна знищуватися зі знищенням об'єктів даного класу або припиненням роботи системи, що моделюється. Зазвичай, цей клас відповідає окремій таблиці бази даних. У цьому випадку його атрибути є полями таблиці, а операції – приєднаними або збереженими процедурами. Цей клас лише приймає повідомлення від інших класів моделі |
|  | <<control>><br>Керуючий клас | клас, відповідальний за координацію дій інших класів. На кожній діаграмі класів повинен бути хоча б один керуючий клас, причому кількість повідомлень, що посилають об'єктам керуючого класу – мала, у порівнянні із числом розсилаємих ними. Керуючий клас відповідає за координацію дій інших класів.<br>Клас не має власної функціональності і делегує відповідальність іншим класам та забезпечує ведення бізнес-правил. |
| | <<interface>><br>інтерфейс | іменована множина операцій, які характеризують поведінку окремого елемента моделі. Інтерфейс у контексті мови UML є спеціальним випадком класу, у якого є операції, але відсутні атрибути.<br>Інтерфейс слугує засобом об’єднання об’єкта з іншими об’єктами. |

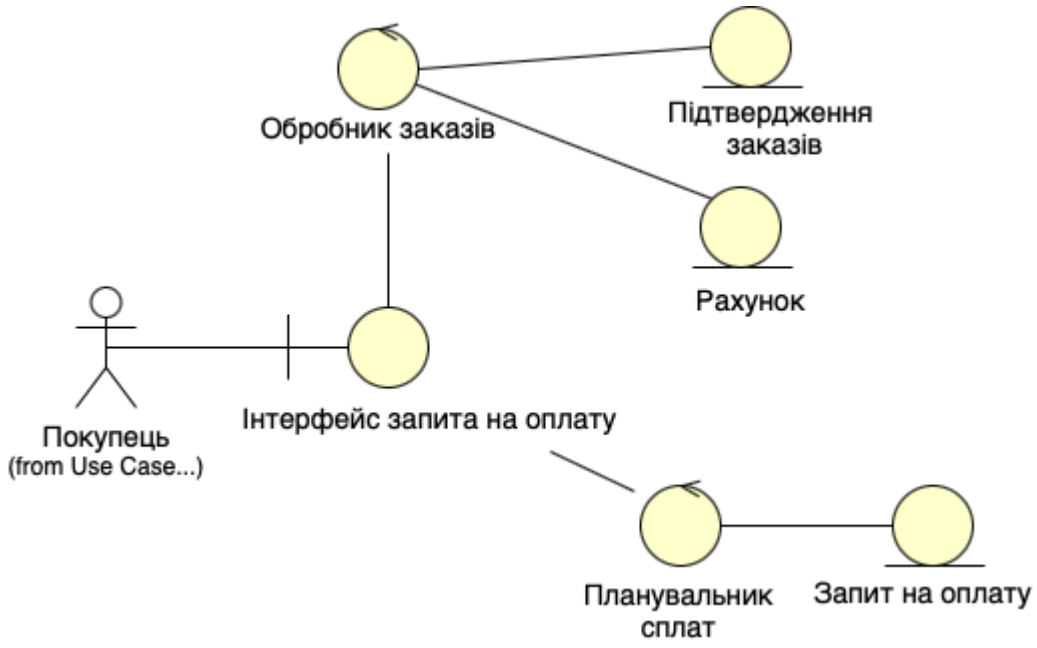



Рисунок 12 – Приклад використання стереотипів

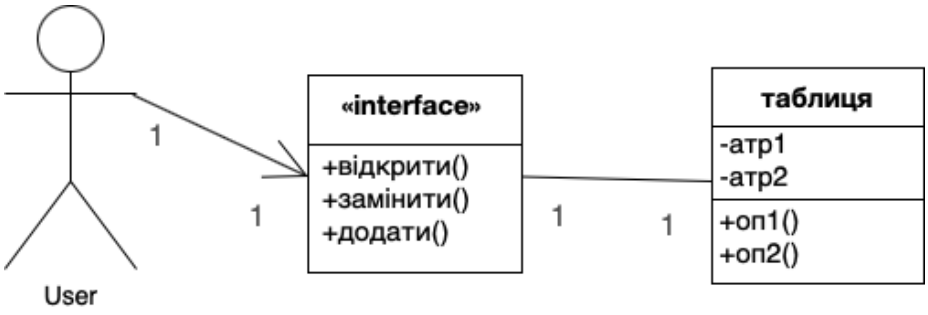



Рисунок 13 – Приклад використання інтерфейсу

У таблиці 8 наведено загальні стереотипи мови UML [1] їх застосування та призначення які студентам бажано використовувати при виконанні лабораторної роботи.

Таблиця 8 – Загальні стереотипи UML

| Стереотип | Для чого застосовується | Призначення |
|---|---|---|
| actor | Клас (class) | Визначає зв’язану множину ролей, які виконує користувач асоційований з прецедентом при взаємодії з ним |
| access | Залежність (dependency) | Повідомляє, що відкритий вміст цільового пакету доступний у просторі імен вихідного пакету |

| Стереотип | Для чого застосовується | Призначення |
|---|---|---|
| association | Кінцева точка зв’язку (link end) | Вказує, що відповідний об'єкт в області видимості асоціації |
| become | Повідомлення (message) | Цільовий об'єкт співпадає з вихідним, але в більш пізній момент часу. При цьому, можливо, він матиме інші значення, стани чи ролі. |
| bind | Залежність (dependency) | Вихідний клас створює екземпляр цільового шаблону із даними фактичними параметрами |
| call | Залежність (dependency) | Вихідна операція викликає цільову операцію |
| copy | Повідомлення (message) | Цільовий об'єкт – це точна, але незалежна копія вихідного |
| create | Подія (event), Повідомлення (message) | Цільовий об'єкт створено в результаті події або повідомлення |
| derive | Залежність (dependency) | Вихідний об'єкт може бути обчислений за даними цільового |
| destroy | Подія (event), Повідомлення (message) | Цільовий об'єкт знищено внаслідок події чи повідомлення |
| document | Компонент (component) | Компонент представляє документ |
| enumeration | Клас (class) | Визначає перелічуваний тип, включаючи його можливі значення як набір ідентифікаторів |
| exception | Класс(class) | Визначає подію, яка може бути збуджена або перехоплена операцією |
| executable | Компонент (component) | Описує компонент, який може бути виконаний у вузлі |
| extend | Залежність (dependency) | Цільовий варіант використання розширює поведінку вихідного в даній точці розширення |
| facade | Пакет (package) | Пакет, який є лише представленням іншого пакету |
| file | Компонент (component) | Компонент, який представляє документ, що містить вихідний код чи дані |
| framework | Пакет (package) | Пакет, що складається в основному із зразків (патернів) |
| friend | Залежність (dependency) | Вихідний клас має спеціальні права видимості у цільовому |
| global | Кінцева точка | Відповідний об'єкт видимий, оскільки |

| Стереотип | Для чого застосовується | Призначення |
|---|---|---|
| | зв’язку (link end) | належить глобальній області |
| import | Залежність (dependency) | Відкритий зміст цільового пакета стає частиною простору імен вихідного пакету, так ніби воно було оголошено безпосередньо в ньому |
| implementation | Узагальнення (generalization) | Нащадок успадковує реалізацію батька, але не відкриває і не підтримує його інтерфейсів, унаслідок чого не може бути підставлений замість батька |
| implementation Class | Клас (class) | Реалізація класу деякою мовою програмування |
| include | Залежність (dependency) | Вихідний прецедент явно включає поведінку іншого прецеденту в точці, що визначається вихідним |
| instanceOf | Залежність (dependency) | Вихідний об'єкт є екземпляром цільового класифікатора |
| instantiate | Залежність (dependency) | Операції над вихідним класом створюють екземпляри цільового класу |
| interface | Клас (class) | Описує множину операцій, що визначають, що може робити клас чи компонент |
| invariant | Обмеження (constraint) | Обмеження, яке завжди має виконуватися для асоційованого елемента |
| library | Компонент (component) | Статична чи динамічна об'єктна бібліотека |
| local | Кінцева точка зв’язку (link end) | Відповідний об'єкт видимий, оскільки знаходиться у локальній області дії |
| metaclass | Класифікатор (classifier) | Класифікатор, всі об'єкти якого є класами |
| model | Пакет (package) | Описує семантично замкнуту абстракцію системи |
| parameter | Кінцева точка зв’язку (link end) | Відповідний об'єкт видимий, оскільки є параметром |
| postcondition | Обмеження (constraint) | Обмеження, яке має виконуватись після виконання операції |
| powertype | Клас (class) | Класифікатор, всі об'єкти якого є нащадками даного батька |
| precondition | Обмеження (constraint) | Обмеження, яке має виконуватися перед виконанням операції |
| process | Клас (class) | Класифікатор, екземпляр якого представляє ресурсомісткий потік управління |

| Стереотип | Для чого застосовується | Призначення |
|---|---|---|
| refine | Залежність (dependency) | Каже, що вихідний об'єкт є більш детальною абстракцією, ніж цільовий |
| requirement | Коментарій (comment) | Описує бажану властивість або поведінку системи |
| responsibility | Комментарий (comment) | Описує контракт чи зобов'язання класу |
| return | Подія (event), Повідомлення (message) | Повідомлення, що повертає значення виконаної операції або процедури об'єкту, що викликав її |
| send | Залежність (dependency) | Вихідна операція посилає цільову подію |
| signal | Клас (class) | Асинхронний стимул (збудження), який передається одним екземпляром іншому |
| stereotype | Клас (class) | Класифікатор – це стереотип, який може бути застосований до інших елементів |
| stub | Пакет (package) | Пакет виступає у ролі заступника для відкритого вмісту іншого пакета |
| subsystem | Пакет (package) | Описує групування елементів, ряд яких становить специфікацію поведінки інших елементів |
| system | Пакет (package) | Описує пакет, що представляє всю систему, що моделюється |
| table | Компонент (component) | Компонент, що представляє таблицю бази даних |
| thread | Клас (class) | Класифікатор, екземпляр якого представляє полегшений потік керування |
| trace | Залежність (dependency) | Цільовий елемент – це історичний предок вихідного |
| type | Клас (class) | Абстрактний клас, який використовується тільки для специфікації структури та поведінки (але не реалізації) множини об'єктів |
| use | Залежність (dependency) | Семантика вихідного елемента залежить від семантики відкритого вмісту цільового елемента |
| utility | Клас (class) | Визначає клас, для якого область дії всіх атрибутів та операцій – клас |

## Завдання на лабораторну роботу

1. Виявити класи, які відносяться до системи що проектується (мінімум 5 класів). Намагатися використовувати повний синтаксис при описі класів.
2. Стереотипи класів використовувати тільки там де вони потрібні! В інших випадках беремо стандартне зображення класу.
3. Коротко текстом описати призначення та використання кожного класу та його атрибутів і операцій.
4. Намагатися використати усі наведені у прикладах в лабораторній роботі зв’язки між класами на діаграмі.

## Контрольні питання

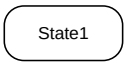
1. Опишіть призначення діаграми класів.
2. Які типи зв’язків існують між класами?
3. У чому відмінність між відношеннями агрегації та композиції?
4. Наведіть види стереотипів класів та приклади використання.

# ЛАБОРАТОРНА РОБОТА №3

## Моделювання станів системи засобами UML

Мета роботи: дослідження діаграм UML, які застосовуються для опису станів системи, що розроблюється, та отримання навиків у їх побудові.

## Короткі теоретичні відомості

Діаграма станів (State machine diagram) [1] (відноситься до класу поведінкових діаграм) показує стани системи, що проектується, чи окремого об’єкта. Нотація стану, у якому може перебувати система, має вигляд 

. Переходи між станами позначаються направленими дугами з назвами подій, що викликали зміни стану системи. Перехід треба описати у вигляді формули за наступним синтаксисом:

Подія [Умова] / Дія ^Ім’яКласу.Ім’яПодії

У станах можна показувати операції системи:

– Операції, які виконуються одразу після переходу до певного стану називають вхідними (entry/).

– Внутрішні операції (do/) закінчуються до переходу в інший стан.

– Операції, які виконуються перед виходом із стану називають вихідними (exit/).

– Дія, яка активізується за певною умовою  позначаються, як „event/”.

На діаграмі можуть бути ***прості і складні стани***. Складні (composite state) включають вкладені підстани. Декомпозиція складного стану може здійснюватися як на основній діаграмі, так і окремо, але на основній діаграмі слід використовувати елемент з піктограмою декомпозиції.

Складний стан може містити два або більше паралельних підавтоматів або кілька послідовних станів.

*Послідовні стани* використовуються для моделювання такої поведінки об'єкта, під час якого в кожний момент часу об'єкт може перебувати в одному і

лише одному стані. Поведінка об'єкта в цьому випадку є послідовною зміною підходів, починаючи від початкового і закінчуючи кінцевим підстаном.

*Паралельні стани* дозволяють специфікувати два і більше підавтомата, які можуть виконуватися паралельно всередині складової події, приклад наведений на рисунку 14.

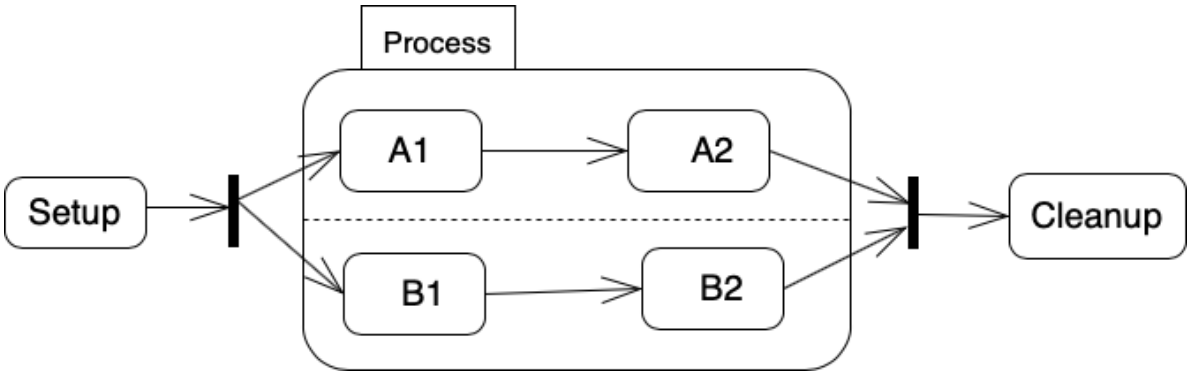



Рисунок 14 – Приклад відображення діаграми станів (паралельні стани)

Переходи можуть здійснюватися як у самий композитний стан, так і в один з його підходів. Таким чином, перехід, стрілка якого з'єднана з межею деякого складеного стану, позначає перехід у складовий стан. Він еквівалентний переходу до початкового стану кожного з підавтоматів. Перехід, що виходить зі складового стану, відноситься до кожного з вкладених підстанів. Це означає, що об'єкт може залишити складовий суперстан, перебуваючи в будь-якому з його станів. Якщо необхідно вказати конкретний стан з якого може здійснитися вихід з композитного стану, достатньо додати перехід від стану в цільовий стан.

Загальний приклад діаграми станів наведено на рисунку 15. Цей приклад відображає стан «дзвінок по телефону» у розгорнутому вигляді.

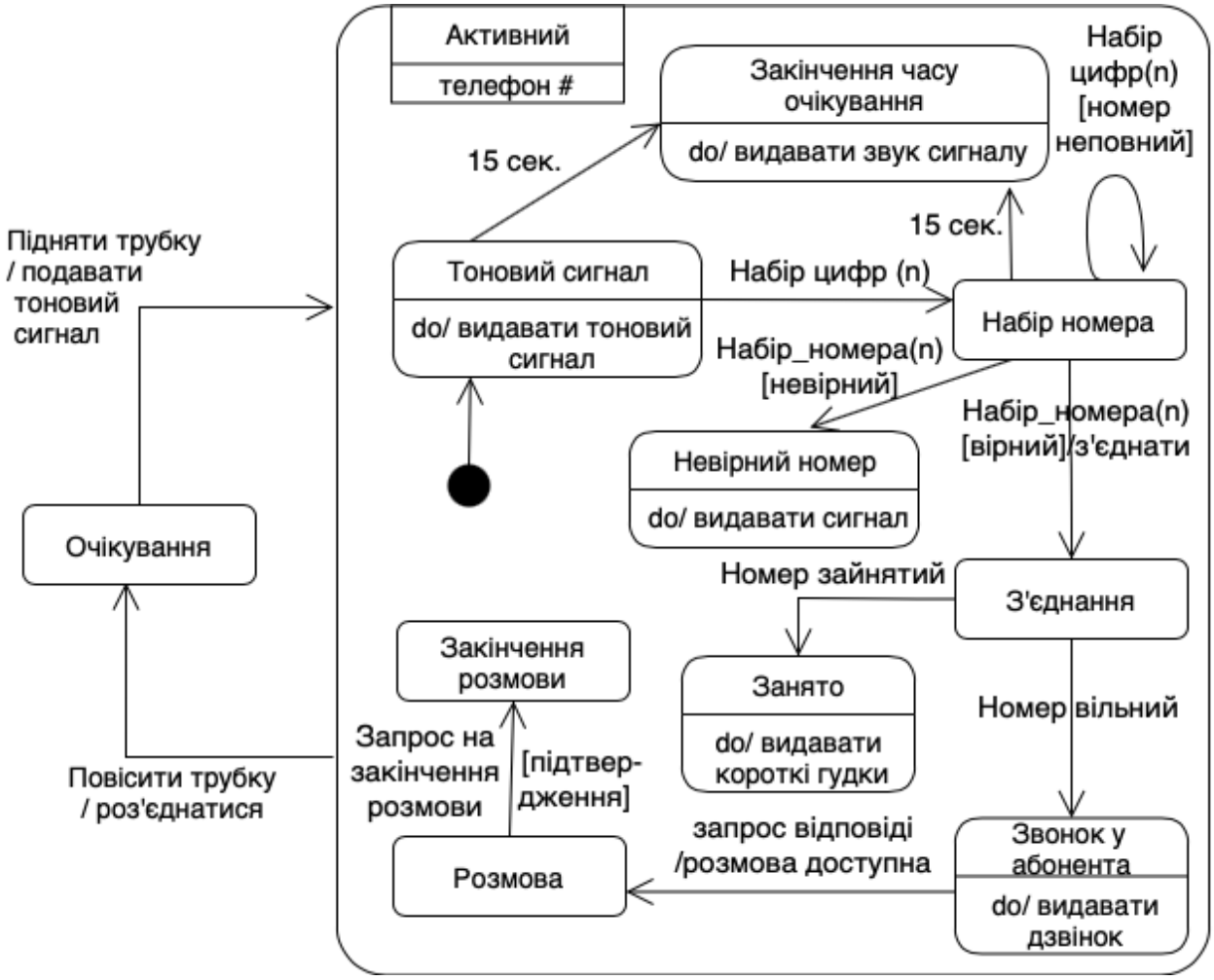



Рисунок 1.15 – Діаграма станів (дзвінок по телефону)

## Завдання на лабораторну роботу

1. Побудувати розгорнуту загальну діаграму станів для усієї системи. Загальних станів (що вміщують підстани) повинно бути мінімум три.

## Контрольні питання

1. Дайте визначення діаграми станів та її зміст. Охарактеризуйте особливості діаграмі станів для елементів UML.

2. Охарактеризуйте прості, складені, паралельні та послідовні стани з точки зору потоку управління. Поясніть, чим характеризуються переходи на діаграмі станів.

3. Наведіть види операцій які існують у діаграмі стану.

## ЛАБОРАТОРНА РОБОТА №4 Моделювання поведінки системи засобами UML

Мета роботи: дослідження діаграм UML, які застосовуються для опису поведінки програмного забезпечення, та отримання навиків у їх побудові.

### Короткі теоретичні відомості

Призначення діаграми діяльності (Activity diagram) [1] (відноситься до класу поведінкових діаграм) полягає у відображенні потоків робіт та операцій, за допомогою елементів, які надані у таблиці 9. Приклад фрагменту діаграми діяльності з послідовними, та паралельними потоками робіт представлений на рисунку 16. Діаграма діяльності може супроводжуватися додатковими специфікаціями, які місять детальний аналізі кожної дії (приклад у таблиці 10).

Таблиця 9 – Основні елементи діаграми діяльності

| Класифікатор | Функція | Нотація |
|---|---|---|
| Дія (activity) | Фрагмент дії |   |
| Перехід (transition) | Напрямок дії |  |
| Точка прийняття рішень (decision point, marge, union) | Умова для можливих альтернативних шляхів (розділяє та об’єднує потоки) |  |
| Лінії сінхронізації (synchronization bars, fork, join) | Об’єднують дії, які виконуються паралельно |  |
| Дані (data) | Передача даних між діями чи операціями |  |
| Початковий стан (start) | Початковий стан дій. Буває тільки один |  |
| Кінцевий стан (end) | Кінець дій. Може бути декілька |  |

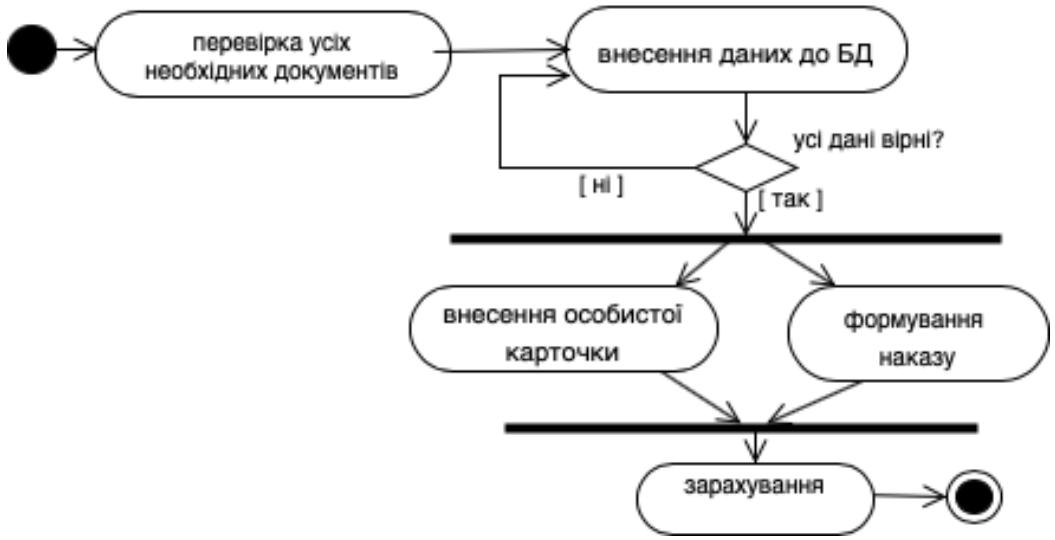



Рисунок 16 – Приклад опису дії відділу кадрів за допомогою діаграми діяльності

Таблиця 10 – Приклад опису специфікації діаграми діяльності

| Формулювання прецеденту | Стан виду діяльності |
|---|---|
| 1.Перевірка усіх документів | Перевірка наявних документів: паспорт, трудова книжка, фотографії, особистий листок і т. ін. |
| 2.Внесення даних до БД | Передбачає внесення даних про працівника до БД підприємства та присвоєння йому табельного номеру, розпорядку робочого дня, тарифної ставки оплати праці |

Якщо є необхідність віднести роботи чи операції до конкретного виконавця (частіше за все – актора), можна використовувати діаграму діяльності з доріжками, де у верхній частині пишуть виконавця (рисунок 17).

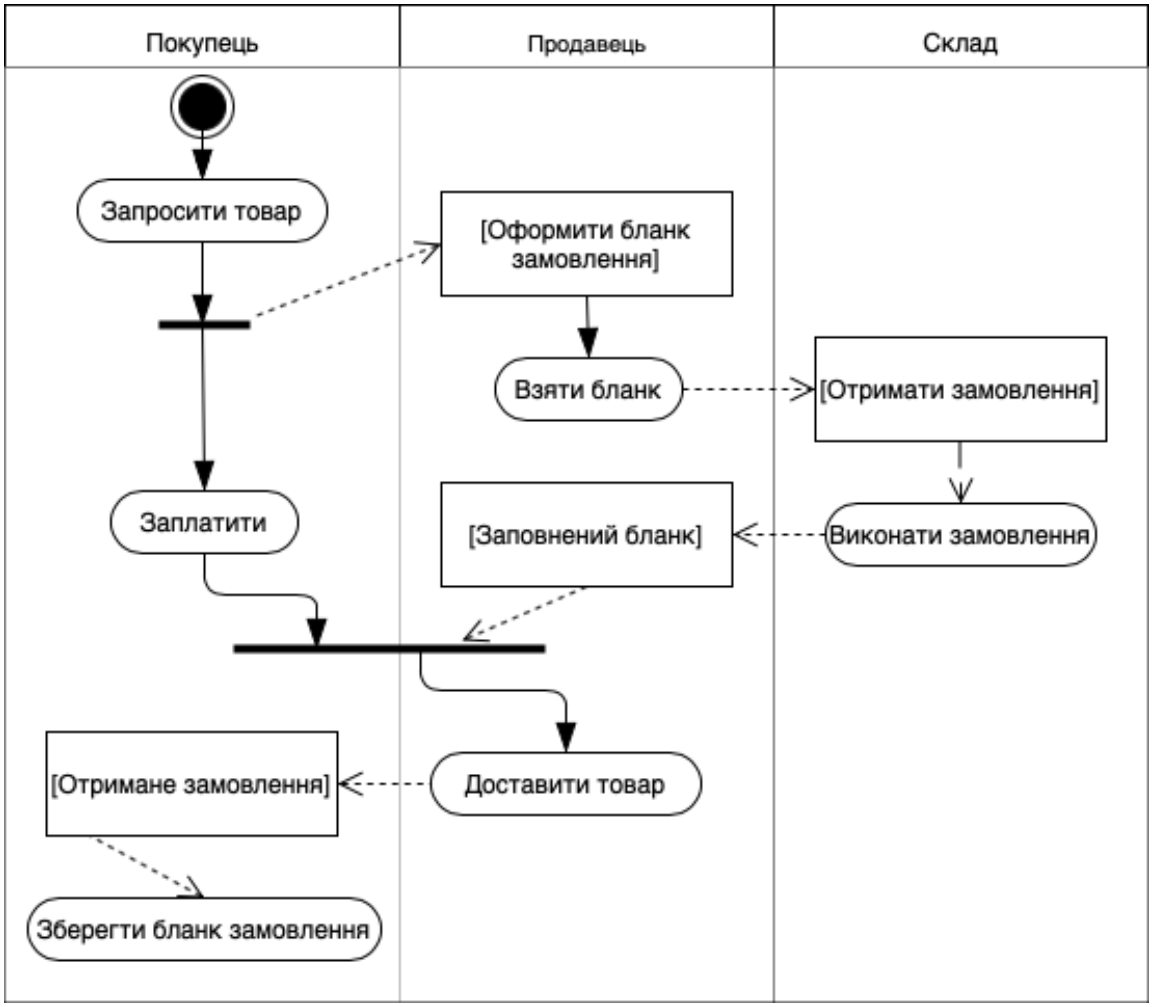



Рисунок 17 – Приклад діаграми діяльності з доріжками

Діаграми взаємодії (interaction diagrams) [1] (відносяться до класу поведінкових діаграм) використовують для моделювання взаємодії між об’єктами системи. Будуть розглянуті діаграми послідовності (Sequence diagram) та діаграми кооперації (Collaboration diagram).

Діаграма послідовності (sequence diagram) [1] – моделює взаємодію об’єктів (зазвичай екземплярів класів) у часі. Графічно об’єкт зображається прямокутником, у якому записують ім’я об’єкту та класу через двокрапку. Кожен об’єкт має лінію життя (object lifeline), яка зображується пунктирною вертикальною лінією та означає періоду часу, у якому об’єкт присутній у взаємодії. Непотрібні об’єкти знищуються за допомогою символу «X», після цього об’єкт припиняє існування у поточному процесі.

Об’єкт має змогу виконувати дію тільки отримавши на себе фокус керування (focus of control), зображений у формі витягнутого вузького прямокутника (рисунок 18).

Часові обмеження на діаграмі обов’язково позначити використовуючи наступний синтаксис:

– {час_прийому_повідомлення < 1 сек.};
– {час_простою_системи = 5 сек.};
– {час_передачі_пакету < 20 сек.}.

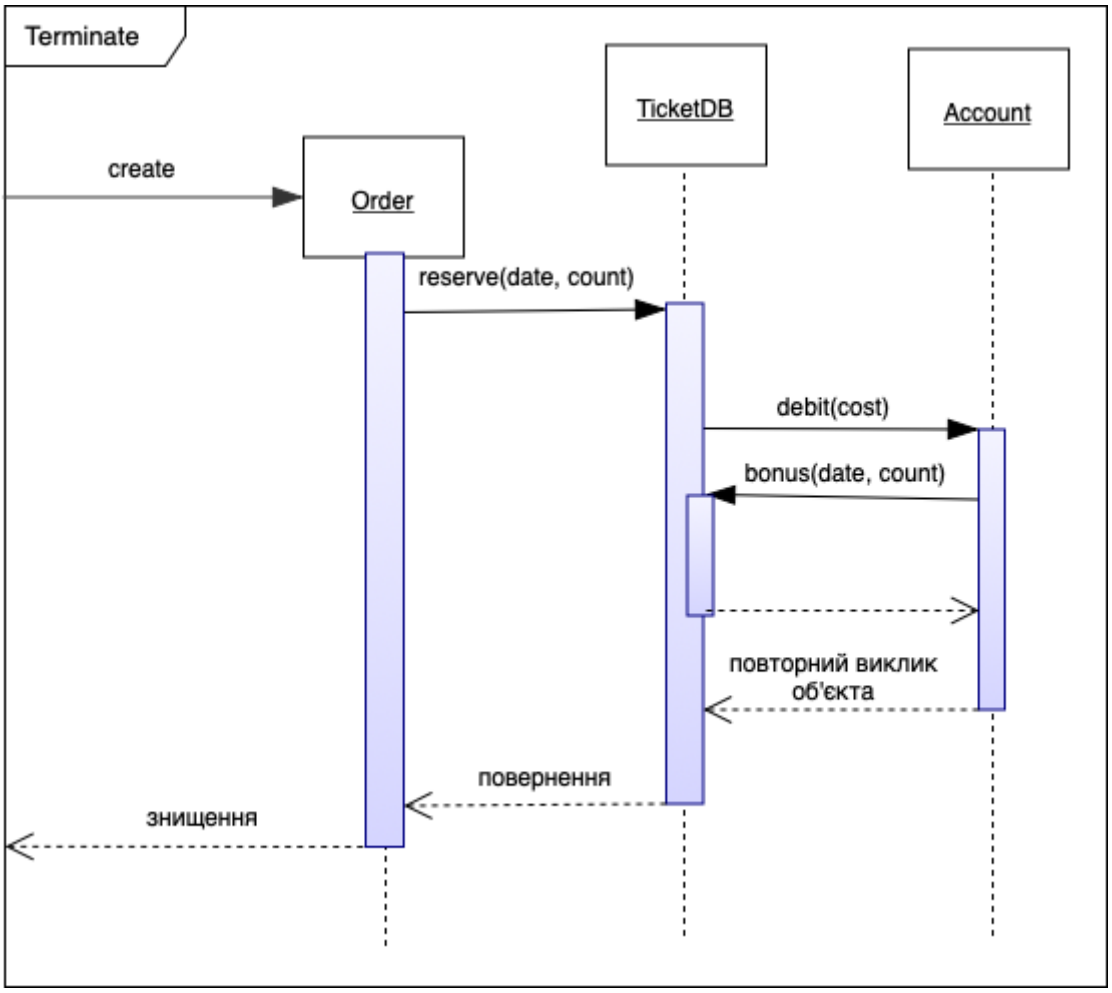



Рисунок 18 – Діаграма послідовності

Повідомлення можуть мати власне позначення операції. У цьому разі після імені операції записуються круглі дужки, в яких вказуються параметри операції. Якщо параметрів немає – записуються пусті дужки (наприклад: «встановити з’єднання між абонентами (a, b)», «створити графік робіт ()»).

Розгалуження потоку керування

Одна з особливостей діаграми послідовності – можливість візуалізувати розгалуження процесу. Для зображення розгалуження використовуються дві або більше стрілки, що виходять із однієї крапки фокусу управління об'єкта. Поруч із кожною з них у формі булевського виразу повинна бути явно зазначена відповідна умова гілки.

Кількість гілок може бути довільною, однак наявність розгалужень може істотно ускладнити інтерпретацію діаграми послідовності. Речення-умова повинне бути явно зазначене для кожної гілки, воно записується у формі звичайного тексту, псевдокоду або виразу на мові програмування. Цей вираз завжди має повертати деякий булевський вираз. Запис цих умов повинен виключати одночасну передачу альтернативних повідомлень по двом і більше гілкам. У противному випадку на діаграмі послідовності може виникнути конфлікт розгалуження.

Діаграма кооперації (collaboration diagram) [1] є інформаційно тотожною діаграмі послідовності, але акцентує увагу на відношеннях між об’єктами та відображає порядок операцій за допомогою порядкових номерів (рисунок 19).

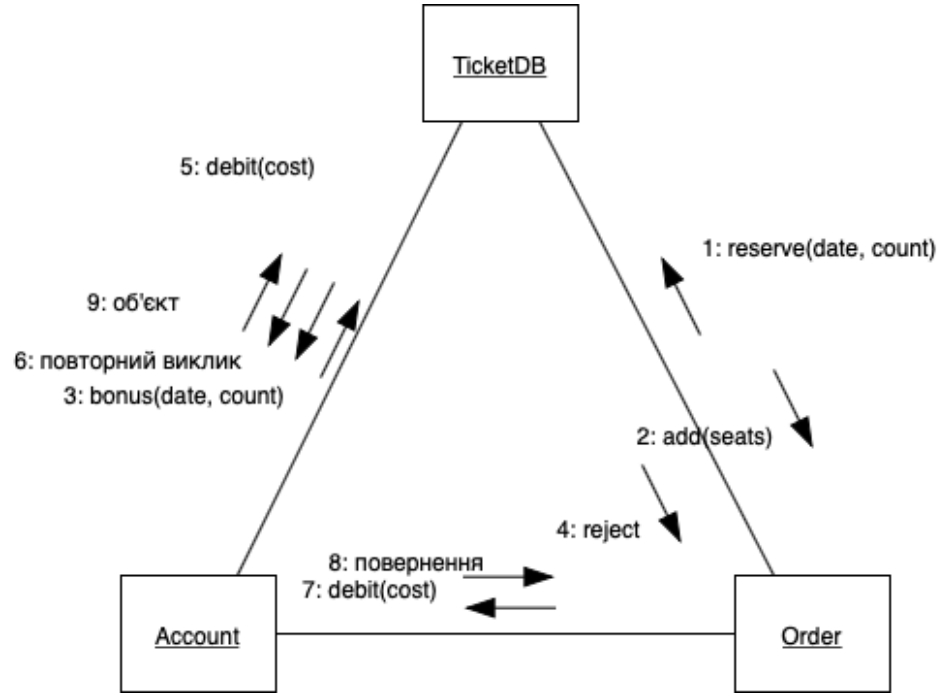



Рисунок 19 – Діаграма кооперації

**Завдання на лабораторну роботу**

1. Побудувати діаграми діяльності (мінімум 2 для різних типів діяльності, тобто діаграми не повинні описувати одне й те ж саме з різних точок зору!).

2. Специфікувати кожну дію в діаграмі діяльності.

3. Побудувати діаграми послідовності (мінімум 2 для різних типів діяльності, тобто діаграми не повинні описувати одне й те ж саме з різних точок зору!) .

**Контрольні питання**

1. Дайте визначення діаграми взаємодії, її зміст та способи побудови. Охарактеризуйте переваги та недоліки діаграм послідовності та кооперації.

2. Для чого використовуються „лінії життя” об’єкту та „фокус управління”?

3. Дайте загальну характеристику повідомлення та опишіть його види. Поясніть зв’язок діаграм послідовності та станів з точки зору повідомлень між об’єктами.

## ЛАБОРАТОРНА РОБОТА №5

### Проектування архітектури

Мета роботи: дослідження діаграм компонентів і розгортання та отримання навиків у їх використанні.

### Короткі теоретичні відомості

Діаграма компонентів (Component diagram) [1] описує фізичне представлення системи та забезпечує перехід від логічного представлення до реалізації проекту в формі програмного коду. *Компонент* є частиною фізичної реалізації системи (наприклад програмний модуль, інтерфейс користувача або база даних), який інкапсулює певний набір функціональних можливостей. З'єднувачі здійснюють взаємодію між компонентами. Компоненту дається ім’я, яке може складатися  з будь-якої кількості букв та цифр.

До стереотипів компонентів відносять наступне: модуль, що виконується (.exe); динамічна бібліотека (.dll); Web-сторінка (.html); база даних (DB); файл (.h, .cpp, .java, .pi, .asp та ін.); документ (.txt, .doc, .hlp).

Приклади діаграм компонентів з відображенням стереотипів у двох нотаціях наведені на рисунках 20 та 21.

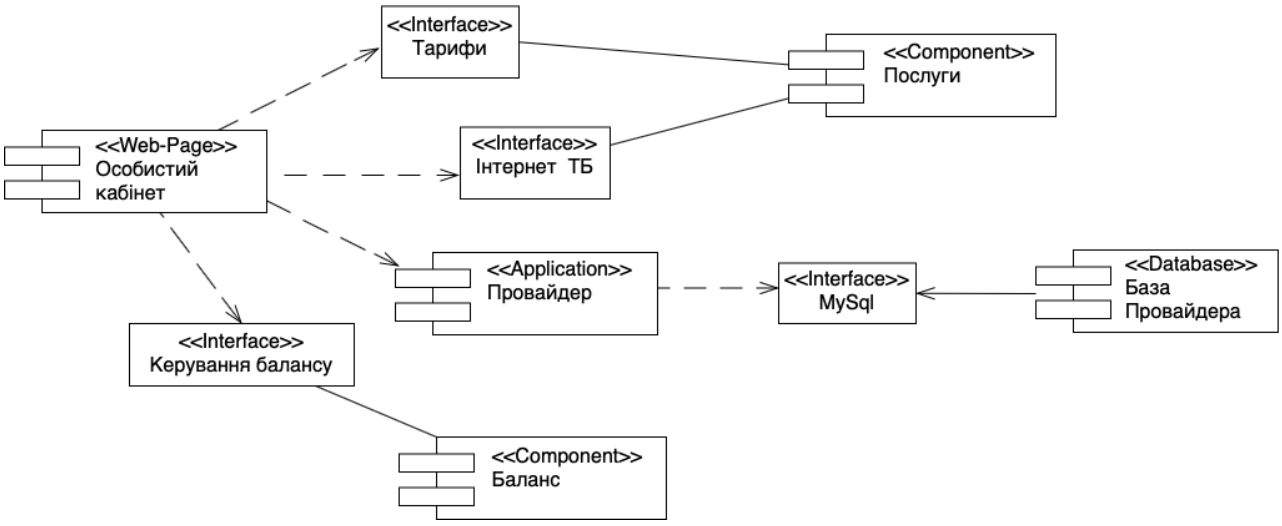



Рисунок 20 – Приклад діаграма компонентів з вказанами стереотипами

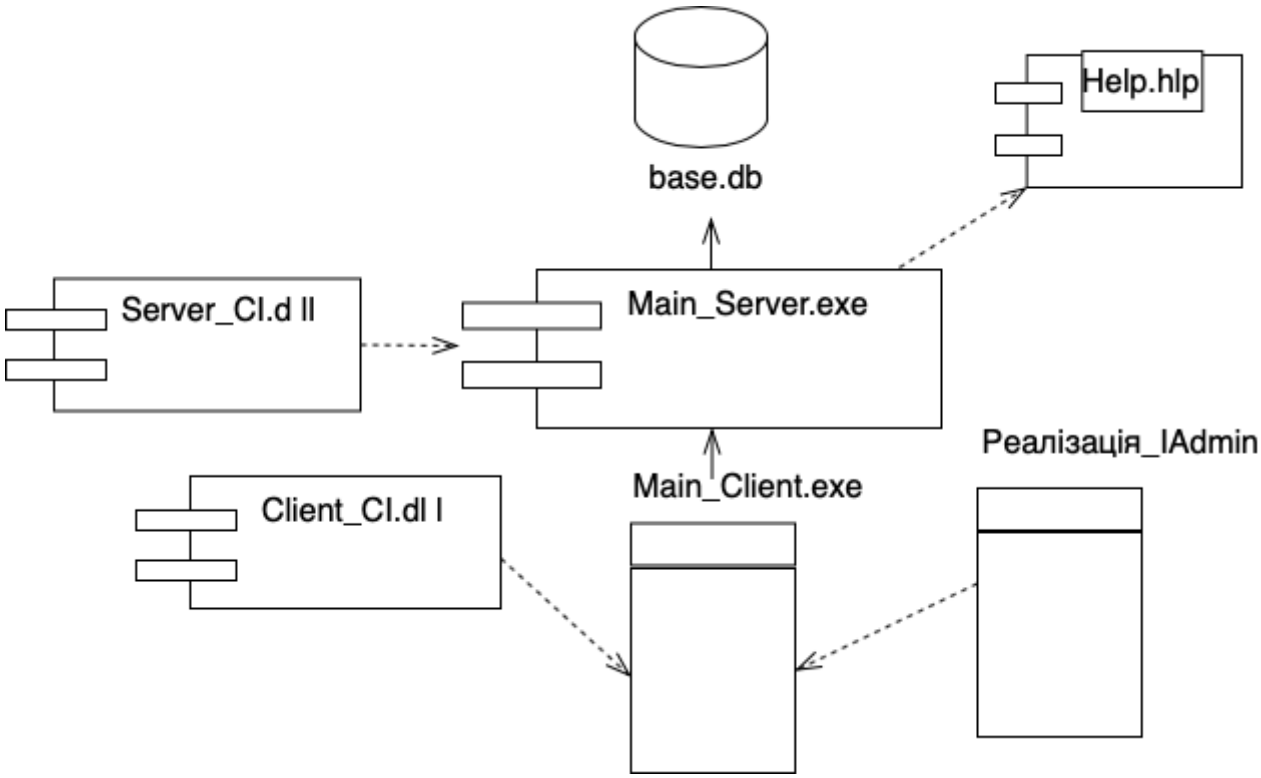



Рисунок 21 – Діаграма компонентів

Діаграма розгортання (Deployment diagram) [1] відображає загальну конфігурацію і топологію системи, фізичний взаємозв’язок між програмними та апаратними компонентами.

Діаграма розгортання має дві форми – ***екземпляра*** та ***дескриптора***. Перша описує місцезнаходження конкретних екземплярів компонентів на конкретних екземплярах вузлів. Форма дескриптора показує, як вузли можуть з’єднуватися.

Вузол – це фізичний об’єкт. Він може бути обчислювальним ресурсом (який має пам’ять, процесор і т. ін.), ресурсом механічної обробки даних, або людським ресурсом. Вузли можуть мати ємність, потужність, надійність, пропускну здатність. Виділяють вузли пристроїв та вузли середовища виконання.

Вузли пристроїв – це фізичні обчислювальні ресурси зі своєю пам'яттю та сервісами для виконання програмного забезпечення, такі як звичайні ПК, мобільні телефони.

Вузол середовища виконання – це програмний обчислювальний ресурс, який працює всередині зовнішнього вузла і який є сервісом, що виконує інші програмні елементи.

Зв’язки між вузлами показують канали, за допомогою яких відбуваються комунікаційні з’єднання.

Приклад діаграми розгортання наведено на рисунку 22.

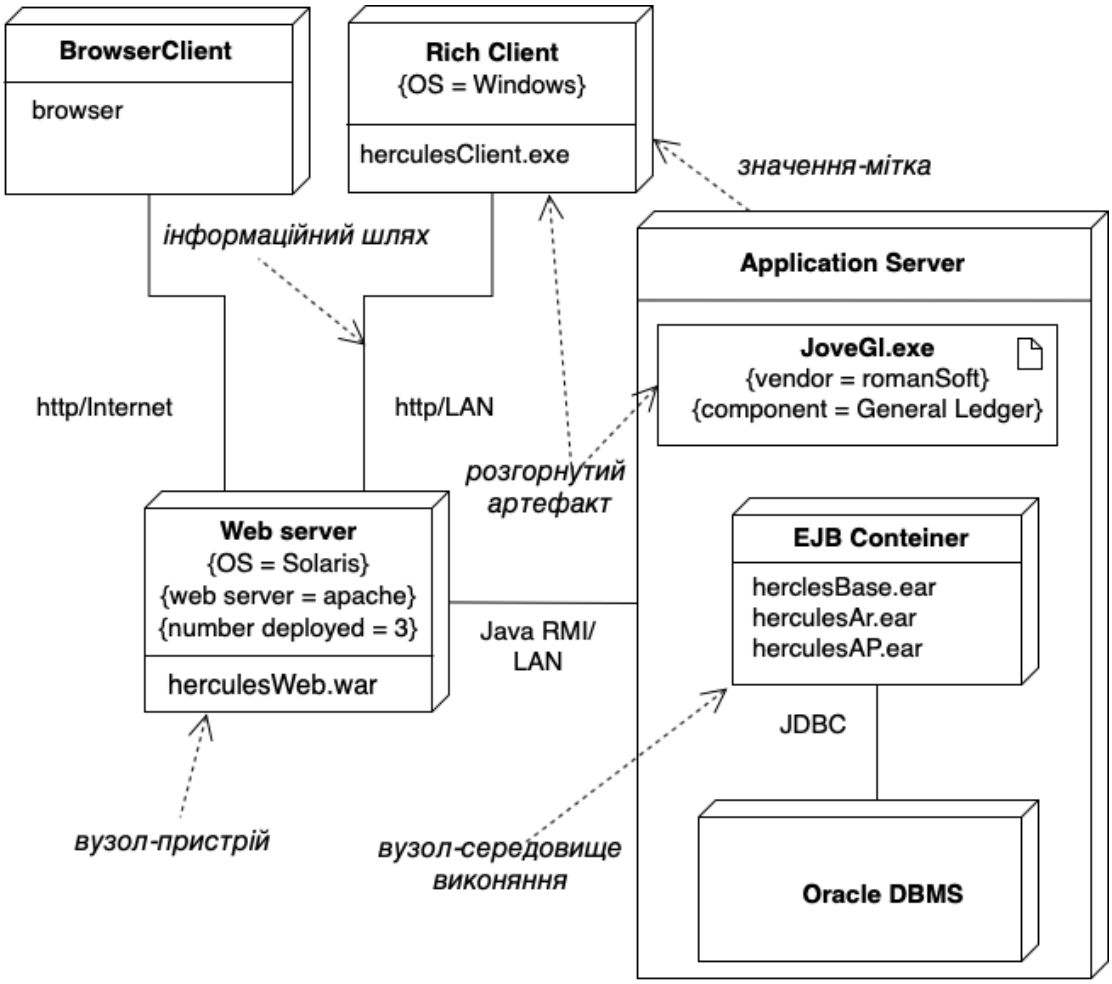



Рисунок 22 – Діаграма розгортання [10]

При потребі на діаграмі треба вказувати обмеження (семантична умова або обмеження, представлене у вигляді лінгвістичного виразу якоюсь текстовою мовою). Обмеження можна приєднати до будь-якого елемента або списку елементів моделі. Воно містить семантичну інформацію, яка відноситься до самого елемента моделі, а не лише до її представленню. У

кожного обмеження є тіло, і мова інтерпретації. Тіло є рядком з логічним виразом, що визначає умову мовою обмежень. Обмеження відноситься до списку впорядкованого з одного і більше елементів моделі. Зверніть увагу, що мовою специфікації може бути як формальна, так і природна мова (рисунок 23).

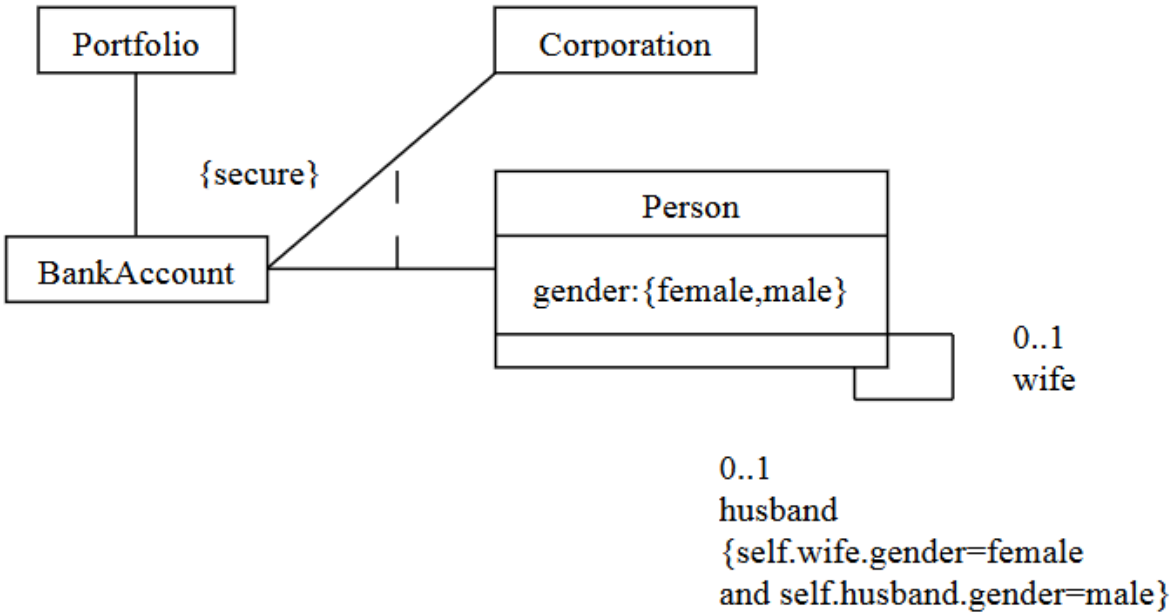



Рисунок 23 – Приклади обмежень

**Завдання на лабораторну роботу**

1. Побудувати діаграму компонентів для свого варіанту (використати мінімум 7 компонентів).

2. Побудувати дві діаграми розгортання для свого варіанту враховуючи одно- та багатокористувацьке використання системи.

**Контрольні питання**

1. Описати призначення діаграми компонентів.
2. Які є основні стереотипи компонентів? Яке їх призначення?
3. Описати призначення діаграми розгортання.
4. Який зв’язок між діаграмами компонентів та розгортання у контексті розробки програмного забезпечення?

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. UML 2.5.1 Офіційна нотація [Електронний ресурс]. – Режим доступу: https://www.omg.org/spec/UML/About-UML/

2. Using Rational RequisitePro. [Електронний ресурс]. – Режим доступу: http://pic.dhe.ibm.com/infocenter/rqmhelp/v2r0.

3. Лавріщева К,М. Програмна інженерія. .–К.– 2008.–319 с.

4. Jacobson I. Object-Oriented Software Engineering. A use Case Driven Approach, Revised Printing. – New York: Addison-Wesley Publ. Co., 1994. – 529 p.

5. М. Фаулер. UML. Основи. 3-е видання – 192с.

6. Кучеров, Д. П. Інженерія програмного забезпечення : навч. посіб. / Д. П. Кучеров, Є. Б. Артамонов. – Київ : НАУ, 2017. – 386 с. : іл.

7. Левус, Є. В. Вступ до інженерії програмного забезпечення : навч. посіб. / Є. В. Левус, Н. Б. Мельник. – Львів : Видавництво Львівської політехніки, 2018. – 246 с. : іл.

8. Бандура В.В., Храбатин Р.І. Архітектура та проектування програмного забезпечення: конспект лекцій. – Івано-Франківськ: ІФНТУНГ, 2012. — 240 с.

9. Сидоров М.О. Вступ до програмної інженерії. – К.:НАУ, 2008.- 65с.

10. Діаграма розгортання UML. Для чого використовується [Електронний ресурс]. – Режим доступу: https://planerka.info/item/diagrammy-razvertyvaniya-uml/

# ДОДАТОК А ТИТУЛЬНИЙ АРКУШ

**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

## Звіт

з лабораторної роботи № 1

з дисципліни

«Компоненті програмної інженерії 1»

**«Тема лабораторної роботи»**

Варіант ____

**Виконав(ла)** *ІП-15* _____
(шифр, прізвище, ім'я, по батькові)

**Перевірив** *Марченко О.І.* _____
(прізвище, ім'я, по батькові)

Київ 2022