МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ПРОМИСЛОВОЇ АВТОМАТИКИ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО
УНІВЕРСИТЕТУ»

Циклова комісія комп'ютерних наук та інженерії програмного забезпечення

Методичні вказівки

до виконання практичних робіт

з навчальної дисципліни

КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

обов'язкова

Освітньо-професійна програма «Комп'ютерні науки»

Код та найменування спеціальності 122 Комп'ютерні науки

Шифр та найменування галузі знань 12 Інформаційні технології

Мова навчання українська

2025

Розроблено та забезпечується: цикловою комісією Комп'ютерних наук та інженерії програмного забезпечення ВСП «Фаховий коледж промислової автоматики та інформаційних технологій Одеського національного технологічного університету»

Розробник:
Юлія БУРМАКІНА, викладач вищої кваліфікаційної категорії ФКПАІТ ОНТУ

Розглянуто та схвалено на засіданні циклової комісії Комп'ютерних наук та інженерії програмного забезпечення

Протокол № 1 від 27. 08. 2025 р.

Голова циклової комісії _____ Тетяна КОСТИРЕНКО
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

Методичні вказівки до виконання практичних робіт з дисципліни «Конструювання програмного забезпечення» розроблено згідно навчальної програми з предмету.

Предметом вивчення навчальної дисципліни є основи технології створення програмного продукту та уніфікована мова моделювання програмного забезпечення UML.

Метою вивчення навчальної дисципліни є оволодіння студентами теоретичних знань та набуття практичних навиків з дисципліни «Конструювання програмного забезпечення», необхідних для спеціальної підготовки та майбутньої професійної діяльності.

Основними завданнями вивчення дисципліни є формування у студентів базових уявлень про технологію створення програмного продукту та основи моделювання програмного забезпечення; інтелектуальний розвиток особистості, розвиток у студентів логічного мислення і просторової уяви, алгоритмічної, інформаційної та графічної культури, пам’яті, уваги, інтуїції; формування у студентів компетенцій за фахом.

Згідно з вимогами освітньо-професійної програми студенти повинні

Вміти:

– аналізувати предметну область на основі об'єктно-орієнтованої методології проектування;

– застосовувати основні методи та інструменти розроблення програмних продуктів;

– будувати діаграми у мові UML для формалізації опису предметної області, для якої розроблюється програмний продукт;

– складати специфікацій щодо вимог різних рівнів (вимоги замовника та розробника) на основі аналізу предметної області з використанням стандартної специфікації вимог до програмного продукту, що розробляється;

– проводити порівняльний аналіз процесів проектування і розробки програмних продуктів і робити обґрунтований вибір;

– виконувати формування та аналіз вимог для розроблення програмних продуктів;

– застосовувати методи моделювання для опису об'єктів інформатизації;

– аналізувати та моделювати бізнес-процеси, будувати регламенти зі створення комп'ютеризованих бізнес-процесів;

– ідентифікувати об'єкти системи, що проектується;

– розробляти діаграми динамічних та статичних аспектів інформаційної системи;

– розробляти діаграми взаємодії об'єктів інформаційної системи;

– розробляти технічну документацію на програмне забезпечення;

– застосовувати сучасні підходи до маркетингу програмних продуктів.

№	Тема практичної роботи
1.	Опис та аналіз предметної області.
2.	Побудова діаграми прецедентів (Use case Diagram)
3.	Побудова діаграми класів (Class Diagram)
4.	Побудова діаграми схем станів (Statechart Diagram)
5.	Побудова діаграми діяльності (Activity Diagram)
6.	Побудова діаграми послідовностей дій (Sequence Diagram)
7.	Побудова діаграми співпраці (кооперації) (Collaboration Diagram)
8.	Побудова діаграми компонентів (Component Diagram)
9.	Побудова діаграми розгортання (Deployment Diagram)
	Всього: 30 годин

Практична робота №1

Тема: Опис та аналіз предметної області.
Мета: Отримати практичні навики побудови діаграм Use Case.

Хід роботи:
I. Теоретичні відомості

Аналіз предметної області

Етап аналізу передбачає докладне дослідження бізнес-процесів (функцій, визначених на етапі вибору стратегії) і інформації, необхідної для їх виконання (сутностей, їх атрибутів і зв'язків (відношень)). На цьому етапі створюється інформаційна модель системи.

Вся інформація про систему формалізується і уточнюється на етапі аналізу. Особливу увагу слід приділити повноті переданої інформації, аналізу інформації на предмет відсутності протиріч, а також пошуку інформації, яка не використовується взагалі або дублюється.

Аналітики збирають і фіксують інформацію в двох взаємопов'язаних формах:

- функції - інформація про події та процеси, які відбуваються в бізнесі;
- сутності - інформація про речі, які мають значення для організації і про яких щось відомо.

Двома класичними результатами аналізу є:

- ієрархія функцій, яка розбиває процес обробки на складові частини (що робиться і з чого це складається);
- модель «сутність-зв'язок» (Entry Relationship model, ER- модель), яка описує сутності, їх атрибути і зв'язки (відношення) між ними.

Ці результати є необхідними, але не достатніми. До достатніх результатів слід віднести діаграми потоків даних.

Нижче розглядаються методології структурного аналізу, що застосовуються

частіше за все:

1. діаграми «сутність-зв'язок» (Entity-Relationship Diagrams, ERD), які служать для формалізації інформації про сутності і їхні відношення;
2. діаграми потоків даних (Data Flow Diagrams, DFD), які служать для формалізації представлення функцій системи.

Діаграми «сутність-зв'язок»

Діаграми «сутність-зв'язок» (ERD) призначені для розробки моделей даних і забезпечують стандартний спосіб визначення даних і відношень між ними. Фактично за допомогою ERD здійснюється деталізація сховищ даних проектованої системи, а також документуються сутності системи і способи їх взаємодії, включаючи ідентифікацію об'єктів, важливих для предметної області (сутностей), властивостей цих об'єктів (атрибутів) і їх відношень з іншими об'єктами (зв'язків).

Під ***сутністю (entity)*** розуміється довільна множина реальних або абстрактних об'єктів, кожен з яких володіє однаковими властивостями і характеристиками. У цьому випадку кожен даний об'єкт може бути екземпляром однієї і тільки однієї сутності, повинен мати унікальне ім'я або ідентифікатор, а також відрізнятися від інших екземплярів даної сутності. Прикладами сутностей можуть бути: банк, клієнт банку, комп'ютер, термінал. Кожна з сутностей може розглядатися з різним ступенем деталізації і на різному рівні абстракції, що визначається конкретною постановкою завдання. Для графічного представлення сутностей використовуються спеціальні позначення (рисунок 1).

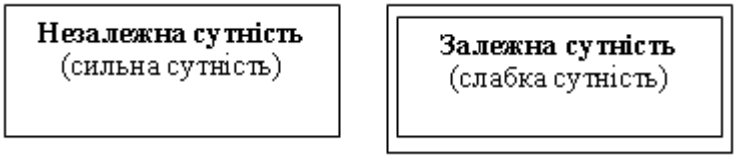


Рисунок 1 - Графічні зображення для позначення сутностей

Зв'язок (relationship) визначається як відношення або деяка асоціація між окремими сутностями. Прикладами зв'язків можуть бути родинні стосунки типу «батько-син» або виробничі відносини типу «начальник-підлеглий». Інший тип зв'язків задається відносинами «мати у власності» або «мати властивість». Різні типи зв'язків графічно зображаються у формі ромба з відповідним ім'ям даного зв'язку (рисунок 2).

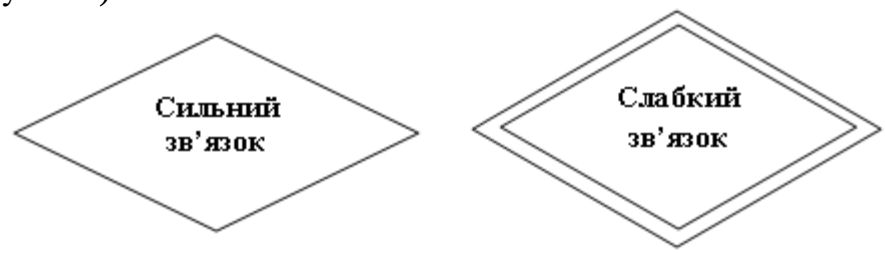


Рисунок 2 - Графічні зображення для позначення зв'язків

Діаграми потоків даних

Діаграми потоків даних являють собою інформаційну модель (DFD),

основними компонентами якої є різні потоки даних, які переносять інформацію від однієї підсистеми до іншої. Кожна з підсистем виконує певні перетворення вхідного потоку даних і передає результати обробки інформації у вигляді потоків даних для інших підсистем.

Основними компонентами діаграм потоків даних є:

- зовнішні сутності,
- накопичувачі даних або сховища,
- процеси,
- потоки даних,
- системи / підсистеми.

Зовнішня сутність є матеріальним об'єктом або фізичною особою, яка може виступати в якості джерела або приймача інформації. Прикладами зовнішніх сутностей можуть служити: клієнти організації, замовники, персонал, постачальники. Зовнішня сутність позначається прямокутником з тінню (рисунок 3), всередині якого вказується її ім'я. При цьому в якості імені рекомендується використовувати іменник у називному відмінку.

Рисунок 3 - Зображення зовнішньої сутності на діаграмі потоків даних

Процес являє собою сукупність операцій з перетворення вхідних потоків даних у вихідні відповідно до певного алгоритму або правила. Хоча фізично процес може бути реалізований різними способами, найбільш часто мається на увазі програмна реалізація процесу. Процес на діаграмі потоків даних зображується прямокутником із закругленими вершинами (рисунок 4), розділеним на три секції або поля горизонтальними лініями. Поле номера процесу служить для ідентифікації останнього. У середньому полі вказується ім'я процесу. Як ім'я рекомендовано використовувати дієслово в невизначеній формі з необхідними доповненнями. Нижнє поле містить вказівку на спосіб фізичної реалізації процесу.

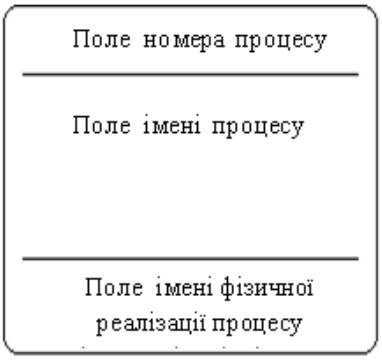


Рисунок 4 - Зображення процесу на діаграмі потоків даних

Накопичувач даних або сховище являє собою абстрактний пристрій або спосіб зберігання інформації, що перетікає між процесами. Передбачається, що дані можна в будь-який момент помістити в накопичувач і через деякий час витягнути, причому фізичні способи приміщення і вилучення даних можуть бути

довільними. Накопичувач даних на діаграмі потоків даних зображується прямокутником з двома полями (рисунок 5).

накопичувача

Рисунок 5 - Зображення накопичувача на діаграмі потоків даних Потік даних визначає якісний характер інформації, переданої через деяке з'єднання від джерела до приймача. Потік даних на діаграмі DFD зображується лінією зі стрілкою на одному з її кінців, при цьому стрілка показує напрямок потоку даних. Кожен потік даних має своє власне ім'я, що відбиває його зміст.

Таким чином, інформаційна модель системи в нотації DFD будується у вигляді діаграм потоків даних, які графічно подаються з використанням відповідної системи позначень.

II Завдання до практичної роботи №1

Згідно варіанту, виконати опис предметної області проектованої програмної системи. Провести об'єктний аналіз отриманого опису і побудувати модель середовища за допомогою діаграми потоків даних (аналіз поведінки системи) і діаграми «сутність-зв'язок» (аналіз даних). Визначити призначення проектованої ІКС.

III Зробити висновки по виконанню практичної роботи №1

IV Контрольні питання для підготовки до практичної роботи №1

1. Мета проведення об'єктного аналізу.
2. Призначення діаграми «сутність-зв'язок».
3. Основні елементи діаграми «сутність-зв'язок».
4. Призначення діаграми потоків даних.
5. Основні елементи діаграми потоків даних.

Практична робота №2

Тема: Побудова діаграми прецедентів (Use case Diagram).
Мета: Отримати практичні навики побудови діаграм Use Case.

Хід роботи:
І. Теоретичні відомості

Проектування - один із важливих кроків під час розробки програми, який дуже часто ігнорується розробниками-початківцями. Зазвичай вони намагаються утримати все в голові або, у кращому разі, записати деякі важливі відомості на аркуші паперу. Як результат, у них немає чіткого плану подальших дій, і проект може бути відкладений у довгий ящик.

Зазвичай під час проектування розробники зображують систему графічно, оскільки людині легко розібратися в такому поданні. Саме тому замість написання громіздких текстів про кожну можливість майбутньої програми розробники будують різні діаграми для опису своїх систем. Це допомагає їм не забувати, що потрібно реалізувати в програмі, і швидко вводити в курс справи своїх колег.

Сьогодні ми розберемося з тим, як використовувати діаграму варіантів використання UML (англ. "Unified Modeling Language") - стандартизована мова моделювання під час проектування програм.

Коли розробник створює свій додаток, він насамперед замислюється над двома питаннями:

- *Що робитиме додаток?*
- *Хто буде користуватися цим додатком?*

Деякими програмами може користуватися безліч людей, тому часто необхідно виділяти різні групи користувачів системи. У кожної такої групи можуть бути свої права і можливості в системі.

Для того щоб описати різні групи користувачів та їхні можливості в майбутній програмі, створюється так звана діаграма варіантів використання.

Діаграма варіантів використання (англ. use-case diagram) - діаграма, що описує, який функціонал розроблюваної програмної системи доступний кожній групі користувачів.

На сьогоднішній парі ми розберемо елементи цієї діаграми, які найчастіше застосовуються під час побудови, на безлічі невеликих прикладів діаграм і на прикладі однієї великої діаграми. Ця велика діаграма буде використовуватися під час проектування якоїсь програмної системи. В якості прикладу такої системи виберемо інформаційну систему для коледжу (можна розглядати її як сайт або як окремий додаток).

У цій системі можна виділити такі групи користувачів:

- Студенти
- Викладачі
- Класні керівники
- Заступники директора

Кожна з груп користувачів може користуватися нашою системою по-своєму.

Студенти можуть:

- Дивитися розклад
- Переглядати свої оцінки

Викладачі можуть:

- Розміщувати матеріали для пар зі своїх дисциплін
- Виставляти оцінки в електронний журнал

Класні керівники можуть робити все те ж саме, що і викладачі плюс:

- Складати розклад батьківських зборів

Заступники директора можуть:

- Складати розклад
- Публікувати пости з важливою інформацією

Крім того, у системи є функціонал, який доступний усім групам користувачів. У розроблюваній системі актуально буде додати месенджер, у якому можна буде швидко зв'язуватися з людиною, що цікавить. Отже, ця функціональність має бути доступна кожному користувачеві. Тобто всі користувачі можуть:

- Надсилати повідомлення

Вийшло багато пунктів, які може бути складно укласти в голові. Для того щоб швидко орієнтуватися в цих пунктах, треба навчитися будувати діаграми варіантів використання.

А чому ми описуємо так мало можливостей?

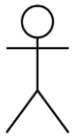
Зауважте, що на діаграмі треба відображати тільки ключовий функціонал системи. Наприклад, дії "увійти в систему", "вийти з системи" або "відновити пароль" можуть бути присутніми в будь-якій системі, і їхню наявність не варто додатково описувати, оскільки це забруднює діаграму несуттєвими елементами.

Взагалі додавання деяких дій на діаграмі залежить від глибини деталізації. Якщо вам усе ж таки потрібно зобразити деякі стандартні дії, ніщо не завадить швидко це зробити.

А тепер, коли були виділені групи користувачів і функціональність системи, можна починати будувати діаграму, щоб зафіксувати та структурувати отримані дані.

Побудова діаграми

Кожна група користувачів на діаграмі варіантів використання позначається чоловічком, під яким записується ім'я групи людей, яку він позначає. Давайте зобразимо групу користувачів "Викладачі":



Викладач

Цей чоловічок позначає всіх викладачів, які користуватимуться системою.

Зверніть увагу, що ім'я групи записується в однині. Символ чоловічка вже позначає групу користувачів, тому не потрібно додатково відображати це в імені.

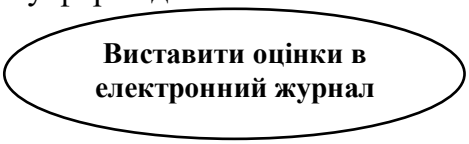
У термінології UML, цей чоловічок називається **актором** (англ. "actor"). У загальному випадку, актор позначає будь-які сутності, що використовують систему. Цими сутностями можуть бути люди, технічні пристрої або навіть інші системи.

Так само зобразимо акторів для решти груп користувачів:

Заст. директора Класний керівник Викладач Студент

На рисунку зображено всі групи користувачів, які можуть користуватися нашою системою

Кожна група користувачів використовує певні функції системи. На діаграмі варіантів використання функція системи зображується еліпсом, усередині якого записується ім'я функції у формі дієслова з пояснювальними словами.



Цей еліпс представляє дію "Виставити оцінки в електронний журнал"

У термінології UML, цей еліпс називається варіантом використання (англ. "use-case"). У загальному випадку, варіант використання - набір дій, який може бути використаний актором для взаємодії з системою.

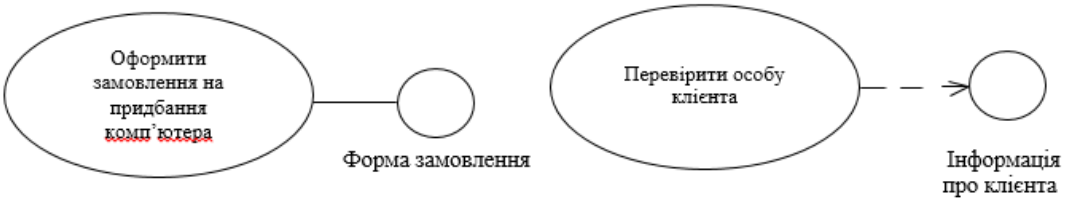
Інтерфейси

Інтерфейс (interface) служить для специфікації параметрів моделі, які видимі ззовні без вказівки їх внутрішньої структури. У мові UML інтерфейс є класифікатором і характеризує тільки обмежену частину поведінки сутності, яка моделюється. Стосовно до діаграм варіантів використання, інтерфейси визначають сукупність операцій, які забезпечують необхідний набір сервісів або функціональності для акторів. Інтерфейси не можуть містити ні атрибутів, ні станів, ні спрямованих асоціацій. Вони містять тільки операції без вказівки особливостей їх реалізації. Формально інтерфейс еквівалентний абстрактному класу без атрибутів і методів з наявністю тільки абстрактних операцій.

На діаграмі варіантів використання інтерфейс зображується у вигляді маленького кола, поруч із яким записується його ім'я. Якщо ім'я записується на англійському, то воно повинне починатися із заголовної букви I, наприклад, Isecurelnformation, Isensor.

Графічне зображення інтерфейсів на діаграмах варіантів використання

Графічний символ окремого інтерфейсу може з'єднуватися на діаграмі суцільною лінією з тим варіантом використання, який його підтримує. Суцільна лінія в цьому випадку вказує на той факт, що пов'язаний з інтерфейсом варіант використання повинен реалізовувати всі операції, необхідні для даного інтерфейсу, а можливо й більше. Крім цього, інтерфейси можуть з'єднуватися з варіантами використання пунктирною лінією зі стрілкою, що означає, що варіант використання призначений для специфікації тільки того сервісу, який необхідний для реалізації даного інтерфейсу.



Графічне зображення взаємозв'язків інтерфейсів з варіантами використання

Примітки

Примітки (notes) у мові UML призначені для включення в модель довільної текстової інформації, що має безпосереднє відношення до контексту розроблювального проекту. У якості такої інформації можуть бути коментарі розроблювача (наприклад, дата й версія розробки діаграми або її окремих компонентів), обмеження (наприклад, на значення окремих зв'язків або екземпляри сутностей) і позначені значення. Стосовно до діаграм варіантів використання примітка може носити саму загальну інформацію, що ставиться до загального контексту системи.

Графічно примітки позначаються прямокутником з "загнутим" верхнім правим куточком. Усередині прямокутника втримується текст примітки. Примітка може ставитися до будь-якого елемента діаграми, у цьому випадку їх з'єднує пунктирна лінія. Якщо примітка ставиться до декільком елементам, то від нього проводяться, відповідно, кілька ліній. Зрозуміло, примітки можуть бути присутнім не тільки на діаграмі варіантів використання, але й на інших канонічних діаграмах.

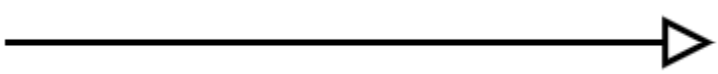
Приклади приміток у мові UML

Зв'язки між елементами

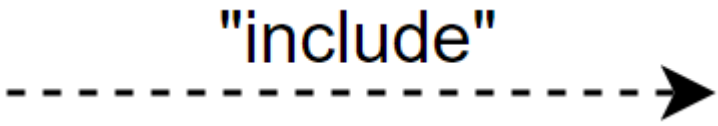
На діаграмах UML для зв'язування елементів використовуються різноманітні сполучні лінії, які називаються відношеннями. Кожне таке відношення має власну назву і використовується для досягнення певної мети.

Всі види відносин, які будуть використовуватися при роботі з діаграмою Use case.

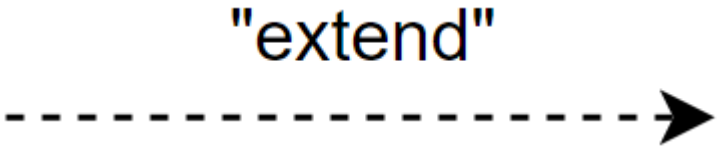
Відношення асоціації (англ. "association relationship")



Відношення узагальнення (англ. "generalization relationship")



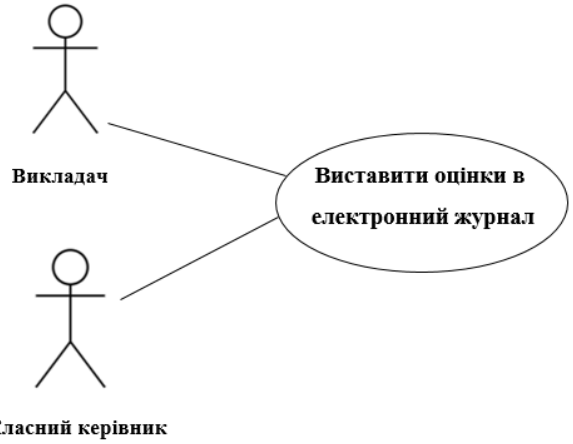
Відношення включення (англ. "include relationship")



Відношення розширення (англ. "extend relationship")

Відношення асоціації

Нам потрібно відображати на діаграмі інформацію про те, які варіанти використання можуть бути використані кожним актором. Зараз, наприклад, ми хочемо показати, що виставляти оцінки можуть тільки викладачі.

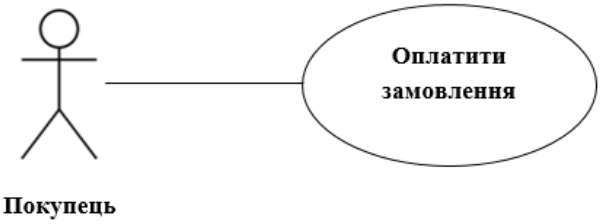


Зображуємо на діаграмі інформацію про те, що викладачі можуть виставляти оцінки

Ми з'єднали акторів із варіантом використання за допомогою суцільної лінії без стрілки. Така лінія називається ***відношенням асоціації***.

Відношення асоціації ("association relationship")

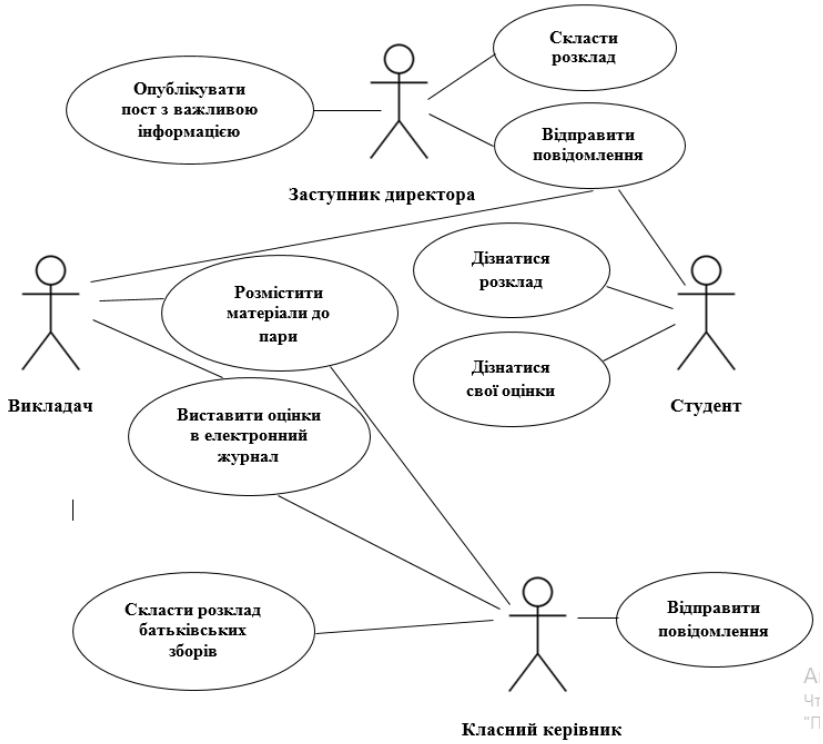
Відношення асоціації призначене **лише** для з'єднання акторів і варіантів використання. Немає жодного сенсу з'єднувати відношенням асоціації двох акторів або два варіанти використання.



Зображуємо на діаграмі можливість покупців оплачувати замовлення

Якщо на діаграмі варіантів використання актор з'єднаний із варіантом використання за допомогою відношення асоціації, це означає, що цей актор може виконувати дії, описані варіантом використання.

Додамо ще варіантів використання і з'єднаємо їх із відповідними акторами:



Перша версія діаграми

Поки що наша діаграма зовсім не вражає, тому продовжимо наповнювати її інформацією. Заодно дізнаємося всі можливості цього виду діаграм.

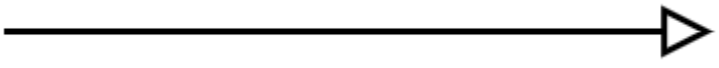
Відношення узагальнення

Зауважимо, що в нашій системі групи користувачів "Викладач" і "Класний керівник" мають схожі можливості. Щоб зобразити це на діаграмі, можна піти одним із трьох шляхів:

1. Дублювати варіанти використання, щоб пов'язати їх із кожним схожим актором (очевидно, невдалий варіант).
2. З'єднати кожного актора з усіма потрібними варіантами використання. Це може породити безліч перетинів ліній, що не найкращим чином позначиться на читабельності діаграми.
3. Показати за допомогою одного з видів відносин, що актори пов'язані між собою. Це означатиме, що один із них може користуватися всіма варіантами використання, з якими з'єднаний інший актор.

Останній варіант схожий на принцип повторного використання коду під час написання програм або на успадкування класів в об'єктно-орієнтованому програмуванні. Перевага цього варіанта в тому, щоб зменшити кількість зв'язків на діаграмі.

Зрозуміло, ми скористаємося третім шляхом. У цьому нам допоможе, так зване, *відношення узагальнення*. Відношення узагальнення позначається суцільною лінією з порожнистою трикутною стрілкою.

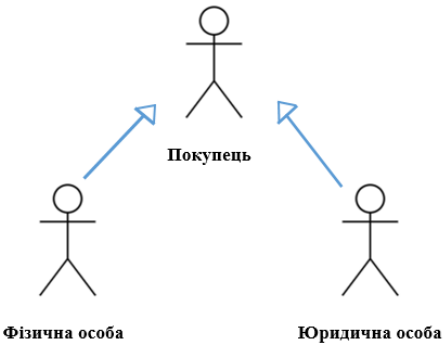


Відношення узагальнення (англ. "generalization relationship")

Відношення узагальнення означає, що деякий актор (варіант використання) може бути узагальнений до іншого актора (варіанту використання). Стрілка спрямована від окремого випадку (спеціалізації) до загального випадку.

Нижче наведено кілька прикладів використання відношення узагальнення.

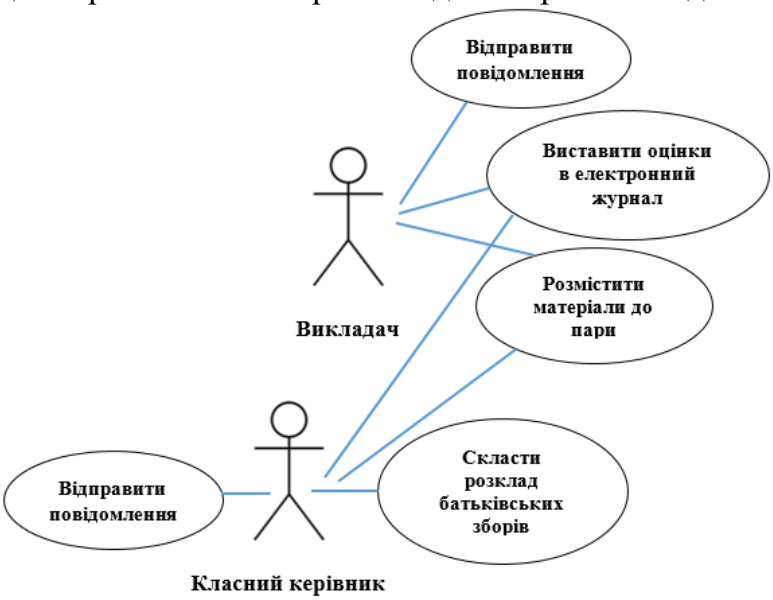
Купівля гірського і швидкісного велосипеда - ЧАСНИЙ випадок купівлі велосипеда



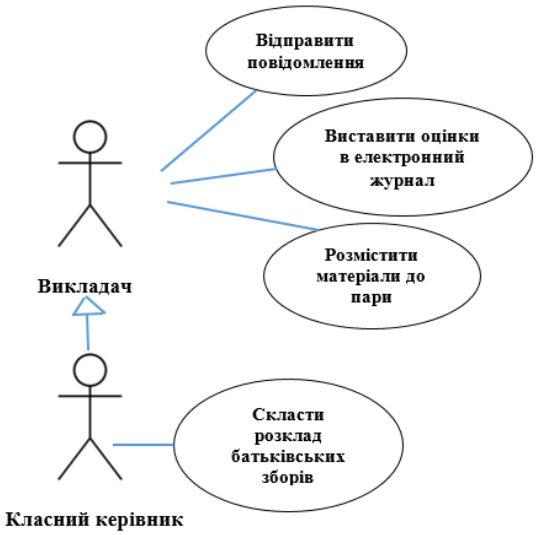
Фізичну особу та юридичну особу можна УЗАГАЛЬНИТИ до звичайного покупця

Як можна помітити, відношення узагальнення використовується, щоб показати, що одна дія є окремим випадком іншої дії або що одну групу людей можна узагальнити до іншої групи.

Повернімося до нашого основного прикладу. Зобразимо відношення узагальнення від актора "Класний керівник" до актора "Викладач".



До використання відношення узагальнення

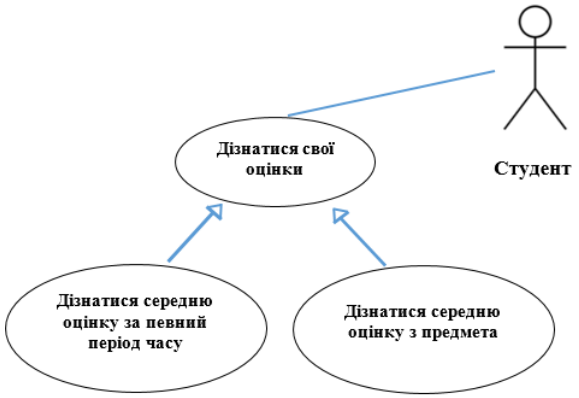


Після додавання відношення узагальнення

На рисунку зверху одразу видно, наскільки зрозумілішою стає діаграма під час використання відношення узагальнення: зникли всі повтори варіантів використання і перетини ліній. Зрозуміло, це величезний плюс для тих, хто читатиме цю діаграму надалі.

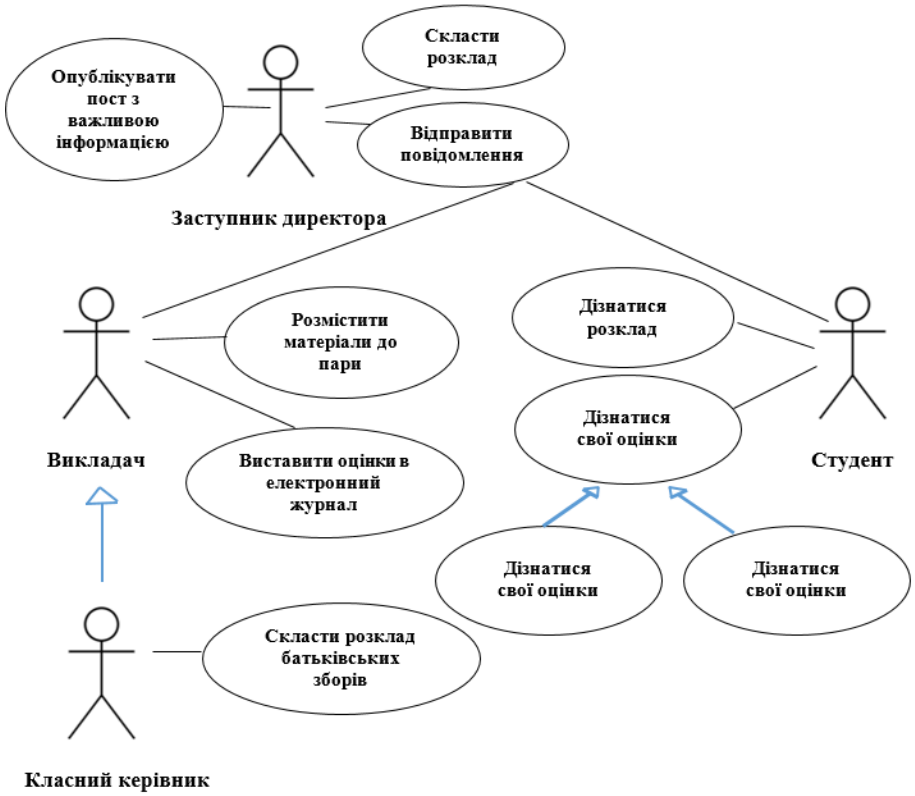
Давайте звернемо увагу на дію "Дізнатися свої оцінки". Логічно припустити, що студенти захочуть не тільки знати список своїх оцінок, а й знати свою середню оцінку за деякий період часу або середню оцінку з певного предмета.

Зобразимо це на діаграмі. Для цього створимо два варіанти використання "Дізнатися середню оцінку за певний період часу" і "Дізнатися середню оцінку з предмета" і з'єднаємо їх із варіантом використання "Дізнатися свої оцінки" відношенням узагальнення.



Уточнюємо на діаграмі, що студенти мають можливість дізнатися середню оцінку за деякий період часу та середній бал із деякого предмета

Приєднаємо це до основної діаграми:



Друга версія діаграми

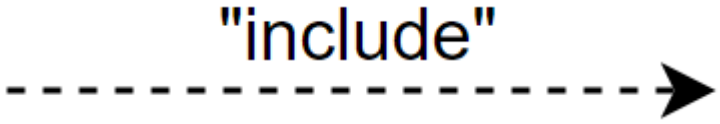
Відношення включення

Для заступника директора ми зазначали, що йому потрібно складати розклади. *Умовно* розклад можна поділити на три категорії:

1. Розклад занять
2. Розклад заходів
3. Розклад канікул

Усе це складається заступником директора, тому покажемо це на діаграмі. Для цього будемо використовувати ***відношення включення***.

Відношення включення позначається пунктирною лінією з V-подібною стрілкою на кінці, над стрілкою додається напис **"include"**.



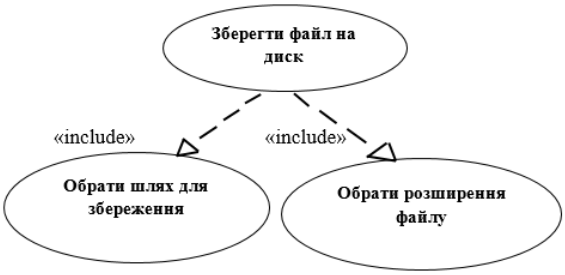
Відношення включення (англ. "include relationship")

У загальному випадку, відношення включення використовується, щоб показати, що деякий варіант використання включає в себе інший варіант використання як складову частину.

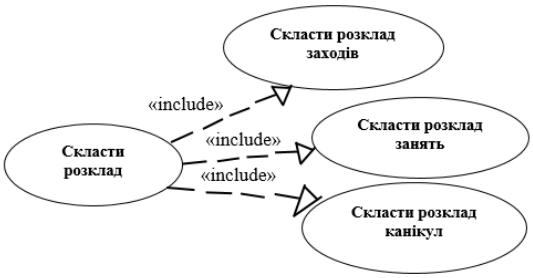
Коли ми використовуємо відношення включення, ми маємо на увазі, що складові варіанти використання ОБОВ'ЯЗКОВО входять до складу загального варіанта використання.

Пояснемо сенс і цього відношення на невеликому прикладі. Коли користувач зберігає результати своєї роботи у файл, він зазначає місце збереження і розширення файлу (наприклад, якщо він редагував фотографію в photoshop, він

може зберегти її в різних форматах). Цей процес можна зобразити на діаграмі варіантів використання таким чином:

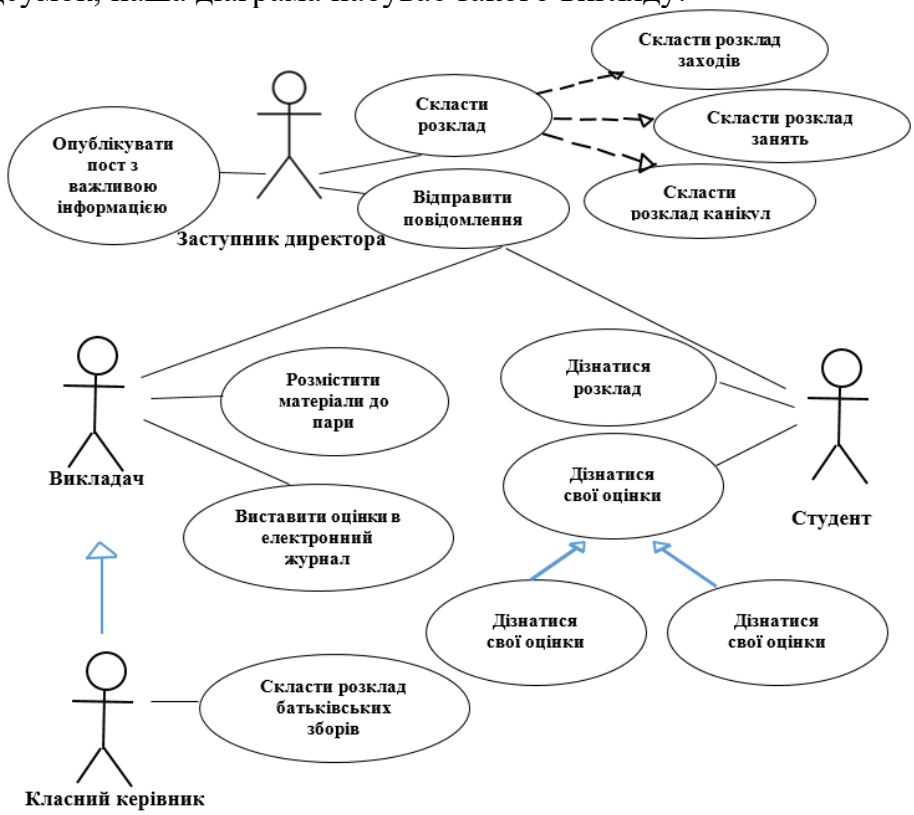


Відношення включення використовується для зображення складеної дії. Знову повернемося до нашого основного прикладу.



Складання розкладу ВКЛЮЧАЄ в себе складання розкладу занять, заходів, канікул (обов'язково).

Як підсумок, наша діаграма набуває такого вигляду:



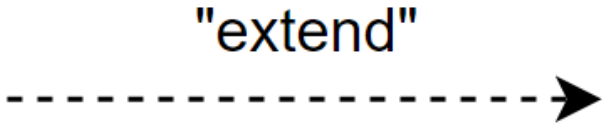
Третя версія діаграми

Загалом, на цьому можна зупинитися. Хоч наш приклад і демонстраційний, він трохи відображає функціональність реального застосунку. Проте залишився ще один елемент, який ми не розглянули.

Відношення розширення

Потрібно сказати, що в діаграмах варіантів використання застосовується ще один вид зв'язку - ***відношення розширення***. Застосування відношення розширення дещо специфічне, оскільки неправильне його використання може заплутати читача діаграми. Проте, для повноти картини ми все одно розглянемо застосування цього відношення на практиці. Востаннє модифікуємо нашу діаграму!

Під час дистанційного навчання студентам необхідно виконувати домашні завдання і надсилати їх у вигляді архіву або фотографій вчителям. Виходить, потрібно додати можливість прикріплювати файл до повідомлення в нашій системі. Щоб відобразити це на діаграмі треба використовувати відношення розширення. Відношення розширення позначається пунктирною лінією з V-подібною стрілкою на кінці (схоже на відношення включення), над стрілкою додається напис "extend ".



Відношення розширення (англ. "extend relationship")

Навіщо над пунктирними лініями додавати написи "include" і "extend"?

Щоб краще зрозуміти цей тип відносин, розглянемо приклад. Припустимо, ви робите замовлення в мережі швидкого харчування. Ви хочете замовити бургер. Вам, найімовірніше, запропонують розширити ваше замовлення картоплею фрі або соусом. Давайте зобразимо процес замовлення на діаграмі варіантів використання.

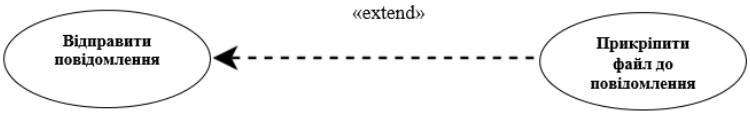
На діаграмі передбачається, що до замовлення МОЖЕ БУТИ додана картопля фрі або соус (необов'язково)

Два нижніх варіанти використання описують можливі "розширення" для базового варіанта використання. Виходячи з цього прикладу, ми можемо зробити важливе зауваження.

Можна сказати, що *відношення розширення* - це вибіркове відношення включення. Якщо відношення включення означає, що елемент обов'язково включається до складу іншого елемента, то у випадку відношення розширення це включення необов'язкове.

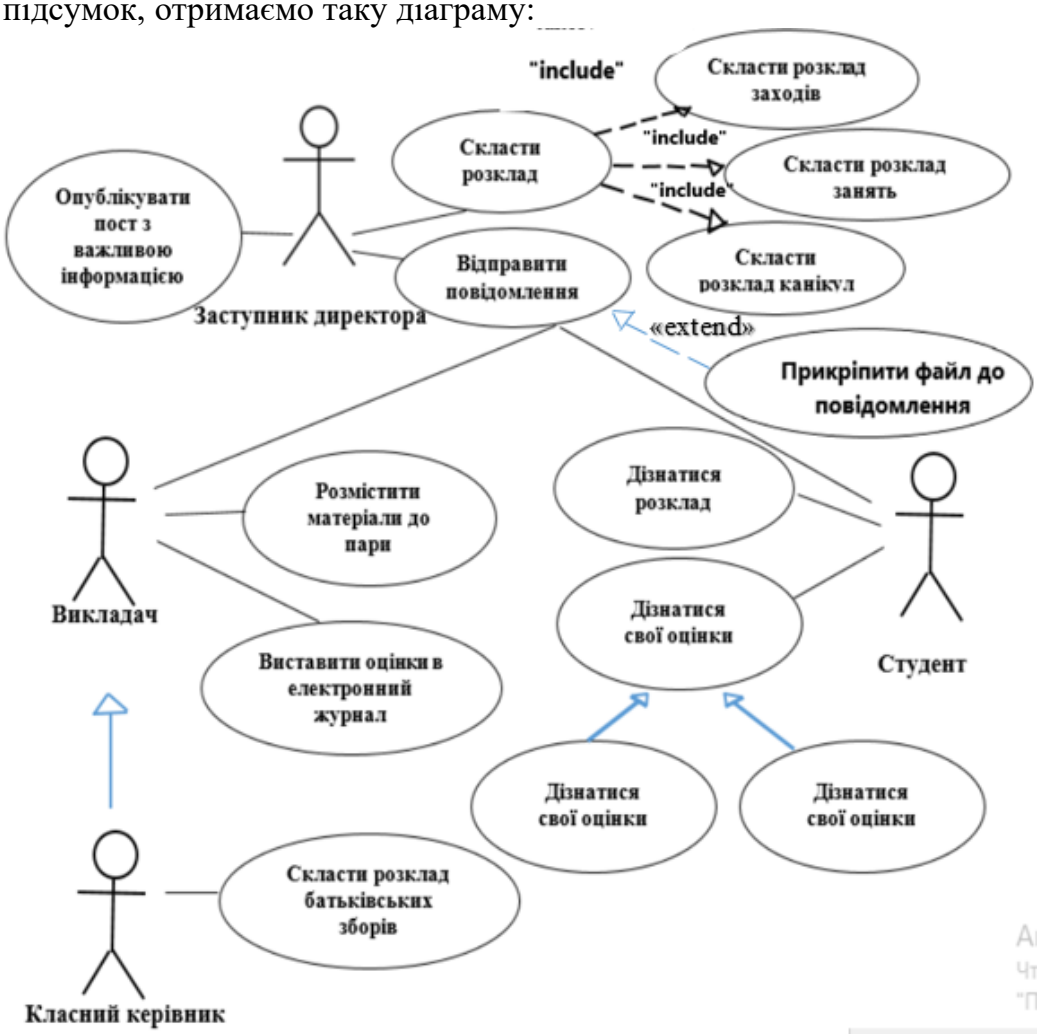
Розуміння цього критично важливе для грамотного використання цього виду відношень.

Повернемося до нашого основного прикладу. Ми хочемо, щоб дія "прикріпити файл до повідомлення" розширювала дію "надіслати повідомлення". На діаграмі це зображується таким чином:



Розширюємо функціонал надсилання повідомлень за допомогою функції прикріплення файлів до повідомлення (Необов'язково прикріплювати файл до кожного повідомлення)

Як підсумок, отримаємо таку діаграму:



Четверта версія діаграми

Хід роботи

Для демонстрації основних модулів було обрано туристичне агентство. Давайте проаналізуємо вступний опис і визначимо дані, які дійсно необхідні для нашої системи. Перед вами опис предметної області (важливі дані ми відзначатимемо маркерами: червоним - роль користувача, жовтим - важливі дії, які можуть здійснювати користувачі).

При першому зверненні клієнта менеджер реєструє його в системі.

Підбір туру виконує менеджер у системі відповідно до отриманої інформації від клієнта і має включати такі пункти:

1. вибір дат туру
2. зазначення вподобань клієнта

3. зазначення верхньої та нижньої меж вартості
4. вибір готелю

Після вибору відповідного туру менеджер може зареєструвати заявку на клієнта. За бажання клієнта менеджер може включити в заявку додаткові послуги, пропоновані турагенством і доступні в рамках обраного туру. Надалі і клієнт, і менеджер зможуть відстежувати актуальну інформацію щодо конкретної заявки на тур.

Кожному клієнту необхідний ваучер на трансфер, ваучер на заселення в готель, квиток на літак, страховий поліс, віза - всі документи клієнт може зберегти на свій пристрій. Подати запит на формування ваучерів може менеджер туристичного агентства.

Після поїздки клієнт може залишити відгук про готель.

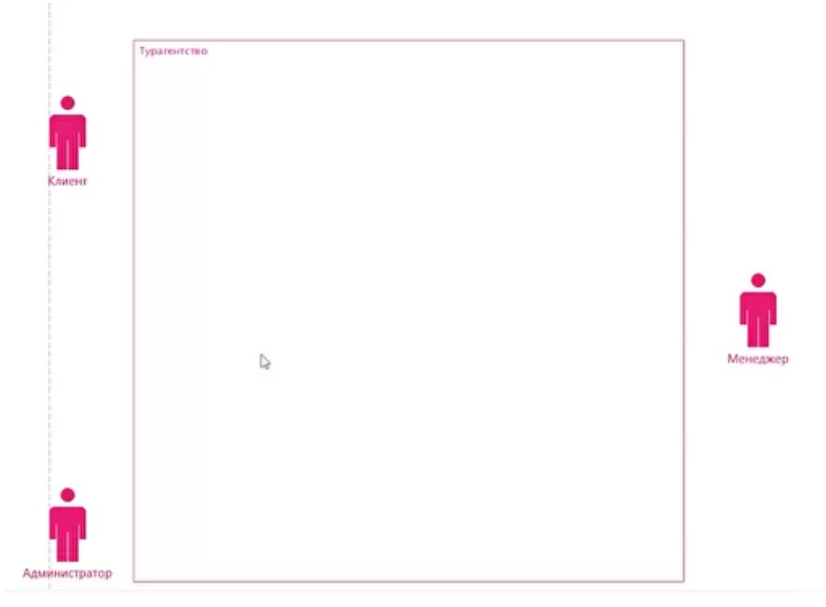
Отже, ми виділили:

- **Адміністратор** - створення нових турів і редагування наявних
- **Менеджер** - реєстрація клієнта в системі
- **Менеджер** - підбір туру для клієнта + 4 додаткові дії
- **Менеджер** - реєстрація заявки на клієнта + включення в заявку додаткових послуг
- **Клієнт і менеджер** - відстеження актуальної інформації за заявкою
- **Клієнт** - збереження ваучерів на свій пристрій
- **Менеджер** - подання запиту на формування ваучерів
- **Клієнт** - можливість залишити відгук про готель

Створення діаграми для турагенства

1. визначення основних груп користувачів (ролей) і розміщення на діаграмі

Це ті, хто використовуватиме систему, і в нашому випадку, як випливає з технічного завдання, - це клієнт, менеджер і адміністратор. Після розміщення буде наочно видно, що різні групи користувачів мають доступ тільки до певного функціоналу.



2. визначення варіантів використання (прецедентів), розміщення їх на діаграмі

А) **Для адміністратора:**

- створити новий тур
- редагувати наявний тур

Б) **Для менеджера:**

- зареєструвати клієнта
- підібрати тур: вибрати дати туру, вказати переваги клієнта, вказати межі вартості, вибрати готель
- зареєструвати заявку: вибрати додаткові послуги
- сформувати ваучер

В) **Для клієнта і менеджера:**

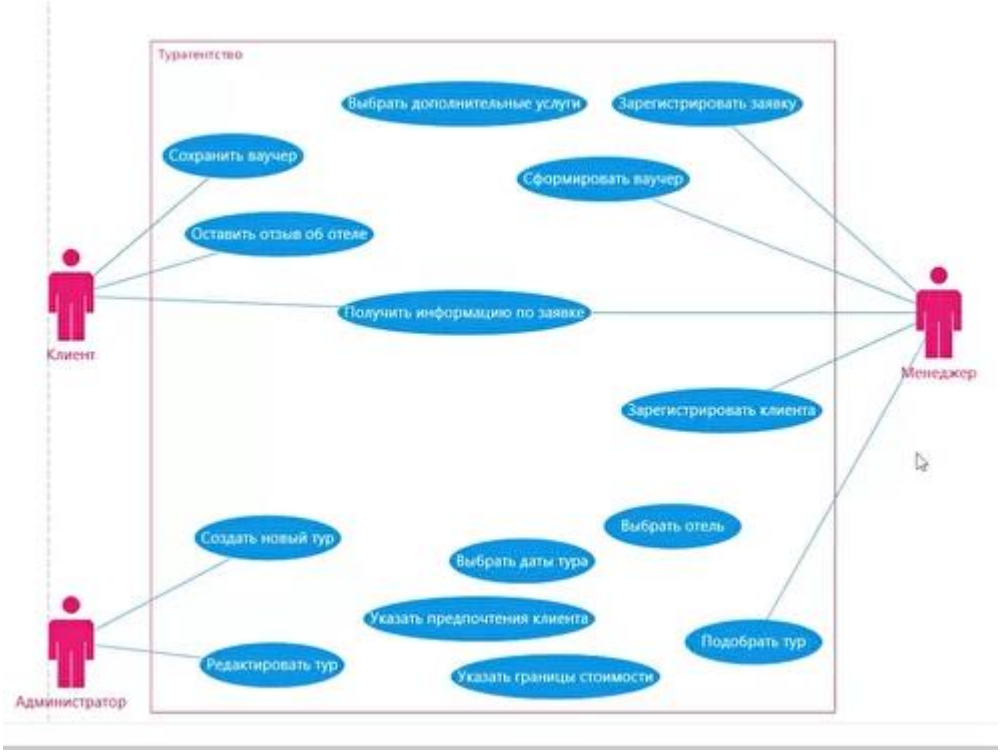
- отримати інформацію за заявкою

Г) **Для клієнта:**

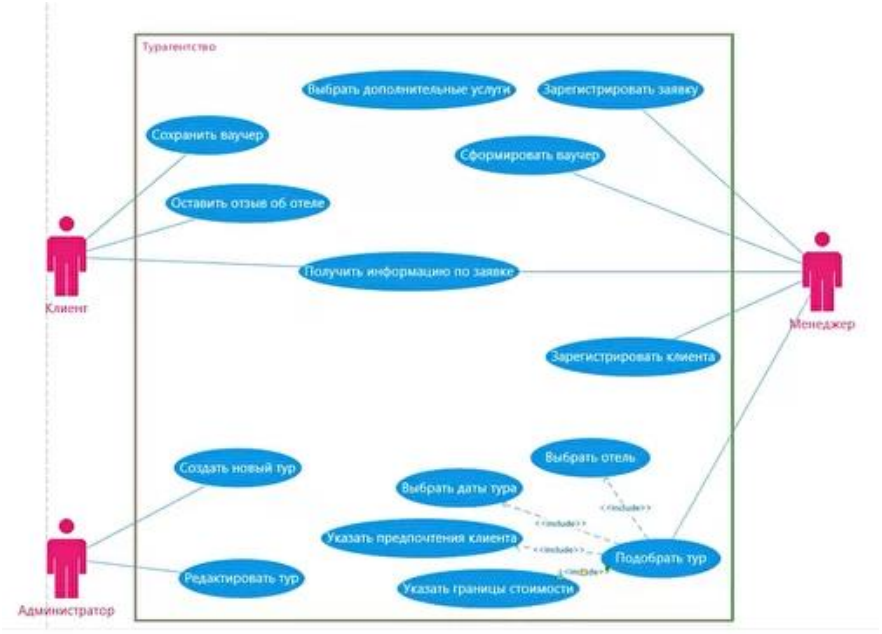
- зберегти ваучер на пристрій
- залишити відгук про готель

Розмежування прецедентів між акторами і розміщення відношень

- *Відношення асоціації* - відображає можливість використання актором прецеденту.

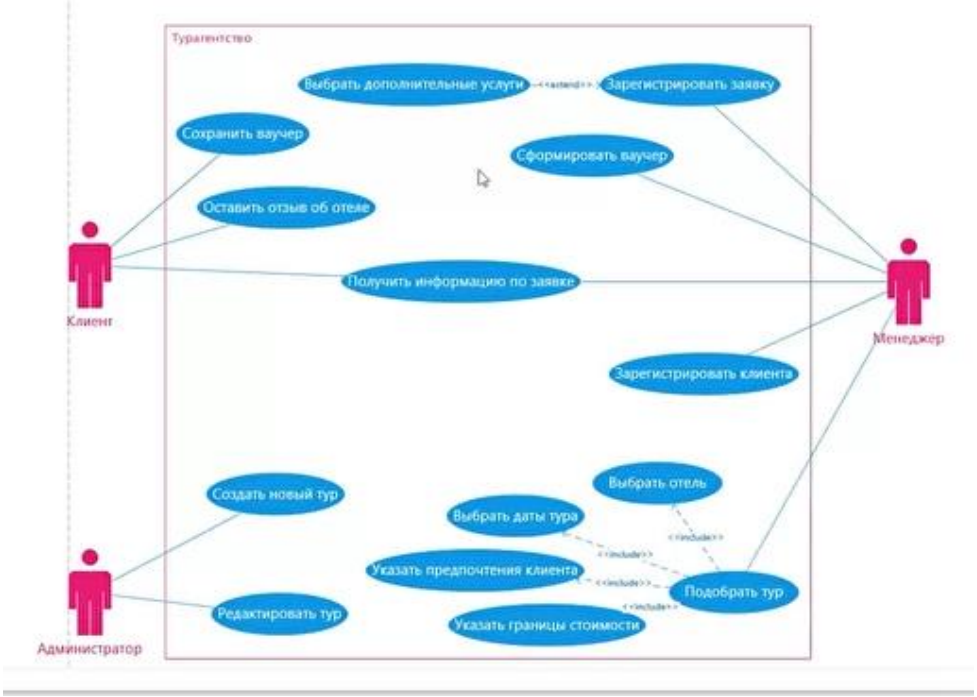


- *Відношення включення* - поведінку одного прецедента включають в інший як складову, до того ж варіант використання, що доповнюється, не зможе виконуватися без основного.



-

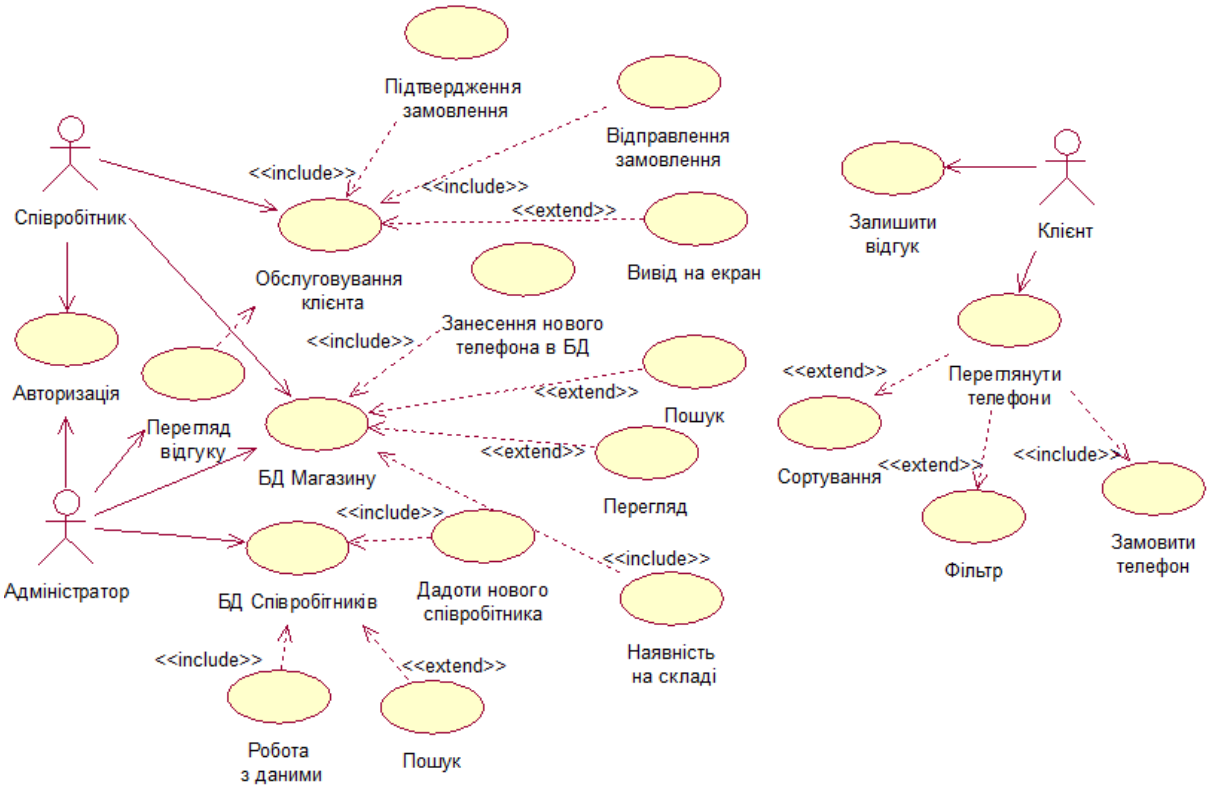
- *Відношення розширення* - відображає можливе приєднання одного використання до іншого, водночас варіант використання, що розширює, виконується лише за певних умов і не є обов'язковим для виконання основного прецеденту



Розглянемо приклад, а саме: діаграма прецедентів для роботи магазину мобільних телефонів.

Дана діаграма складається з:

- прецедентів “Авторизація”, “Обслуговування клієнта”, “БД Магазину” та актору “Співробітник”.
- прецедентів “Авторизація”, “Перегляд відгуку”, “БД Магазину”, “БД Співробітників” та актору “Адміністратор”.
- прецедентів “Залишити відгук”, “Переглянути телефони” та актору “Клієнт”.



II Завдання до практичної роботи №2

За власною предметною областю побудуйте діаграму прецедентів (мінімум з точок зору 2-х акторів). При побудові діаграми намагайтеся використовувати всі види відношень.

III Зробити висновки по виконанню практичної роботи №2

IV Контрольні питання для підготовки до практичної роботи №2

1. З яких елементів складається діаграма Use Case?
2. Для чого застосовують діаграми Use Case?
3. Які відносини дозволені між елементами діаграми Use Case? Описати призначення кожного з них та як вони зображуються на діаграмі.

Практична робота №3

Тема: Побудова діаграми класів (Class Diagram)
Мета: Отримати практичні навики побудови діаграми класів.

Хід роботи:
І. Теоретичні відомості

Діаграма класів (*class diagram*) - статичне представлення структури моделі. Відображає статичні елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів є ключовим елементом редактора UML-діаграм, оскільки часто додатки генеруються саме з діаграми класів

На відміну від діаграми послідовностей, діаграми діяльності тощо, діаграма класів є найпопулярнішою діаграмою UML.

Переваги діаграми класів:

- ❖ Будь-яку просту або складну модель даних можна проілюструвати за допомогою діаграми класів для отримання максимальної інформації.
- ❖ Схематику програми можна зрозуміти за допомогою неї.
- ❖ Будь-яка потреба в системі може бути візуалізована та передана по всьому бізнесу для конкретних дій.
- ❖ Будь-яка вимога щодо реалізації конкретного коду може бути виділена за допомогою діаграм і запрограмована до описаної структури.
- ❖ Опис, незалежний від реалізації, може бути наданий та переданий компонентам.

Недоліки діаграми класів:

- ❖ Діаграми класів часто можуть зайняти багато часу
- ❖ Складна діаграма не допомагає розробникам програмного забезпечення в їх роботі. Можуть виникнути ситуації, коли розробники засмучуються через структуру діаграм класів.
- ❖ Розміщення конструкції може перешкоджати розробникам та компаніям.

Незважаючи на важливість діаграми класів у життєвому циклі розробки програмного забезпечення, вона, звичайно, не позбавлена недоліків і може ускладнити життя розробникам та компаніям.

Діаграма класів може бути поділена на три компоненти:

1) Верхній розділ, який складається з назви класу, і є обов'язковим компонентом.
2) У середньому розділі описані якості класу та використовуються під час опису конкретного екземпляра класу.
3) У нижньому розділі описано взаємодію класу з даними.

На діаграмах класів зазвичай представлені такі елементи:

- ✓ Класи;
- ✓ Інтерфейси;
- ✓ Залежності, узагальнення та асоціації

Також діаграми можуть включати в себе пакети або підсистеми; ті та інші групують елементи моделі у більш великі утворення. Іноді в діаграму класів потрібно додати примірники.

Класи

Клас в мові UML служить для позначення множини об’єктів, які мають однакову структуру, поведінку і відносини з об’єктами інших класів.

Графічно клас зображується у вигляді прямокутника, який додатково може бути розділений горизонтальними лініями на розділи або секції. У цих розділах можуть зазначатися ім’я класу, атрибути (змінні) та операції (методи).

Ім’я класу має бути унікальним в межах пакету, який описується деякою сукупністю діаграм класів або однією діаграмою.

Ім'я вказується в самій верхній секції прямокутника, тому вона часто називається секцією імені класу.

Ім'я класу записується в центрі секції імені напіжирним шрифтом і повинне починатися з великої літери.

Для позначення імені абстракного класу використовується курсив

У деяких випадках необхідно вказати, до якого пакета відноситься клас. Для цього використовується спеціальний символ розподільник – подвійна двокрапка – (::)

Синтаксис рядка буде таким <Ім’я пакету>::<Ім’я класу>

Наприклад **Банк::Рухунок**

Атрибути класу

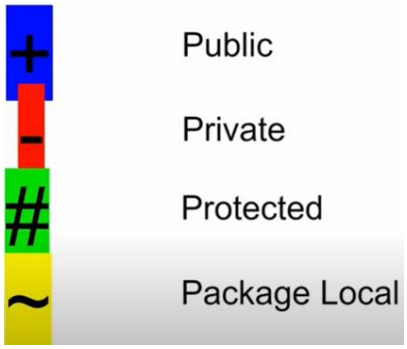
Ім’я атрибута – рядок тексту, який використовується в якості ідентифікатора відповідного атрибута і тому повинний бути унікальним в межах даного класу.

Атрибути класу або властивості записуються у другій зверху секції прямокутника класу. В UML кожному атрибуту класу відповідає окремий рядок тексту, який складається з квантора видимості атрибута і , можливо, початкового значення:

<квантор видимості><ім’я атрибута>[кратність]: <тип атрибута> = <початкове значення>

{рядок властивості}

Квантор видимості може приймати одне із можливих значень і відображається за допомогою відповідних спеціальних символів:



Квантор видимості може приймати одне із можливих значень і відображається за допомогою відповідних спеціальних символів:

«+» позначає атрибут з областю видимості типу загальнодоступний (public). Атрибут з цією областю видимості доступний з будь-якого іншого класу пакету, в якому визначена діаграма;

«#» атрибут із зоною видимості типу захищений (protected). Недоступний для всіх класів, за винятком підкласів даного класу;

«-» атрибут із зоною видимості типу закритий «private». Недоступний для всіх класів без винятків;

«~» видимість для елементів того ж простору імен (пакета) (package). Квантор видимості може бути опущений. У цьому випадку його відсутність просто означає, що видимість атрибута не вказується. Ця ситуація відрізняється від прийнятих за замовчуванням угод у традиційних мовах програмування, коли відсутність квантора видимості трактується як public або private. Однак замість умовних графічних позначень можна записувати відповідне ключове слово: public, protected, private.

Ім'я атрибута являє собою рядок тексту, який використовується в якості ідентифікатора відповідного атрибута й тому повинен бути унікальним в межах даного класу. Ім'я атрибута є єдиним обов'язковим елементом синтаксичного позначення атрибута.

Кратність атрибута характеризує загальну кількість конкретних атрибутів даного типу, що входять до складу окремого класу. У загальному випадку кратність записується у формі рядка тексту у квадратних дужках після імені відповідного атрибута:

[нижня_границя1 .. верхня_границя1, нижня_границя2.. верхня_границя2, ..., нижня_границяk .. верхня_границяk],

де нижня_границя й верхня_границя є позитивними цілими числами, кожна пара яких служить для позначення окремого замкненого інтервалу цілих чисел, у якого нижня (верхня) границя дорівнює значенню нижня_границя (верхня_границя).

У цілому дана умовна позначка кратності відповідає теоретико-множинному об'єднанню відповідних інтервалів. У якості верхньої_границі може використовуватися спеціальний символ "*", який означає довільне позитивне ціле число. Інакше кажучи, це означає необмежене зверху значення кратності відповідного атрибута.

Значення кратності з інтервалу випливають у монотонно зростаючому порядку без пропуску окремих чисел, що лежать між нижньої й верхньої границями. При цьому дотримуються наступного правила: відповідні нижні й

верхні границі інтервалів включаються в значення кратності. Якщо в якості кратності вказується однина, то кратність атрибута ухвалюється рівної даному числу. Якщо ж вказується єдиний знак "*", то це означає, що кратність атрибута може бути довільним позитивним цілим числом або нулем.

Якщо кратність атрибута не зазначена, то за замовчуванням ухвалюється її значення рівне 1..1, тобто в точності 1.

Тип атрибута являє собою вираження, семантика якого визначається мовою специфікації відповідної до моделі. У нотації UML тип атрибута іноді визначається залежно від мови програмування, яка передбачається використовувати для реалізації даної моделі. У найпростішому випадку тип атрибута вказується рядком тексту, що має осмислене значення в межах пакета або моделі, до яких ставиться розглянутий клас.

Початкове значення служить для завдання деякого початкового значення для відповідного атрибута в момент створення окремого екземпляра класу. Тут необхідно дотримуватися правила приналежності значення типу конкретного атрибута. Якщо початкове значення не зазначене, то значення відповідного атрибута не визначене на момент створення нового екземпляра класу. З іншого боку, конструктор відповідного об'єкта може перевизначати початкове значення в процесі виконання програми, якщо в цьому виникає необхідність.

Операції (методи) класу

Квантор видимості, як і у випадку атрибутів класу, може ухвалювати одне із трьох можливих значень і, відповідно, відображається за допомогою спеціального символу.

Символ "+" позначає операцію з областю видимості типу загальнодоступний (public). Символ "#" позначає операцію з областю видимості типу захищений (protected).

І, нарешті, символ "-" використовується для позначення операції з областю видимості типу закритий (private).

Квантор видимості для операції може бути опущений.

Ім'я операції являє собою рядок тексту, який використовується в якості ідентифікатора відповідної до операції й тому повинна бути унікальної в межах даного класу. Ім'я атрибута є єдиним обов'язковим елементом синтаксичного позначення операції.

Список параметрів є переліком розділених комами формальних параметрів, кожний з яких може бути представлений у наступному виді:

<вид параметра><ім'я параметра>:<вираження типу>=<значення параметра за замовчуванням>.

Тут *вид параметра* - є одне із ключових слів in, out або inout зі значенням in за замовчуванням, у випадку якщо вид параметра не вказується.

Ім'я параметра є ідентифікатор відповідного формального параметра.

Вираження типу є залежною від конкретної мови програмування специфікацією типу значення, що вертається, для відповідного формального параметра.

Значення за замовчуванням у загальному випадку являє собою вираження для значення формального параметра, синтаксис якого залежить від конкретної мови програмування й підкоряється прийнятим у ньому обмеженням.

Вираз типу значення, що повертається, також є залежною від мови реалізації специфікацією типу або типів значень параметрів, які повертаються об'єктом після виконання відповідної операції.

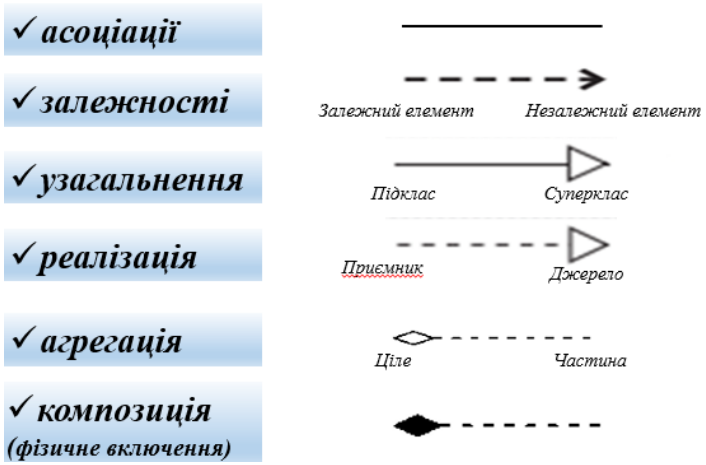
Рядок властивості служить для вказівки значень властивостей, які можуть бути застосовані до даного елемента. Рядок властивості не є обов'язковим, він може бути відсутнім, якщо ніякі властивості не специфіковані.

Відношення між класами

Крім внутрішньої будови або структури класів на відповідній діаграмі вказуються відношення між класами.

Кожне з цих відношень має власне графічне представлення на діаграмі, яке відображає взаємозв'язки між об'єктами відповідних класів.

Базовими відношеннями в мові UML є:



Відношення асоціації

Відношення асоціації відповідає наявності деякого відношення між класами. Дане відношення позначається суцільною лінією з додатковими спеціальними символами, які характеризують окремі властивості конкретної асоціації.

Ім'я асоціації є необов'язковим елементом її позначення. Якщо воно задане, то записується з великої букви поруч з лінією відповідної асоціації.

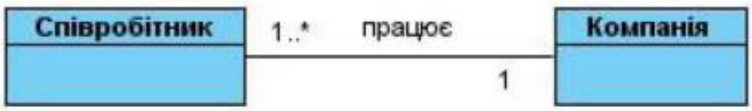
Для асоціації може позначатися кількість екземплярів об’єктів кожного класу, які беруть участь у зв’язку.

(0 - якщо жодного, 1 - якщо один, * - якщо багато)

Можуть вказуватися мінімальна й максимальна кількість, наприклад, 0 або 1...* означає, що на відповідному кінці асоціації може не бути жодного екземпляра, бути один або багато.

Бінарна асоціація – служить для зображення довільного відношення між двома класами. Вона зв’язує в точності два різних класа й може бути ненаправленим або направленим відношенням. Окремий випадок бінарної асоціації- рефлекційна асоціація, що зв'язує клас із самим собою. Ненаправлена асоціація зображується лінією без стрілки.

Як простий приклад ненаправленої бінарної асоціації можна розглянути відношення між двома класами – класом Компанія та класом Співробітник. Вони зв’язані між собою бінарною асоціацією «працює». Для даного відношення визначений наступний порядок читання - співробітник працює в компанії.



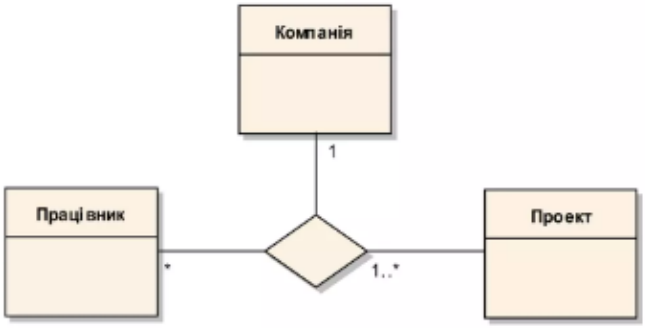
Тернарна асоціація й асоціації більш високої арності в загальному випадку називаються *N-арною асоціацією*. Така асоціація зв'язує деяким відношенням 3 і більш класів, при цьому один клас може брати участь в асоціації більш ніж один раз.

Клас асоціації має певну роль у відповідному відношенні, що може бути явно зазначене на діаграмі. Кожний екземпляр N-арної асоціації являє собою N-арний кортеж значень об'єктів з відповідних класів.

Бінарна асоціація є часним випадком N-арної асоціації, коли значення N=2, і має своє власне позначення.

N-арна асоціація графічно позначається ромбом, від якого ведуть лінії до символів класів даної асоціації. У цьому випадку ромб з'єднується із символами відповідних класів суцільними лініями. Звичайно лінії проводяться від вершин ромба або від середини його сторін. Ім'я N-арної асоціації записується поруч із ромбом відповідної асоціації.

Деякий клас може бути приєднаний до ромба пунктирною лінією. Це означає, що даний клас забезпечує підтримку властивостей відповідної N-Арної асоціації, а сама N-Арна асоціація має атрибути, операції й/або асоціації. Інакше кажучи, така асоціація, у свою чергу, є класом з відповідним позначенням у вигляді прямокутника і є самостійним елементом мови UML - асоціацією-класом (Association Class). N- Арная асоціація не може містити символ агрегації ні для якої зі своїх ролей.



Графічне зображення тернарної асоціації між трьома класами

Відношення узагальнення

Узагальнення – відношення між більш загальним поняттям (предком) і менш загальним поняттям (нащадком).

Менш загальний елемент моделі повинен бути позгодженний з більш загальним елементом і може містити додаткову інформацію. Дане відношення використовується для подання ієрархічних взаємозв'язків між різними елементами мови UML, такими як пакети, класи, варіанти використання.

Наслідування (Inheritance) - спеціальний концептуальний механізм, за допомогою якого більш спеціальні елементи включають в себе структуру і поведінку більш загальних елементів. Клас-нащадок має всі властивості і поведінку

класу-предка, а також має власні властивості і поведінку, які можуть бути відсутніми у класа-предка.

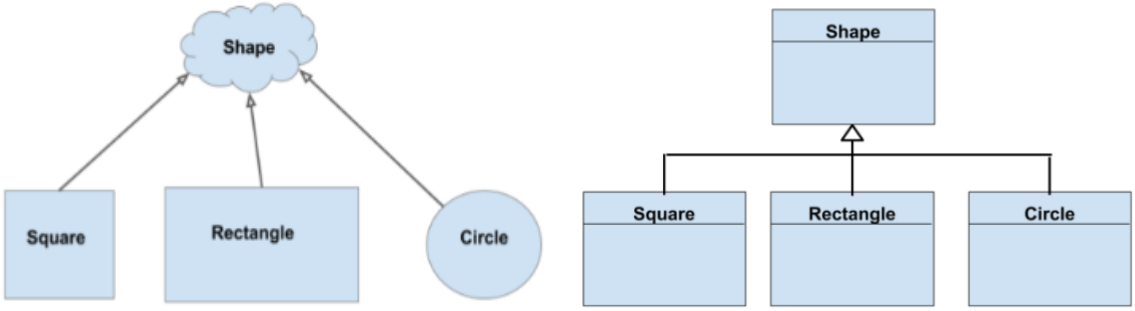
Батько, предок (Parent) - щодо узагальнення більш загальний елемент.

Нащадок (Child) - спеціалізація одного з елементів відношення узагальнення, званого в цьому випадку батьком.

На діаграмах відношення узагальнення позначається суцільною лінією з трикутною стрілкою на одному з кінців. Стрілка вказує на більш загальний клас (клас-предок), а її початок - на більш спеціальний клас (клас-нащадок).

На діаграмі може вказуватися кілька ліній для одного відношення узагальнення, що відбиває його характер. У цьому випадку більш загальний клас розбивається на підкласи одним відношенням узагальнення.

Наприклад, клас Shape(Фігура) може виступати в якості суперкласу для підкласів, що відповідають конкретним геометричним фігурам, таким як, Square(Квадрат), Rectangle(Прямокутник), Circle(Коло) та ін. Даний факт може бути представлений графічно у формі діаграми класів наступного виду:



Поруч із стрілкою узагальнення може розміщатися рядок тексту, що вказує на деякі додаткові властивості цього відношення. Даний текст буде ставитися до всіх ліній узагальнення, які йдуть до класів-нащадків. Інакше кажучи, відзначена властивість стосується всіх підкласів даного відношення. При цьому текст слід розглядати як обмеження, і тоді він записується у фігурних дужках.

У якості обмежень можуть бути використані наступні ключові слова мови UML:

1. ***{complete}*** - *означає, що в даному відношенні узагальнення специфіковані всі класи-нащадки, і інших класів-нащадків у даного класу-предка бути не може. Приклад - клас Клієнт_банку є предком для двох класів: Фізична_особа й Компанія, і інших класів-нащадків він не має. На відповідній діаграмі класів це можна вказати явно, записавши поруч із лінією узагальнення даний рядок- обмеження;*
2. ***{disjoint}*** - *означає, що класи-нащадки не можуть містити об'єктів, що одночасно є екземплярами двох або більше класів. У наведеному вище прикладі ця умова також виконується, оскільки передбачається, що ніяка конкретна фізична*

особа не може бути одночасно й конкретною компанією. У цьому випадку поруч із лінією узагальнення можна записати даний рядок-обмеження;

3. {incomplete} *- означає випадок, протилежний першому. А саме, передбачається, що на діаграмі зазначені не всі нащадки. Надалі можливо заповнити їхній перелік не змінюючи вже побудовану діаграму. Приклад - діаграма класу "Автомобіль", для якої вказівка всіх без винятку моделей автомобілів порівняння зі створенням відповідного каталогу. З іншого боку, для окремого завдання, такого як розробка системи продажу автомобілів конкретних моделей, у цьому немає необхідності. Але вказати неповноту структури класів-нащадків все-таки випливає;*

4. {overlapping} *- означає, що окремі екземпляри класів- нащадків можуть належати одночасно декільком класам. Приклад - клас "Багатокутник" є класом-предком для класу "Прямокутник" і класу "Ромб". Однак існує окремий клас "Квадрат", екземпляри якого одночасно є об'єктами перших двох класів. Цілком природно таку ситуацію вказати явно за допомогою даною рядка-обмеження.*

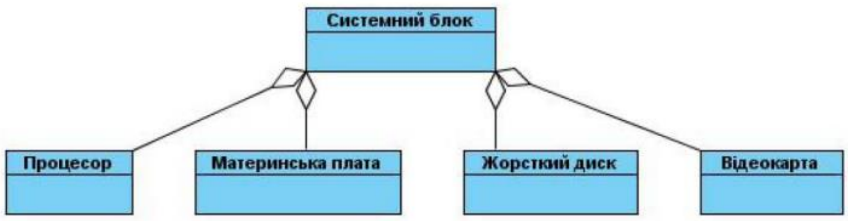
Відношення Агрегація

Агрегація (Aggregation) - спеціальна форма асоціації, яка служить для подання відношення типу "частина-ціле" між агрегатом (ціле) і його складовою частиною.

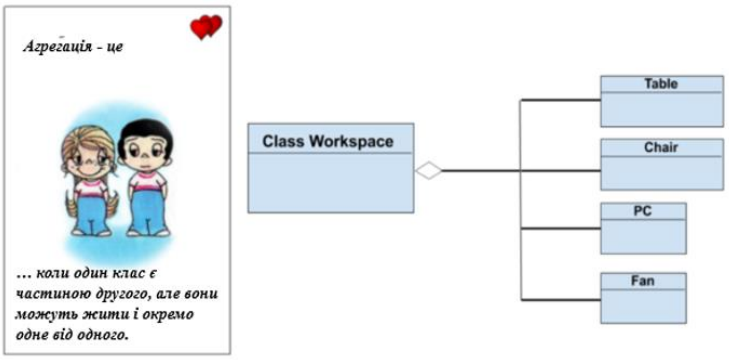
Ставлення агрегації має місце між декількома класами в тому випадку, якщо один з класів є сутність, яка включає в себе в якості складових частин інші сутності.

Графічно відношення агрегації зображується суцільною лінією, один з кінців якої являє собою не зафарбований усередині ромб. Цей ромб вказує на той клас, який являє собою "ціле" або клас-контейнер. Решта класи є його "частинами".

Як приклад відношення агрегації можна розглянути взаємозв'язок типу "частина-ціле", який має місце між класом Системний блок персонального комп'ютера і його складовими частинами: Процесор, Материнська плата, Оперативна пам'ять, Жорсткий диск і Гнучкий диск. Використовуючи позначення мови UML, компонентний склад системного блоку можна представити у вигляді відповідної діаграми класів, яка в даному випадку ілюструє ставлення агрегації.



Наприклад, робоче місце програміста складається зі стільця, стола, комп’ютера та вентилятора, але при знищенні класу «робоче місце», в нас просто залишуться всі ці класи, лише окремо один від одного.

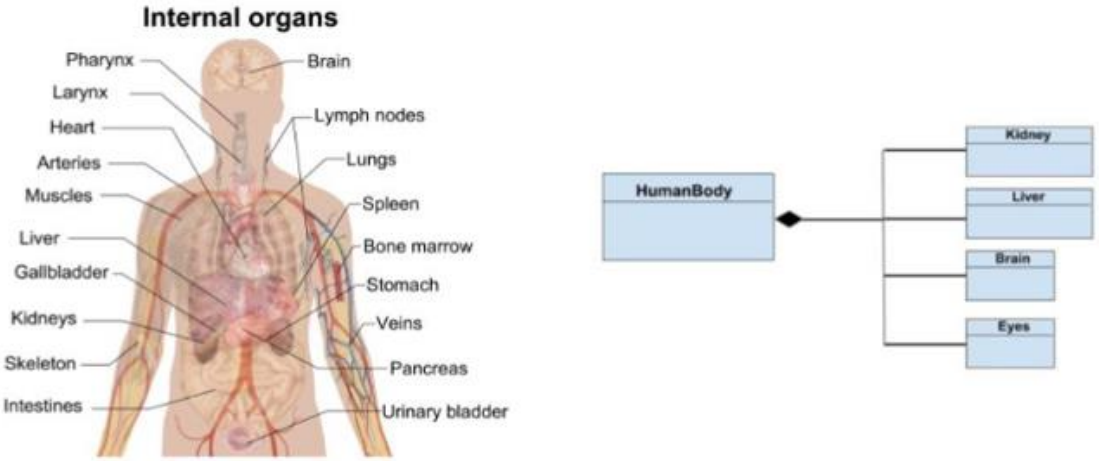


Відношення композиції

Композиція (Composition) - різновид відношення агрегації, при якій складові частини цілого мають такий же час життя, що й саме ціле. Ці частини знищуються разом зі знищенням цілого.

Графічно відношення композиції зображується суцільною лінією, один з кінців якої являє собою зафарбований усередині ромб. Цей ромб вказує на той клас, який представляє собою клас-композит. Решта класи є його "частинами"

Наприклад, наше тіло складається з органів, але самі по собі вони не дієздатні.

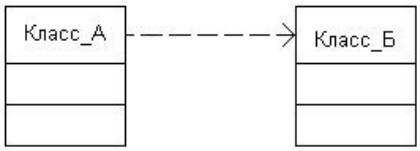


Зауваження: Для виявлення типу зв'язку – *агрегація* чи *композиція* варто проаналізувати, що буде з класом-частиною після знищення класу контейнеру. Чи матиме він окремий сенс в рамках поставленої задачі? Так, наприклад, після знищення Будівлі, Приміщення припиняє своє існування, а от Обладнання матиме сенс.

Відношення залежності

Відношення залежності графічно зображуються пунктирною лінією між відповідними елементами зі стрілкою, направленою від класу-клієнту залежності до незалежного класу або класу-джерела.

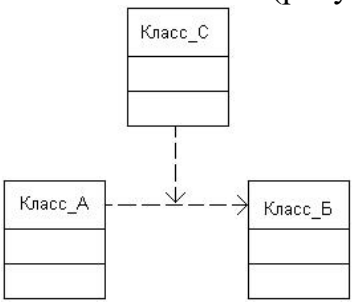
На рисунку зображено два класи: Клас_А і Клас_Б, при цьому Клас_Б є джерелом деякої залежності, а Клас_А - клієнтом цієї залежності.



Графічне зображення відношення залежності на діаграмі

Наприклад, у класа Person є метод hasRead з вхідним параметром book, який повертає true, якщо, наприклад, людина прочитала книгу.

У якості класу-клієнта й класу-джерела залежності можуть виступати цілі безлічі елементів моделі. У цьому випадку одна лінія зі стрілкою, що виходить від джерела залежності, розщеплюється в деякій крапці на кілька окремих ліній, кожна з яких має окрему стрілку для класу-клієнта. Наприклад, якщо функціонування Класу_С залежить від особливостей реалізації Класу_А и Класу_Б, то дана залежність може бути зображена в такий спосіб (рисунок).



Графічне представлення залежності між класом-клієнтом (Клас_С) і класами-джерелами (Клас_А и Клас_Б)

Стрілка може позначатися необов'язковим, але стандартним ключовим словом у лапках і необов'язковим індивідуальним іменем.

Для відношення залежності зумовлені ключові слова (стереотипи), які позначають деякі спеціальні його види:

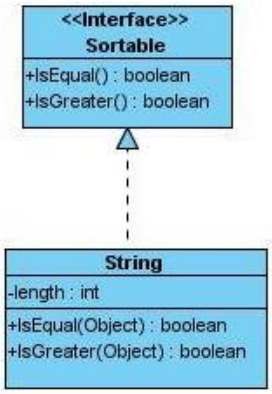
- **«access»** - для позначення доступності відкритих атрибутів і операцій класу-джерела для класів-клієнтів;
- **«bind»** - клас-клієнт може використовувати деякий шаблон для своєї подальшої параметризації;
- **«derive»** - атрибути класу-клієнту можуть бути обчислені по атрибутах класу-джерела;
- **«import»** - відкриті атрибути і операції класу-джерела стають частиною класу-клієнту, так, ніби вони були оголошені безпосередньо в ньому;
- **«refine»** - вказує, що клас-клієнт служить уточненням клас-джерела в силу причин історичного характеру, коли з'являється додаткова інформація в ході роботи над проектом;
- **«use»** - семантика початкового елементу залежить від семантики відкритого змісту цільового елемента;
- та інші.

Відношення реалізації

Реалізація (Realization) - це семантичне відношення між класифікаторами, при якому один класифікатор визначає сутність, а інший гарантує її виконання.

Відношення реалізації в діаграмах класів зустрічаються між інтерфейсами і класами, що реалізовують їх.

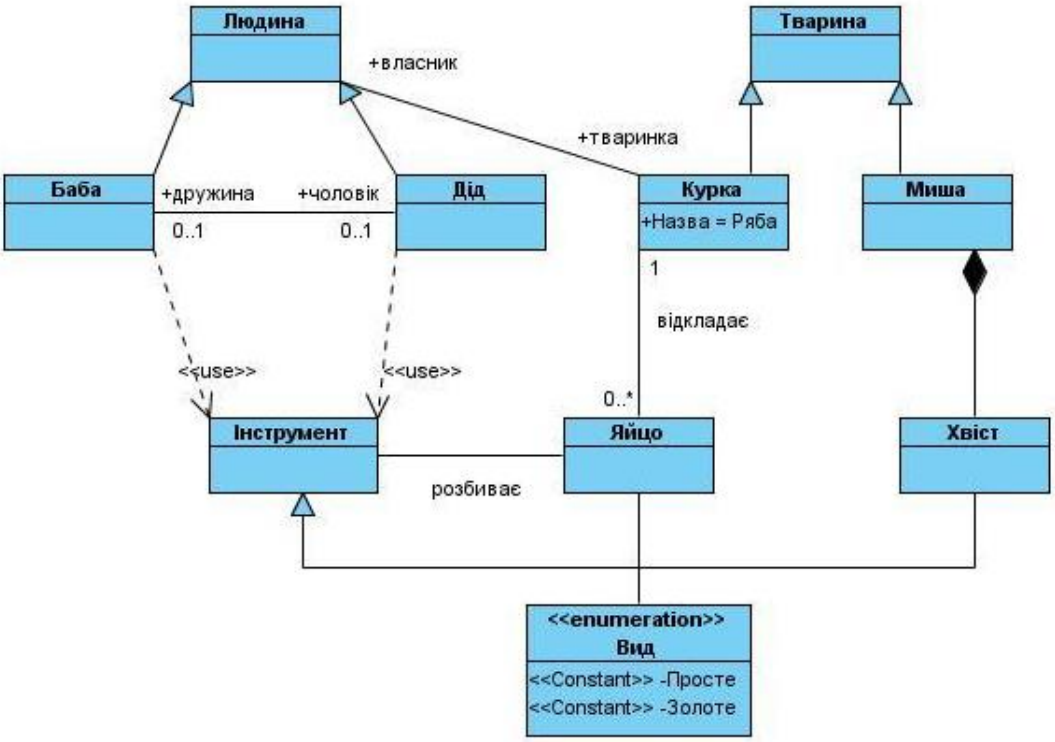
Зображається відношення реалізації у вигляді пунктирної лінії з незафарбованою стрілкою, як щось середнє між відношеннями узагальнення та залежності.



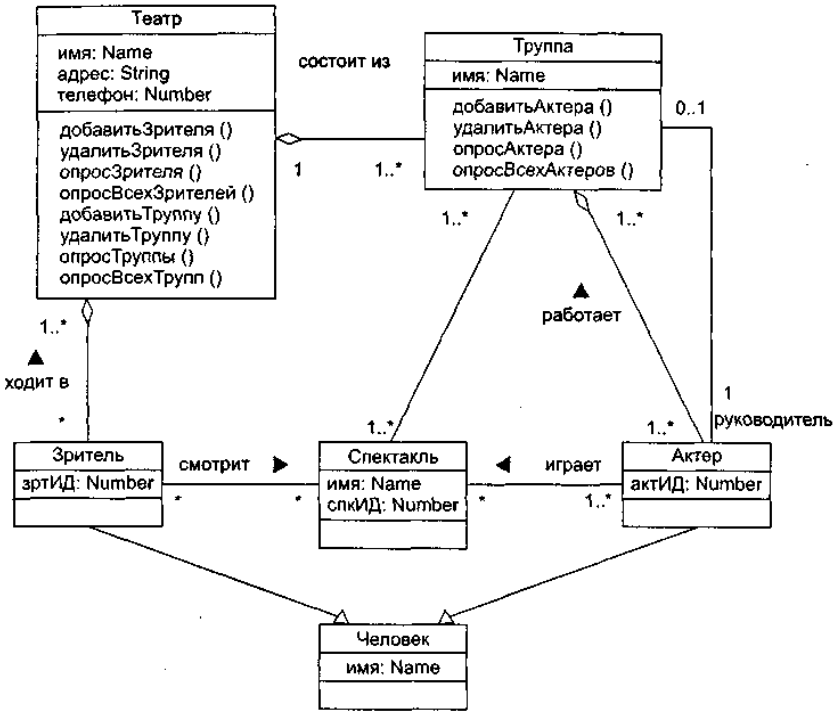
Приклад:

Для наочності вищевикладеного матеріалу, розглянемо діаграму класів, яка ілюструє відому казку «Курка Ряба».

У казці Дід (клас Дід) та Баба (клас Баба) є Людьми (класи Дід та Баба наслідують від класу Людина), та вони є власниками (відношення асоціації) Курки (клас Курка) на ім’я Ряба. В свою чергу, Курка Ряба є твариною (тобто клас Курка наслідує від класу Тварина). Курка може відкладати яйця (клас Яйце, відношення асоціації «відкладати»). Зауважимо, що яйце в казці фігурувало як золоте, так і просте, що відображується на діаграмі за допомогою класу <<enumeration>> Вид. Ще однією дійовою особою казки є Миша (клас Миша), яка також є Твариною та володіє Хвостом (клас Хвіст) як інструментом (клас Інструмент) для розбиття яйця (відношення асоціації «розбиває)».



Приклад діаграми класів інформаційної системи театру



II Завдання до практичної роботи №3

По своїй предметній області побудувати діаграму класів використовуючи різні види відношень з вказівкою ролей та кратності.

III Зробити висновки по виконанню практичної роботи 3

IV Контрольні питання для підготовки к практичній роботі №3

1. Що таке клас?
2. Що таке атрибут класу?

3. Що таке квантор видимості?
4. Що таке тип атрибуту?
5. Що таке операція класу?
6. Як записуються операції?
7. Що таке асоціація?
8. Для чого використовується кратність?
9. Що таке N-арна асоціація?
10. Що таке спадкування?
11. Що таке множинне спадкування?
12. Для чого використовують рядки-обмеження?
13. Які ключові слова можна використовувати в якості рядків-обмежень? Що вони означають?
14. Що таке агрегація? Як графічно можна представити агрегацію?
15. В чому різниця між агрегацією та композицією?
16. Що таке залежність?
17. Що таке стереотип залежності? Які стереотипи залежностей Вам відомі?

Практична робота №4

Тема: Побудова діаграми схем станів (Statechart Diagram).

Мета: Отримати практичні навики побудови діаграми станів.

Хід роботи:

І. Теоретичні відомості

Для моделювання поведінки системи використовують:

- автомати
- взаємодії.

Діаграми схем станів

Діаграма схем станів - одна з п'яти діаграм UML, що моделюють динаміку систем. Діаграма схем станів відображає кінцевий автомат, виділяючи потік керування, що випливає від стану до стану. *Кінцевий автомат* - поведінка, яка визначає послідовність станів у ході існування об'єкта. Ця послідовність розглядається як відповідь на події й включає реакції на ці події.

Діаграма схем станів показує:

1) набір станів системи;

2) події, які викликають перехід з одного стану в інше;

3) дії, які відбуваються в результаті зміни стану.

Головне призначення цієї діаграми - описати можливі послідовності станів і переходів, які в сукупності характеризують поведінка елемента моделі протягом його життєвого циклу. Діаграма станів представляє динамічну поведінку сутностей, на основі специфікації їх реакції на сприйняття деяких конкретних подій. Системи, які реагують на зовнішні дії від інших систем або від користувачів, іноді називають реактивними. Якщо такі дії ініціюються в довільні випадкові моменти часу, то говорять про асинхронну поведінку моделі.

Хоча діаграми станів найчастіше використовуються для опису поведінки окремих екземплярів класів (об'єктів), але вони також можуть бути застосовані для

специфікації функціональності інших компонентів моделей, таких як варіанти використання, актори, підсистеми, операції й методи.

Діаграма станів по суті є графом спеціального виду, який представляє деякий автомат. Поняття автомата в контексті UML має досить специфічну семантику, засновану на теорії автоматів. Вершинами цього графа є стани й деякі інші типи елементів автомата (псевдо-стани), які зображуються відповідними графічними символами. Дуги графа служать для позначення переходів зі стану в стан. Діаграми станів можуть бути вкладені друг у друга, створюючи вкладені діаграми більш детального представлення окремих елементів моделі.

Стан

У мові UML під *станом* розуміється абстрактний мета клас, використовуваний для моделювання окремої ситуації, протягом якої має місце виконання деякої умови. Стан може бути заданий у вигляді набору конкретних значень атрибутів класу або об'єкта, при цьому зміна їх окремих значень буде відбивати зміну стану моделюємого класу або об'єкта.

Не кожний атрибут класу може характеризувати його стан. Як правило, мають значення тільки такі властивості елементів системи, які відбивають динамічний або функціональний аспект її поведінки. У цьому випадку стан буде характеризуватися деякою інваріантною умовою, що включають у себе тільки значимі для поведінки класу атрибути і їх значення.

Елемент переходить у стан в момент початку відповідної діяльності й залишає даний стан у момент її завершення.

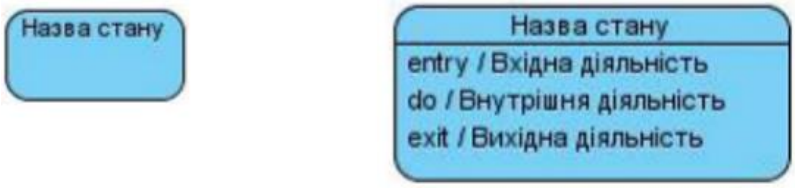


Рисунок 1 - Графічне зображення станів на діаграмі станів

Стан на діаграмі зображується прямокутником з округленими вершинами (рис. 1). Цей прямокутник, у свою чергу, може бути розділений на дві секції горизонтальною лінією. Якщо зазначена лише одна секція, то в ній записується тільки ім'я стану (рис. 1 а). А якщо ні, то в першій з них записується ім'я стану, а в другий - список деяких *внутрішніх дій* або *переходів* у даному стані (рис. 1 б). При цьому під *дією* в мові UML розуміють деяку атомарну операцію, виконання якої приводить до зміни стану або поверненню деякого значення (наприклад, "істина" або "неправда").

Список внутрішніх дій

Ця секція містить перелік внутрішніх дій або *діяльностей*, які виконуються в процесі знаходження елемента в даному стані. Кожне з дій записується у вигляді окремого рядка й має наступний формат:

< мітка-дії '/' вираження - дії>

Мітка дії вказує на обставини або умови, при яких буде виконуватися діяльність, певна вираженням дії. При цьому *вираження дії* може використовувати будь-які атрибути й зв'язки, які належать області імен або контексту об'єкта.

Перелік міток дії має фіксовані значення в мові UML, які не можуть бути використані в якості імен подій. Ці значення наступні:

- entry - ця мітка вказує на дію, специфікована наступним за нею вираженням дії, яка виконується в момент входу в даний стан (вхідна дія);
- exit - ця мітка вказує на дію, специфікована наступним за нею вираженням дії, яка виконується в момент виходу з даного стану (вихідна дія);
- do - ця мітка специфікує діяльність, що виконується ("do activity"), яка виконується протягом усього часу, поки об'єкт перебуває в даному стані, або доти, поки не закінчиться обчислення, специфіковане наступним за нею вираженням дії.

В останньому випадку при завершенні події генерується відповідний результат.

- include - ця мітка використовується для звертання до під-автомату, при цьому наступне за нею вираження дії містить ім'я цього під-автомату.

Як приклад стану розглянемо аутентифікацію клієнта для доступу до банкомату. Список внутрішніх дій у даному стані може включати наступні дії: перша **дія вхідна** – вона виконується при вході в цей стан і пов'язана з одержанням рядка символів, що відповідають PIN-коду клієнта. Далі виконується діяльність по перевірці введеного клієнтом PIN-коду. При успішному завершенні цієї перевірки виконується **дія на виході**, що відображає меню доступних для клієнта опцій.

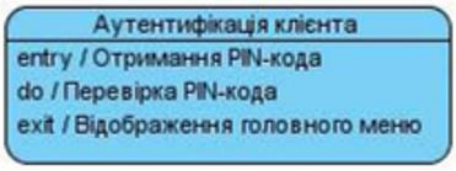


Рисунок 2 - Приклад стану з непустою секцією внутрішніх дій

Початковий стан

Початковий стан являє собою окремий випадок стану, який не містить ніяких внутрішніх дій (псевдостан). У цьому стані перебуває об'єкт за замовчуванням у початковий момент часу. Воно служить для вказівки на діаграмі станів графічної області, від якої починається процес зміни станів.

Рисунок 3 – Початковий та кінцевий стани

Кінцевий стан

Кінцевий (фінальний) стан являє собою окремий випадок стану, який також не містить ніяких внутрішніх дій (псевдостан). У цьому стані буде перебувати об'єкт за замовчуванням після завершення роботи автомата в кінцевий момент часу. Воно служить для вказівки на діаграмі станів графічної області, у якій завершується процес зміни станів або життєвий цикл даного об'єкта.

Перехід

Простий перехід (simple transition) являє собою відношення між двома послідовними станами, яке вказує на факт зміни одного стану іншим. Перебування об'єкта в першому стані може супроводжуватися виконанням деяких дій, а перехід у другий стан буде можливий після завершення цих дій, а також після задоволення деяких додаткових умов. У цьому випадку говорять, що перехід спрацьовує, Або відбувається спрацьовування переходу. До спрацьовування переходу об'єкт

перебуває в попередньому від нього стані, називаним вихідним станом, або в джерелі, а після його спрацьовування об'єкт перебуває наступному від нього стані (цільовому стані).

Перехід здійснюється при настанні деякої події: закінчення виконання діяльності (do activity), одержанні об'єктом повідомлення або прийманням сигналу. Спрацьовування переходу може залежати не тільки від настання деякої події, але й від виконання певного умови, називаної *сторожовою умовою*. Об'єкт перейде з одного стану в інше в тому випадку, якщо відбулася зазначена подія й сторожова умова прийняла значення "істина".

На діаграмі станів перехід зображується суцільною лінією зі стрілкою, яка спрямована в цільовий стан. Кожний перехід може бути позначений рядком тексту, який має наступний загальний формат:

<сигнатура події>'['<сторожова умова>']' <вираження дії>.

При цьому сигнатура події описує деяка подія з необхідними аргументами:

<ім'я події>'('<список параметрів, розділених комами>')'.

Розглянемо приклад побудови схем станів на прикладі кофейного апарату:

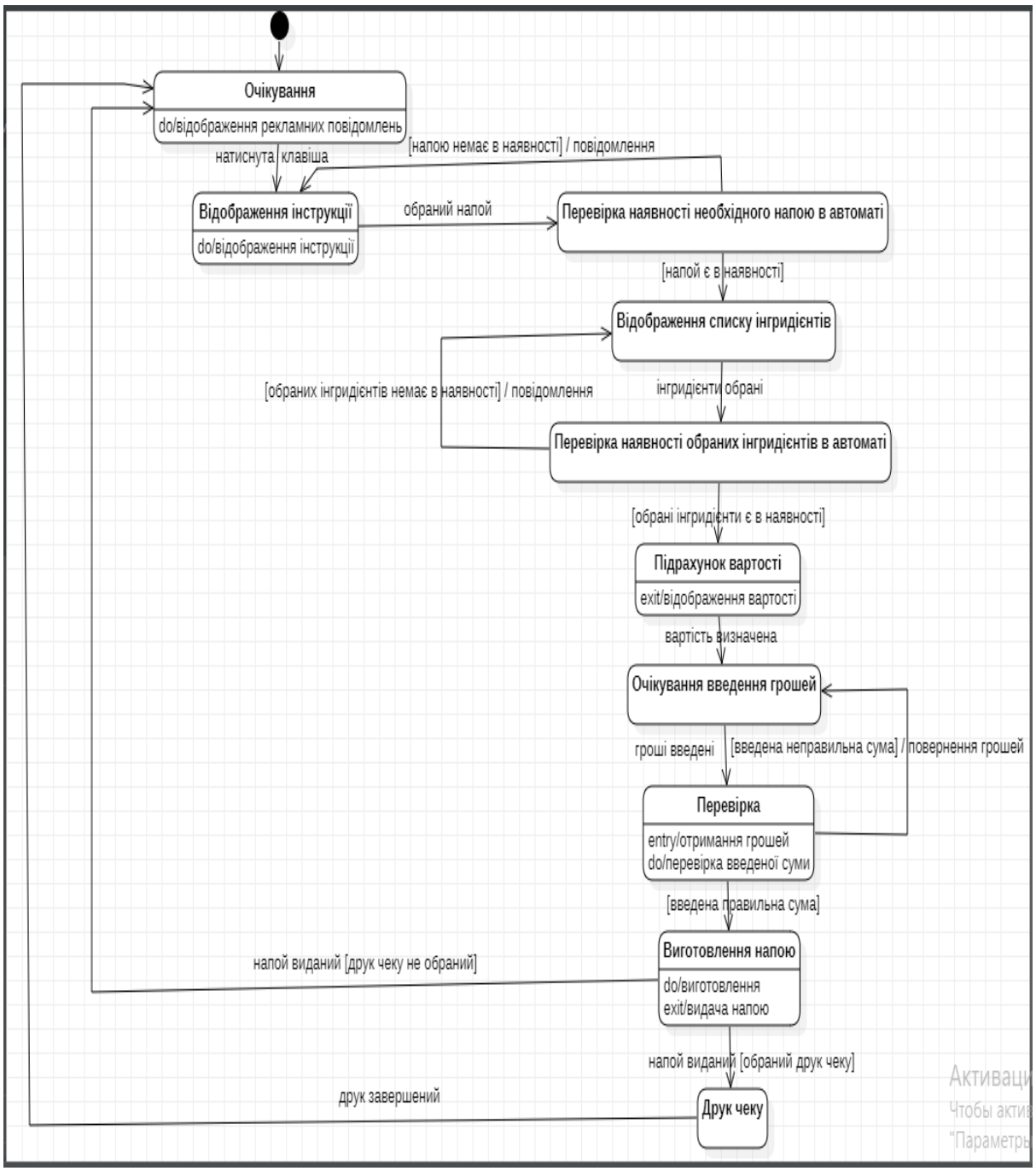
Автомат автоматично видає склянку, цукор і ложечку. За допомогою панелі керування можна встановити режим готування напою з декофеїнової кави (якщо в один з контейнерів засипана розчинна кава). Є функції скасування подачі стаканчика, регулювання цукру на напій.

Усі налаштування апарата можна задавати з зовнішньої панелі керування (встановлюється витрата продуктів на порцію, температура обробки, режим самоочистки, знімаються показники лічильників приготовлених порцій, виторгу і т. д).

Автомат обладнаний платіжними системами:

- монетоприємником з видачею решти грошей
- банкнотоприємником зі стекером

Робота з автоматом відбувається в такий спосіб. Автомат за замовчуванням перебуває в режимі очікування й відображає на екрані рекламну інформацію. Людина, яка бажає одержати напій, натискає будь-яку клавішу. Автомат переходить в активний режим і відображає на екрані інструкцію з його використання. Людина вибирає бажаний напій, додаткові інгридієнти до нього. При відсутності чого-небудь автомат видає про це повідомлення й пропонує зробити інший вибір. Далі автомат на екрані відображає вартість обраного напою й переходить у режим очікування введення грошей. Людина вводить в автомат необхідну суму. Якщо введена сума вірна, автомат готовить і видає напій. Інакше - повертає гроші. При бажанні людини може роздрукувати чек. Потім автомат знову переходить у режим очікування.


Очікування
do/відображення рекламних повідомлень
натиснута клавіша
[напою немає в наявності] / повідомлення
Відображення інструкції
do/відображення інструкції
обраний напой
Перевірка наявності необхідного напою в автоматі
[напой є в наявності]
Відображення списку інгридієнтів
[обраних інгридієнтів немає в наявності] / повідомлення
інгридієнти обрані
Перевірка наявності обраних інгридієнтів в автоматі
[обрані інгридієнти є в наявності]
Підрахунок вартості
exit/відображення вартості
вартість визначена
Очікування введення грошей
гроші введені
[введена неправильна сума] / повернення грошей
Перевірка
entry/отримання грошей
do/перевірка введеної суми
[введена правильна сума]
Виготовлення напою
do/виготовлення
exit/видача напою
напой виданий [друк чеку не обраний]
напой виданий [обраний друк чеку]
друк завершений
Друк чеку

Опис станів:

Стан	Опис стану
Очікування	Автомат по виготовленню кави знаходиться в режимі очікування вхідних впливів. В цей час автомат відображає рекламні повідомлення.
Відображення інструкції	Після натискання будь-якої із клавіш автомат знаходиться в режимі відображення інструкції.
Перевірка наявності необхідного напою в автоматі	Автомат знаходиться в режимі перевірки наявності обраного напою. У випадку його відсутності автомат видає відповідне повідомлення та переходить у стан відображення інструкції.
Відображення списку інгридієнтів	Автомат відображає список інгридієнтів, які можна додати до обраного напою.
Перевірка наявності обраних інгридієнтів в автоматі	Автомат перевіряє наявність обраних інгридієнтів. В випадку їх відсутності автомат видає відповідне повідомлення та переходить в стан відображення списку інгридієнтів.
Підрахунок вартості	Автомат визначає вартість напою та відображає її на екрані
Очікування введення грошей	Автомат очікує вводу грошей в відповідний приймомний пристрій
Перевірка	Автомат порівнює отриману суму з потрібною. Якщо сума неправильна, автомат повертає гроші та переходить в стан очікування введення грошей
Виготовлення напою	Якщо введена правильна сума, автомат починає готувати обраний напой та потім видає його людині
Друк чеку	Автомат друкує чек, якщо людина обрала друк чеку

II Завдання до практичної роботи №4

1. По своїй предметній області розробити діаграму схем станів.
2. Кожний стан описати у вигляди таблиці як показано в прикладі.

III Зробити висновки по виконанню практичної роботи №4

IV Контрольні питання для підготовки к практичній роботі №4

1. Що відображає діаграма схем станів?
2. Що таке стан?
3. Дайте визначення поняттю список внутрішніх подій.
4. Які види псевдо-станів Вам відомі?
5. Що таке подія?
6. Для чого використовується сторожова умова?

Практична робота №5

Тема: Побудова діаграми діяльності (Activity Diagram).

Мета: Отримати практичні навики побудови діаграми діяльності.

Хід роботи:

І. Теоретичні відомості

При моделюванні поведінки проектованої або аналізованої системи виникає необхідність не тільки представити процес зміни її станів, але й деталізувати особливості алгоритмічної й логічної реалізації виконуваних системою операцій. Традиційно для цієї мети використовувалися блок-схеми або структурні схеми алгоритмів. Кожна така схема акцентує увагу на послідовності виконання певних дій або елементарних операцій, які в сукупності приводять до одержання бажаного результату.

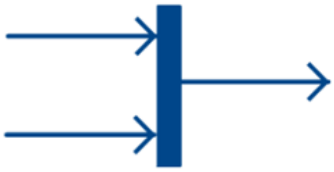
Важливо підкреслити та обставина, що зі збільшенням складності системи строге дотримання послідовності виконуваних операцій здобуває усе більш важливе значення. Якщо спробувати заварити кава холодною водою, то ми можемо тільки зіпсувати одну порцію напою. Порушення послідовності операцій при ремонті двигуна може привести до його поломки або виходу з ладу.

Для моделювання процесу виконання операцій у мові UML використовуються так звані *діаграми діяльності*. Застосовувана в них графічна нотація багато в чому схожа на нотацію діаграми станів, оскільки на діаграмах діяльності також присутні позначення станів і переходів. Відмінність полягає в семантиці станів, які використовуються для вистави не діяльностей, а дій, і у відсутності на переходах сигнатури подій. Кожний стан на діаграмі діяльності відповідає виконанню деякої елементарної операції, а перехід у наступний стан спрацьовує тільки при завершенні цієї, операції в попередньому стані. Графічно діаграма діяльності представляється у формі графа діяльності, вершинами якого є стани дії, а дугами - переходи від одного стану дії до іншого.

Таким чином, діаграми діяльності можна вважати часткам случаємо діаграм станів. Саме вони дозволяють реалізувати в мові UML особливості процедурного й синхронного керування, обумовленого завершенням внутрішніх діяльностей і дій. Метамодель UML надає для цього необхідні терміни й семантику. Основним напрямком використання діаграм діяльності є візуалізація особливостей реалізації операцій класів, коли необхідно представити алгоритми їх виконання. При цьому кожний стан може бути виконанням операції деякого класу або її частини, дозволяючи використовувати діаграми діяльності для опису реакцій на внутрішні події системи.

У контексті мови UML *діяльність (activity)* являє собою деяку сукупність окремих обчислень, виконуваних автоматом. При цьому окремі елементарні обчислення можуть приводити до деякого результату або дії (action). На діаграмі діяльності відображається логіка або послідовність переходу від однієї діяльності до іншої, при цьому увага фіксується на результаті діяльності. Сам же результат може привести до зміни стану системи або поверненню деякого значення.

Позначення діаграми діяльності

Символ	Ім'я	Використання
●	Початковий вузол	Відправна точка, або початковий стан
	Дія	Представлення діяльності, завдання для виконання
	Потік керування	Спрямований потік, контрольний потік діяльності
	Кінцевий вузол активності	Кінцевий стан, завершення усіх потоків процесу
	Кінцевий вузол потоку	Кінець одного потоку
	Вузол прийняття рішення	Розгалуження з умовою та кількома варіантами дій. Має один вхід декілька виходів
	Вузол злиття	Об'єднання потоків створених вузлом прийняття рішень. Має кілька входів і один вихід
	Вилка	Розподілення потоку на кілька паралельних без прийняття рішення
	Злиття	Об'єднання декількох паралельних потоків
	Надсилання сигналу	Вказує на те що сигнал надсилається на приймальну діяльність
	Отримання сигналу	Вказує на отримання сигналу
	Коментар	Дозволяє робити коментарі до діаграми. З'єднується пунктирною лінією

Стан дії

Стан дії (action state) є спеціальним випадком стану з деякою вхідною дією й принаймні одним вихідним зі стану переходом. Цей перехід неявно припускає, що вхідна дія вже завершилася. Стан дії не може мати внутрішніх переходів, оскільки воно є елементарним. Звичайне використання стану дії полягає в моделюванні одного кроку виконання алгоритму (процедури) або потоку керування.

Графічно стан дії зображується фігурою, що нагадує прямокутник, бічні сторони якого замінені опуклими дугами (рис. 1). Усередині цієї фігури записується вираження дії (action-expression), яка повинна є унікальним у межах однієї діаграми діяльності.

Рисунок 1 – Графічне зображення стану дії

Дія може бути записане природньою мовою, деякому псевдокоді або мові програмування. Ніяких додаткових або неявних обмежень при записі дій не накладається.

Іноді виникає необхідність представити на діаграмі діяльності деяка складна дія, яка, у свою чергу, складається з декількох більш простих дій. У цьому випадку можна використовувати спеціальне позначення так званого стану під-діяльності (subactivity state). Такий стан є графом діяльності й позначається спеціальною піктограмою в правому нижньому куті символу стану дії (рис. 2). Ця конструкція може застосовуватися до будь-якого елемента мови UML, який підтримує "вкладеність" своєї структури. При цьому піктограма може бути додатково позначена типом вкладеної структури.

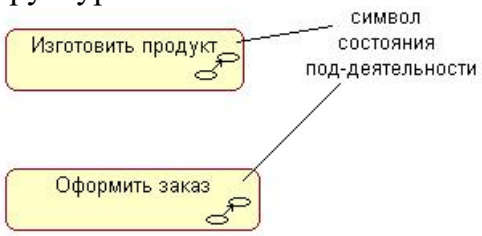


Рисунок 2 - Графічне зображення стану під-діяльності

Кожна діаграма діяльності повинна мати єдиний початковий і єдиний кінцевий стану. Вони мають такі ж позначення, як і на діаграмі станів. При цьому кожна діяльність починається в початковому стані й закінчується в кінцевому стані. Саму діаграму діяльності прийнято мати у своєму розпорядженні такий образ, щоб дії випливали зверху вниз. У цьому випадку початковий стан буде зображуватися у верхній частині діаграми, а кінцеве - у її нижній частині.

Переходи

При побудові діаграми діяльності використовуються тільки *не тригерні переходи,* тобто такі, які спрацьовують відразу після завершення діяльності або виконання відповідного дії. Цей перехід переводить діяльність у наступний стан відразу, як тільки закінчиться дія в попередньому стані. На діаграмі такий перехід зображується суцільною лінією зі стрілкою.

Якщо зі стану дії виходить єдиний перехід, то він може бути ніяк не позначений. Якщо ж таких переходів кілька, то спрацювати може тільки один з

них. Саме в цьому випадку для кожного з таких переходів повинне бути явно записана *сторожова умова* в прямих дужках. При цьому для всіх вихідних з деякого стану переходів повинне виконуватися вимога істинності тільки одного з них. Подібний випадок зустрічається тоді, коли послідовно виконувана діяльність повинна розділитися на альтернативні галузі залежно від значення деякого проміжного результату. Така ситуація одержала назву *розгалуження*, а для її позначення застосовується спеціальний символ.

Графічно розгалуження на діаграмі діяльності позначається невеликим ромбом, усередині якого немає ніякого тексту (рис. 3). У цей ромб може входити тільки одна стрілка від того стану дії, після виконання якого потік керування повинен бути продовжений по одній з галузей, що взаємно виключають. Прийняте вхідну стрілку приєднувати до верхньої або лівій вершині символу розгалуження. Вихідних стрілок може бути дві або більше, але для кожної з них явно вказується відповідна сторожова умова у формі булевского вираження.

Як приклад розглянемо фрагмент відомого алгоритму знаходження корнів квадратного рівняння. У загальному випадку після приведення рівняння другому ступеня до канонічного виду: a*x*x+в*x+c=0 необхідно обчислити його дискримінант. Причому, у випадку негативного дискримінанта рівняння не має розв'язку на безлічі дійсних чисел, і подальші обчислення повинні бути припинені. При ненегативному дискримінанті рівняння має розв'язок, коріння якого можуть бути отримані на основі конкретної розрахункової формули.

Графічно фрагмент процедури обчислення корінь квадратного рівняння може бути представлений у вигляді діаграми діяльності із трьома станами дії й розгалуженням (рис. 3). Кожний з переходів, що виходять зі стану "Обчислити дискримінант", має сторожову умову, що визначає єдину галузі, по якій може бути продовжений процес обчислення корінь залежно від знаку дискримінанта. Очевидно, що у випадку його заперечності, ми відразу попадаємо в кінцевий стан, тим самим завершуючи виконання алгоритму в цілому.

Рисунок 3 - Фрагмент діаграми діяльності для алгоритму знаходження кореня квадратного рівняння

У розглянутому прикладі, як видне з рис. 3, виконувані дії з'єднуються в кінцевому стані. Однак це зовсім не є обов'язковим. Можна зобразити ще один символ розгалуження, який буде мати кілька вхідних переходів і один вихідний.

У наступному прикладі (рис. 4) розраховується загальна вартість товарів, що купуються по кредитній картці в супермаркеті. Якщо ця вартість перевищує $50, то виконується аутентифікація особистості власника картки. У випадку позитивної перевірки (картка дійсна) або якщо вартість товарів не перевищує $50, відбувається зняття суми з рахунку й оплата вартості товарів. При негативному результаті (картка недійсна) оплати не відбувається, і товар залишається в продавця.

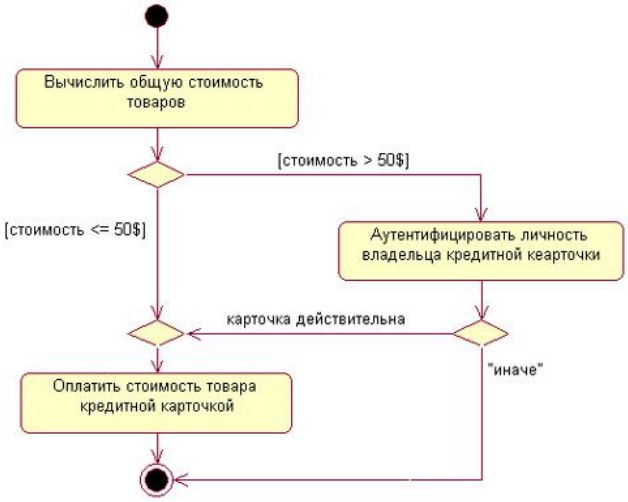


Рисунок 4 - Різні варіанти розгалужень на діаграмі діяльності

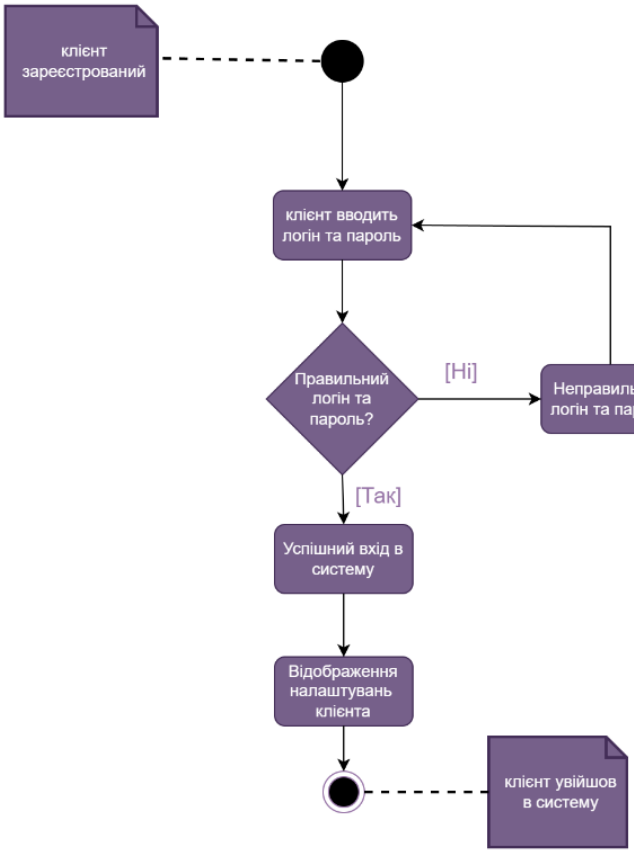


Рисунок 5 - Діаграма входу в систему

Один з найбільш значимих недоліків звичайних блок-схем або структурних схем алгоритмів пов'язаний із проблемою зображення паралельних галузей окремих обчислень. Оскільки розпаралелювання обчислень суттєво підвищує

загальна швидкодія програмних систем, необхідні графічні примітиви для вистави паралельних процесів. У мові UML для цієї мети використовується спеціальний символ для поділу й злиття паралельних обчислень або потоків керування. Таким символом є пряма риска, аналогічно позначенню переходу у формалізмі мереж Петри.

Як правило, така риска зображується відрізком горизонтальної лінії, товщина якої трохи ширше основних суцільних ліній діаграми діяльності. При цьому поділ (concurrent fork) має один вхідний перехід і кілька вихідних (рис. 6, а). Злиття (concurrent join), навпаки, має кілька вхідних переходів і один вихідний (рис. 6, б).

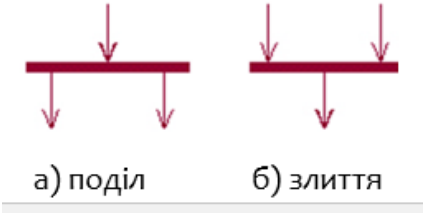


Рисунок 6 - Графічне зображення поділу й злиття паралельних потоків керування

Доріжки

Діаграми діяльності можуть бути використані не тільки для специфікації алгоритмів обчислень або потоків керування в програмних системах. Не менш важлива область їх застосування пов'язана з моделюванням бізнес-процесів. Дійсно, діяльність будь-якої компанії (фірми) також являє собою не що інше, як сукупність окремих дій, спрямованих на досягнення необхідного результату. Однак стосовно до бізнес-процесам бажане виконання кожної дії асоціювати з конкретним підрозділом компанії. У цьому випадку підрозділ відповідає за реалізацію окремих дій, а сам бізнес-процес представляється у вигляді переходів дій з одного підрозділу до іншого.

Для моделювання цих особливостей у мові UML використовується спеціальна конструкція назва, що одержала, доріжки (swimlanes). Мається на увазі візуальна аналогія із плавальними доріжками в басейні, якщо дивитися на відповідну діаграму. При цьому всі стани дії на діаграмі діяльності діляться на окремі групи, які відділяються друг від друга вертикальними лініями. Дві сусідні лінії й утворюють доріжку, а група станів між цими лініями виконується окремим підрозділом (відділом, групою, відділенням, філією) компанії (рис. 7).

Назви підрозділів явно вказуються у верхній частині доріжки. Перетинати лінію доріжки можуть тільки переходи, які в цьому випадку позначають вихід або вхід потоку керування у відповідний підрозділ компанії. Порядок проходження доріжок не несе якої-небудь семантичної інформації й визначається міркуваннями зручності.

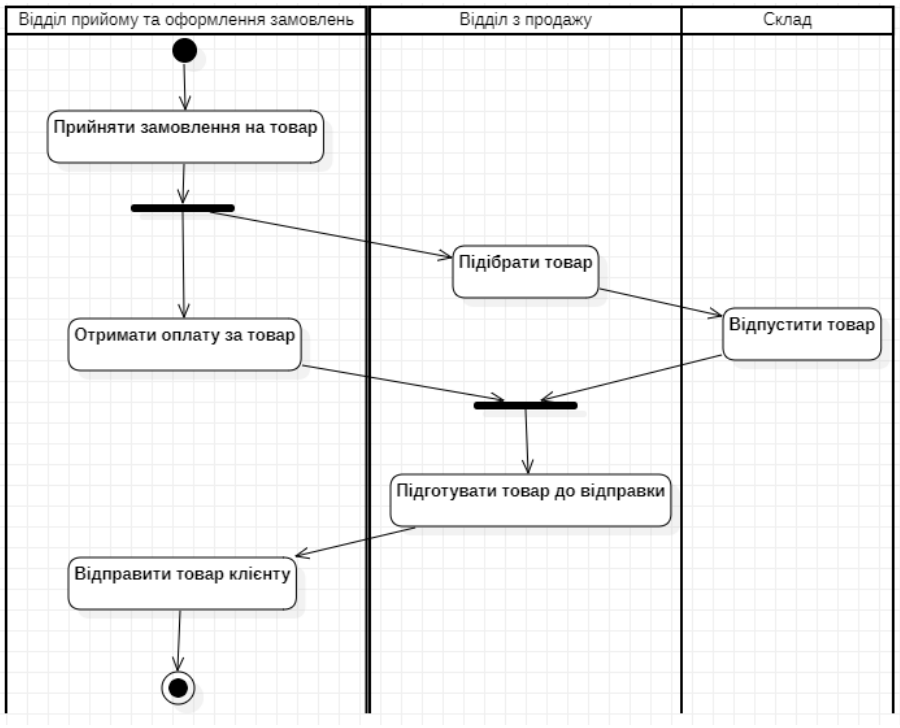


Рисунок 7 - Варіант діаграми діяльності з доріжками

Об'єкти

У загальному випадку дії на діаграмі діяльності виконуються над тими або іншими об'єктами. Ці об'єкти або ініціюють виконання дій, або визначають деякий результат цих дій. При цьому дії специфікують виклики, які передаються від одного об'єкта графа діяльності до іншого. Оскільки в такому ракурсі об'єкти відіграють певну роль у розумінні процесу діяльності, іноді виникає необхідність явно вказати їх на діаграмі діяльності.

Для графічної вистави об'єктів використовуються прямокутник класу, з тим відмінністю, що ім'я об'єкта підкреслюється. Далі після імені може вказуватися характеристика стану об'єкта в прямих дужках. Такі прямокутники об'єктів приєднуються до станів дії відношенням залежності пунктирною лінією зі стрілкою. Відповідна залежність визначає стан конкретного об'єкта після виконання попереднього дії.

На діаграмі діяльності з доріжками розташування об'єкта може мати деякий додатковий зміст. А саме, якщо об'єкт розташовано на границі двох доріжок, те це може означати, що перехід до наступного стану дії в сусідній доріжці асоційований з готовністю деякого документа (об'єкт у деякому стані). Якщо ж об'єкт цілком розташований усередині доріжки, то й стан цього об'єкта цілком визначається діями даної доріжки.

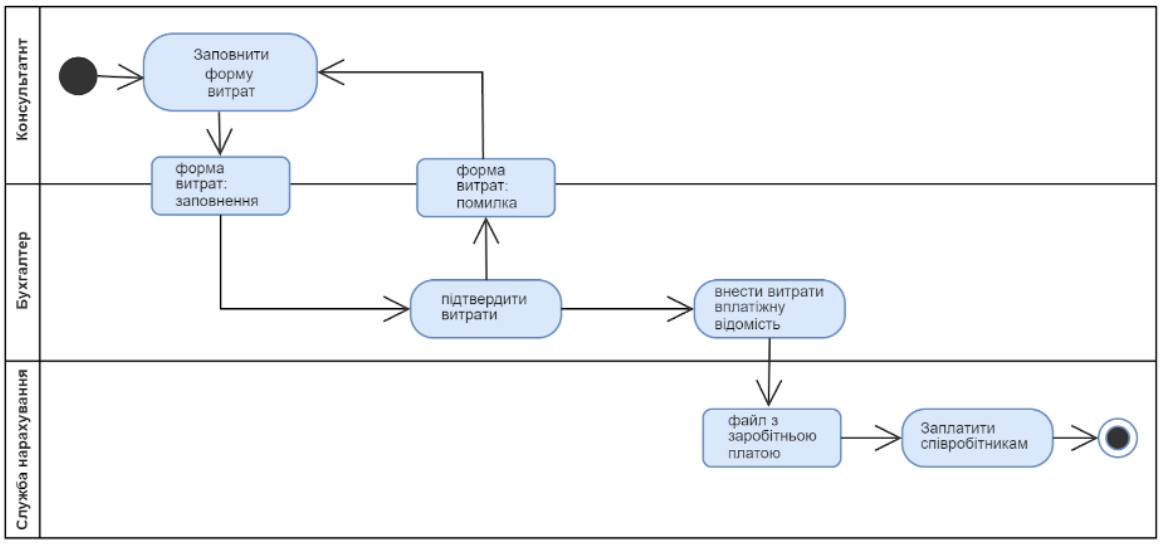


Рисунок 8 - Діаграма виплати заробітної плати

II Завдання до практичної роботи №5

1. По своїй предметній області розробити діаграму діяльності.
2. Діаграма має містити доріжки та об’єкти.
3. При побудові діаграми використовуйте: розподіл, злиття та розгалуження.

III Зробити висновки по виконанню практичної роботи №5

IV Контрольні питання для підготовки к практичній роботі №5

1. В чому різниця діаграми діяльності від діаграми станів?
2. Які елементи використовуються на діаграмі діяльності?
3. Що таке діяльність?
4. Що таке стан дії?
5. Які переходи використовуються на цих діаграмах?
6. Для чого необхідні доріжки?
7. Що відображають об’єкти?
8. Як відобразити паралельні дії?

Практична робота №6

Тема: Побудова діаграми послідовності дій (Sequence Diagram).

Мета: Отримати практичні навики побудови діаграми послідовності.

Хід роботи:

I. Теоретичні відомості

Діаграма послідовності - другий різновид діаграм взаємодії. Відбиваючи сценарій поведінки в системі, ця діаграма забезпечує більш наочне представлення порядку передачі повідомлень. Правда, вона не дозволяє показати такі деталі, які видні на діаграмі співробітництва (структурні характеристики об'єктів і зв'язків).

Об'єкти

На діаграмі послідовності зображуються виняткова ті об'єкти, які безпосередньо беруть участь у взаємодії й не показуються можливі статичні асоціації з іншими об'єктами. Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об'єктів у часі. При цьому діаграма послідовності має як би два виміри.

Перший вимір - ліворуч праворуч у вигляді вертикальних ліній, кожна з яких зображує *лінію життя* окремого об'єкта, що бере участь у взаємодії. Графічно кожний об'єкт зображується прямокутником і розташовується у верхній частині своєї лінії життя (рис. 1). Усередині прямокутника записуються ім'я об'єкта й ім'я класу, розділені двокрапкою. При цьому весь запис підкреслюється, що є ознакою об'єкта, яка, як відомо, являє собою екземпляр класу.

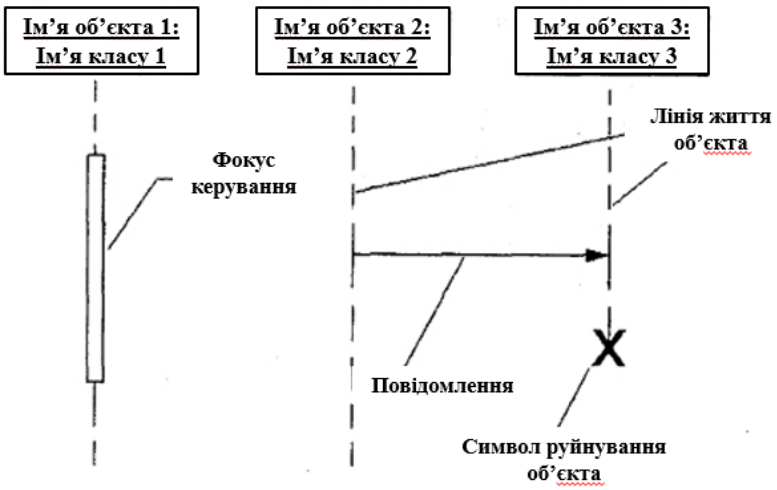


Рисунок 1 - Різні графічні примітиви діаграми послідовності

Крайнім ліворуч на діаграмі зображується об'єкт, який є ініціатором взаємодії (об'єкт 1 на рис. 1). Праворуч зображується інший об'єкт, який безпосередньо взаємодіє з першим. Таким чином, усі об'єкти на діаграмі послідовності утворюють деякий порядок, обумовлений ступенем активності цих об'єктів при взаємодії один з одним.

Другий вимір діаграми послідовності - вертикальна часова вісь, спрямована зверху вниз. Початковому моменту часу відповідає сама верхня частина діаграми. При цьому взаємодії об'єктів реалізуються за допомогою повідомлень, які посилають одними об'єктами іншим. Повідомлення зображуються у вигляді горизонтальних стрілок з іменем повідомлення й також утворюють порядок за часом свого виникнення. Інакше кажучи, повідомлення, розташовані на діаграмі послідовності вище, ініціюються раніше тих, які розташовані нижче. При цьому масштаб на осі часу не вказується, оскільки діаграма послідовності моделює лише тимчасову впорядкованість взаємодій типу " раніше-пізніше".

Лінія життя об'єкта

Лінія життя об'єкта (object lifeline) зображується пунктирною вертикальною лінією, асоційованої з єдиним об'єктом на діаграмі послідовності. Лінія життя служить для позначення періоду часу, протягом якого об'єкт існує в системі й, отже, може потенційно брати участь у всіх її взаємодіях. Якщо об'єкт існує в системі постійно, то і його лінія життя повинна тривати по всій площині діаграми

послідовності від самої верхньої її частини до самої нижньої (об'єкти 1 і 2 на рис. 1).

Окремі об'єкти, виконавши свою роль у системі, можуть бути знищені (зруйновані), щоб звільнити займані ними ресурси. Для таких об'єктів лінія життя обривається в момент його знищення. Для позначення моменту знищення об'єкта в мові UML використовується спеціальний символ у формі латинської букви "X" (об'єкт 3 на рис. 1). Нижче цього символу пунктирна лінія не зображується, оскільки відповідного об'єкта в системі вже ні, і цей об'єкт повинен бути виключений із усіх наступних взаємодій.

Зовсім не обов'язково створювати всі об'єкти в початковий момент часу. Окремі об'єкти в системі можуть створюватися в міру необхідності, суттєво заощаджуючи ресурси системи й підвищуючи її продуктивність. У цьому випадку прямокутник такого об'єкта зображується не у верхній частині діаграми послідовності, а в тій її частині, яка відповідає моменту створення об'єкта (об'єкт 6 на рис. 2). При цьому прямокутник об'єкта вертикально розташовується в тому місці діаграми, яке по осі часу збігається з моментом його виникнення в системі. Очевидно, об'єкт обов'язково створюється зі своєю лінією життя й, можливо, з фокусом керування.

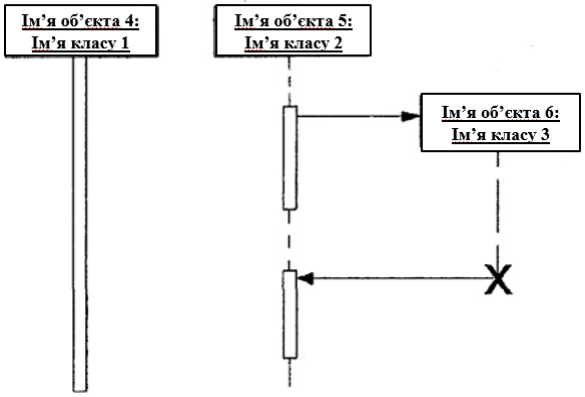


Рисунок 2 - Графічне зображення різних варіантів ліній життя й фокусів керування об'єктів

Фокус керування

У процесі функціонування об'єктно-орієнтованих систем одні об'єкти можуть перебувати в активному стані, безпосередньо виконуючи певні дії або в стані пасивного очікування повідомлень від інших об'єктів. Щоб явно виділити подібну активність об'єктів, у мові UML застосовується спеціальне поняття, що одержала назва фокуса керування (focus of control). Фокус керування зображується у формі витягнутого вузького прямокутника (див. об'єкт 1 на рис. 1), верхня сторона якого позначає початок одержання фокуса управління об'єкта (початок активності), а її нижня сторона - закінчення фокуса керування (закінчення активності). Цей прямокутник розташовується нижче позначення відповідного об'єкта й може заміняти його лінію життя (об'єкт 4 на рис. 2), якщо на всьому її протязі він є активним.

З іншого боку, періоди активності об'єкта можуть чергуватися з періодами його пасивності або очікування. У цьому випадку в такого об'єкта є кілька фокусів керування (об'єкт 5 на рис. 2). Важливо розуміти, що одержати фокус керування може тільки існуючий об'єкт, у якого в цей момент є лінія життя. Якщо ж деякий

об'єкт був знищений, то знову виникнути в системі він уже не може. Замість нього лише може бути створений інший екземпляр цього ж класу, який, строго говорячи, буде іншим об'єктом.

В окремих випадках ініціатором взаємодії в системі може бути актор або зовнішній користувач. У цьому випадку актор зображується на діаграмі послідовності найпершим об'єктом ліворуч зі своїм фокусом керування (рис. 3). Найчастіше актор і його фокус керування будуть існувати в системі постійно, відзначаючи характерну для користувача активність в ініціюванні взаємодій із системою. При цьому сам актор може мати власне ім'я або залишатися анонімним.

Іноді деякий об'єкт може ініціювати рекурсивна взаємодія із самим собою. Мова йде про те, що наявність у багатьох мовах програмування спеціальних засобів побудови рекурсивних процедур вимагає візуалізації відповідних понять у формі графічних примітивів. На діаграмі послідовності рекурсія позначається невеликим прямокутником, приєднаним до правої сторони фокуса керування того об'єкта, для якого зображується ця рекурсивна взаємодія (об'єкт 7 на рис. 3).

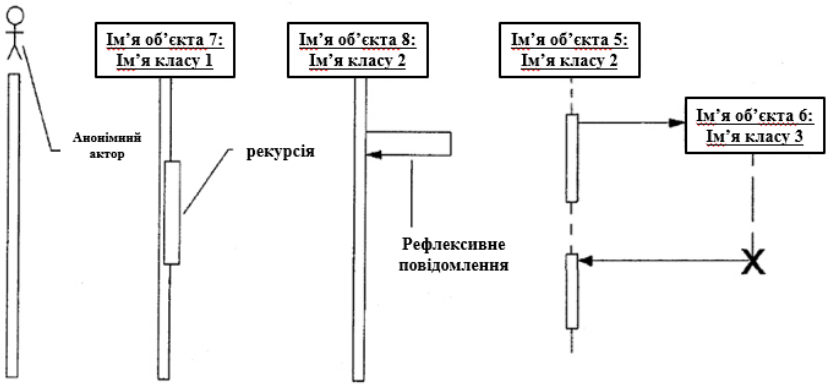


Рисунок 3 - Графічне зображення актора, рекурсії й рефлексивного повідомлення на діаграмі послідовності

Повідомлення

Як було відзначено вище, мета взаємодії в контексті мови UML полягає в тому, щоб специфікувати комунікацію між безліччю взаємодіючих об'єктів. Кожна взаємодія описується сукупністю повідомлень, якими об'єкти обмінюються між собою. У цьому змісті повідомлення (message) являє собою закінчений фрагмент інформації, який відправляється одним об'єктом іншому. При цьому приймання повідомлення ініціює виконання певних дій, спрямованих на розв'язок окремого завдання тим об'єктом, якому це повідомлення відправлене.

Таким чином, повідомлення не тільки передають деяку інформацію, але й вимагають або припускають від об'єкта, що ухвалює, виконання очікуваних дій.

Повідомлення можуть ініціювати виконання операцій об'єктом відповідного класу, а параметри цих операцій передаються разом з повідомленням. На діаграмі послідовності всі повідомлення впорядковані за часом свого виникнення в системі, яка моделюється.

У такому контексті кожне повідомлення має напрямок від об'єкта, який ініціює й відправляє повідомлення, до об'єкта, який його одержує. Іноді відправника повідомлення називають клієнтом, а одержувача-сервером. При цьому повідомлення від клієнта має форму запиту деякого сервісу, а реакція сервера на

запит після одержання повідомлення може бути пов'язана з виконанням певних дій або передачі клієнтові необхідної інформації теж у формі повідомлення.

У мові UML можуть зустрічатися кілька різновидів повідомлень, кожне з яких має своє графічне зображення.

Повідомлення бувають:

1. Синхронні, що очікують відповіді (у вигляді суцільної лінії зі стрілкою).
2. Асинхронні – не очікують відповіді (у вигляді пунктирної лінії зі стрілкою).
3. Рекурсивні – направлені до себе (починаються та завершуються на одній лінії життя).

Звичайно повідомлення зображуються горизонтальними стрілками, що з'єднують лінії життя або фокуси керування двох об'єктів на діаграмі послідовності. При цьому неявно передбачається, що час передачі повідомлення досить малий в порівнянні із процесами виконання дій об'єктами. Вважається також, що за час передачі повідомлення з відповідними об'єктами не може відбутися ніяких подій. Інакше кажучи, стани об'єктів залишаються без зміни. Якщо ж це припущення не може бути визнане слушним, то стрілка повідомлення зображується під деяким нахилом, так щоб кінець стрілки розташовувався нижче її початку.

В окремих випадках об'єкт може посилати повідомлення самому собі, ініціюючи так звані рефлексивні повідомлення (об'єкт 8 на рис.3). Такі повідомлення зображуються прямокутником зі стрілкою, початок і кінець якої збігаються. Подібні ситуації виникають, наприклад, при обробці натискань на клавіші клавіатури при введенні тексту в документ, що редагується, при наборі цифр номера телефону абонента.

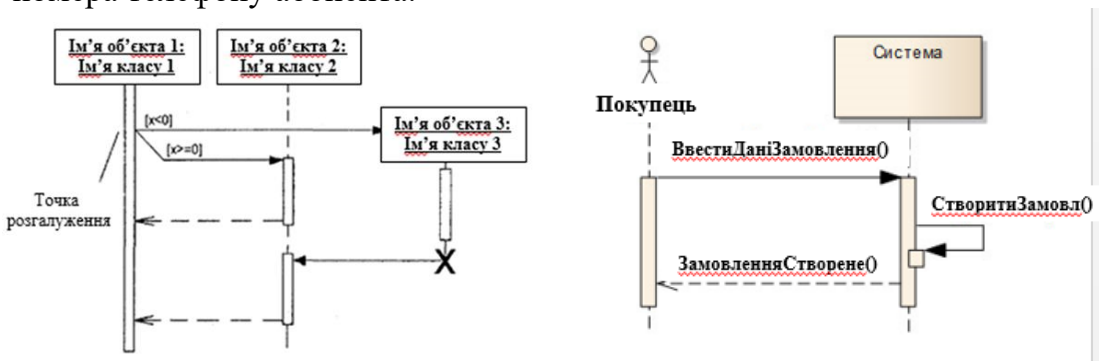


Рисунок 4 - Графічне зображення різних видів повідомлень між об'єктами на діаграмі послідовності

Таким чином, у мові UML кожне повідомлення асоціюється з деякою дією, яка повинна бути виконана його об'єктом, що прийняв. При цьому дія може мати деякі аргументи або параметри, залежно від конкретних значень яких може бути отриманий різний результат. Відповідні параметри буде мати й зухвала ця дія повідомлення. Більше того, значення параметрів окремих повідомлень можуть містити умовні вираження, утворюючи розгалуження або альтернативні шляхи основного потоку керування.

Розгалуження потоку керування

Для зображення розгалуження рисуються дві або більше стрілки, що виходять із однієї крапки фокуса керування об'єкта (фокус керування об'єкта 1 на рис. 5).

При цьому відповідні умови повинні бути явно зазначені поруч із кожної зі стрілок у формі сторожової умови. Як неважко представити, якщо умова записана у формі булевского виразу, то розгалуження буде містити тільки дві галузі. У кожному разі умови повинні взаємно виключати одночасну передачу альтернативних повідомлень.

II Завдання до практичної роботи №6

По власній предметній області побудуйте діаграму послідовностей дій.

III Зробити висновки по виконанню практичної роботи №6

IV Контрольні питання для підготовки к практичній роботі №6

1. Для чого призначена діаграма послідовностей?
2. Як на діаграмі послідовностей зображуються об’єкти?
3. Що таке лінія об'єкта та як вона зображується на діаграмі послідовностей?
4. Для позначення чого в мові UML використовується спеціальний символ у формі латинської букви "X"?
5. Що таке фокус керування та яким чином він зображується на діаграмі послідовностей?
6. Для чого призначені повідомлення на діаграмі послідовностей?

Практична робота №7

Тема: Побудова діаграми співпраці (кооперації) (Collaboration Diagram)

Мета: Отримати практичні навики побудови діаграми співпраці(кооперацій).

Хід роботи:

I. Теоретичні відомості

Діаграми взаємодії призначені для моделювання динамічних аспектів системи. Діаграма взаємодії показує взаємодію, що включає набір об'єктів і їх відносин, а повідомлення, що також пересилаються між об'єктами.

Існують два різновиди діаграми взаємодії - діаграма послідовності й діаграма кооперацій.

Діаграма послідовності - це діаграма взаємодії, яка виділяє впорядкування повідомлень за часом.

Діаграма кооперацій - це діаграма взаємодії, яка виділяє структурну організацію об'єктів, що посилають повідомлення, що її ухвалюють. Елементами діаграм взаємодії є учасники взаємодії - об'єкти, зв'язки, повідомлення.

Діаграми співробітництва (кооперації або співпраці) (collaboration diagram)

Подібно до діаграм послідовності, діаграми кооперації відображають потік подій у конкретному сценарії варіанта використання. Головна особливість діаграми кооперації полягає в можливості графічно уявити не тільки послідовність взаємодії, а й усі структурні відносини між об'єктами, що беруть участь у цій взаємодії.

Насамперед на діаграмі кооперації у вигляді прямокутників зображуються об'єкти, які беруть участь у взаємодії, що містять ім'я об'єкта, його клас і, можливо, значення атрибутів. Далі, як і на діаграмі класів, вказуються асоціації між об'єктами у вигляді різних сполучних ліній. При цьому можна явно вказати імена

асоціації та ролей, які відіграють об'єкти в даній асоціації. Додатково можуть бути зображені динамічні зв'язки - потоки повідомлень. Вони подаються також у вигляді сполучних ліній між об'єктами, над якими розташовується стрілка із зазначенням напрямку, імені повідомлення та порядкового номера в загальній послідовності ініціалізації повідомлень.

На відміну від діаграми послідовності, на діаграмі кооперації зображуються тільки відносини між об'єктами, які відіграють певні ролі у взаємодії, а послідовність взаємодій і паралельних потоків визначається за допомогою порядкових номерів.

Об'єкти діаграми кооперації

Окремі аспекти специфікації об'єктів як елементів діаграм уже розглядалися раніше під час опису діаграм послідовності. Ці ж об'єкти є основними елементами з яких будується діаграма кооперації. Для графічного зображення об'єктів використовується такий самий символ прямокутника, що й для класів.

Об'єкт є окремим екземпляром класу, який створюється на етапі виконання програми. Він може мати своє власне ім'я та конкретні значення атрибутів. Для позначення ролі класифікатора необхідно вказати або ім'я класу (разом із двокрапкою), або ім'я ролі (разом із похилою рискою). В іншому разі прямокутник відповідатиме звичайному класу. Якщо роль, яку має відігравати об'єкт, успадковується від кількох класів, то всі вони мають бути зазначені явно і розмежовуватися комою і двокрапкою.

Окремі приклади зображення об'єктів і класів на діаграмі кооперації наводяться на рис. 1. У першому випадку (рис. 1, а) позначено об'єкт з ім'ям «клієнт», що грає роль «ініціатор запиту». Далі (рис. 1, б) слідує позначення анонімного об'єкта, який відіграє роль ініціатора запиту. В обох випадках не вказано клас, на основі якого буде створено ці об'єкти. Позначення класу присутнє в наступному варіанті запису (рис. 1, в), причому об'єкт також анонімний.

Рисунок 1 - Варіанти запису імен об'єктів, ролей і класів на діаграмах кооперації

Стосовно рівня специфікації на діаграмах кооперації можуть бути присутніми іменовані класи із зазначенням ролі класу в кооперації (рис. 1, г) або анонімні класи, коли зазначено тільки його роль (рис. 1, д). Останній випадок характерний для ситуації, коли в моделі можуть бути присутніми кілька класів з ім'ям «Клієнт», тому потрібно явно вказати ім'я відповідного пакету База даних (рис. 1, є).

Мультиоб'єкт (multiobject) являє собою цілу множину об'єктів на одному з кінців асоціації (рис. 2, а). На діаграмі кооперації мультиоб'єкт використовується для того, щоб показати операції та сигнали, адресовані всій безлічі об'єктів, а не

тільки одному. При цьому стрілка повідомлення відноситься до всієї множини об'єктів, які позначають даний мультиоб'єкт. На діаграмі кооперації може бути явно вказано відношення композиції між мультиоб'єктом і окремим об'єктом із його множини (рис. 2, б).

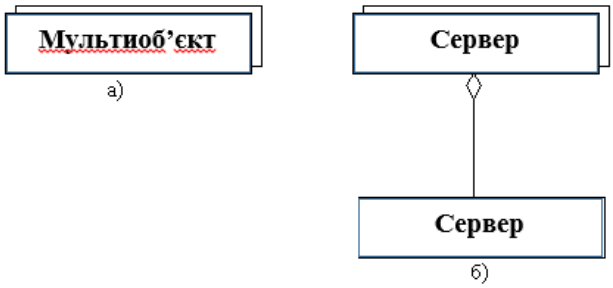


Рисунок 2 - Графічне зображення мультиоб'єктів на діаграмі кооперації

У контексті мови UML усі об'єкти поділяються на дві категорії: пасивні та активні. Пасивний об'єкт оперує тільки даними і не може ініціювати діяльність з управління іншими об'єктами. Водночас пасивні об'єкти можуть посилати сигнали в процесі виконання запитів, які вони отримують.

Активний об'єкт має свою власну нитку управління і може ініціювати діяльність з управління іншими об'єктами. При цьому під ниткою розуміється потік керування, який може виконуватися паралельно з іншими обчислювальними нитками або нитками керування в межах одного обчислювального процесу.

У наведеному фрагменті діаграми кооперації (рис. 3) активний об'єкт "a: Абонент, що викликає" є ініціатором процесу встановлення з'єднання для обміну інформацією з іншим абонентом.

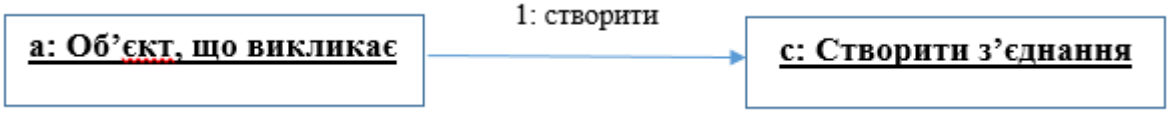


Рисунок 3 - Активний об'єкт (ліворуч) на діаграмі кооперації

Складений об'єкт (composite object) або об'єкт-контейнер призначений для представлення об'єкта, що має власну структуру та внутрішні потоки (нитки) управління. Складений об'єкт є екземпляром складеного класу (класу-контейнера), який пов'язаний відношенням агрегації або композиції зі своїми частинами. Аналогічні відносини пов'язують між собою і відповідні об'єкти.

На діаграмах кооперації складовий об'єкт складається з двох секцій: верхньої та нижньої. У верхній секції записується ім'я складового об'єкта, а в нижній - його елементи (рис. 4), які можуть бути складовими об'єктами.

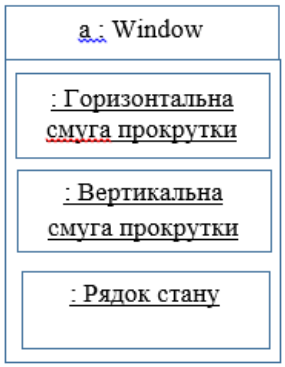


Рисунок 4 - Складовий об'єкт на діаграмі кооперації

Зв'язок (link) є екземпляром або прикладом довільної асоціації. Зв'язок як елемент мови UML може мати місце між двома і більше об'єктами. Зв'язок на діаграмі кооперації зображується відрізком прямої лінії, що з'єднує два прямокутники об'єктів. На кожному з кінців цієї лінії можуть бути явно вказані імена ролей даної асоціації. Поруч із лінією в її середній частині може записуватися ім'я відповідної асоціації. Зв'язки не мають власних імен, оскільки повністю ідентичні як екземпляри асоціації. Для зв'язків не вказується також і кратність.

Стосовно діаграм кооперації повідомлення мають деякі додаткові семантичні особливості. Вони визначають комунікацію між двома об'єктами, один з яких передає іншому деяку інформацію. При цьому перший об'єкт очікує, що після отримання повідомлення другим об'єктом послідує виконання деякої дії. Таким чином, саме повідомлення є причиною або стимулом для початку виконання операцій, надсилання сигналів, створення і знищення окремих об'єктів. Зв'язок забезпечує канал для спрямованої передачі повідомлень між об'єктами від об'єкта-джерела до об'єкта-одержувача.

Приклад діаграми кооперації

На рис. 5 наведено діаграму кооперацій, що описує, як клієнт знімає з рахунку 20$.

Рисунок 5 - Діаграма кооперацій для зняття клієнтом 20$

З діаграми кооперацій легше зрозуміти потік подій і відносини між об'єктами, проте важче усвідомити послідовність подій, тому для сценарію створюють діаграми обох типів.

II Завдання до практичної роботи №7

По власній предметній області побудуйте діаграму кооперацій.

III Зробити висновки по виконанню практичної роботи №7

IV Контрольні питання для підготовки к практичній роботі №7

1. Як представляється ім'я об'єкта в діаграмі кооперацій?
2. Що таке мультиоб’єкт на діаграмі кооперацій?
3. Що таке зв’язок?
4. Що загального в діаграмі послідовності й діаграмі співробітництва? Чим вони відрізняються друг від друга?

Практична робота №8

Тема: Побудова діаграми компонентів (Component Diagram)

Мета: Отримати практичні навики побудови діаграми компонентів.

Хід роботи:

I. Теоретичні відомості

Компонентна діаграма - перша із двох різновидів діаграм реалізації, що моделюють фізичні аспекти об'єктно-орієнтованих систем. Компонентна діаграма показує організацію набору компонентів і залежності між компонентами.

Елементами компонентних діаграм є компоненти й інтерфейси, а також відношення залежності й реалізації. Як і інші діаграми, компонентні діаграми можуть включати примітки й обмеження. Крім того, компонентні діаграми можуть містити пакети або підсистеми, використовувані для угруповання елементів моделі у великі фрагменти.

Компоненти

По своїй суті компонент є фізичним фрагментом реалізації системи, який містить у собі програмний код (вихідний, двійковий, що виконується), сценарні описи або набори команд операційної системи (маються на увазі командні файли). Мова UML дає наступне визначення.

Компонент - фізична й замінна частина системи, яка відповідає набору інтерфейсів і забезпечує реалізацію цього набору інтерфейсів.

Інтерфейс - дуже важлива частина поняття "компонент". Графічно компонент зображується як прямокутник із вкладками, що звичайно включає ім'я (рис. 1).

Рисунок 1 - Позначення компонента

Як показано на рис. 2, за допомогою відношення залежності можна явно відобразити відношення між компонентом і класами, які він реалізує.

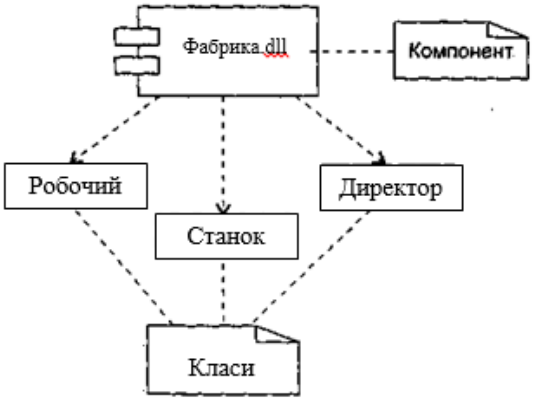


Рисунок 2 - Класи в компоненті

Інтерфейси

Інтерфейс - список операцій, які визначають послуги класу або компонента.

Компонування системи

За останні піввіку розроблювачі апаратури пройшли шлях від комп'ютерів розміром з кімнату до малюсіньких "ноутбуків", що забезпечили зрослі функціональні можливості. За ті ж піввіку розроблювачі програмного забезпечення пройшли шлях від великих систем на Асемблері й Фортрані до ще більших систем на C++ і Java. На жаль, але програмний інструментарій розбудовується повільніше, чим апаратний інструментарій.

Цей секрет - компонента. Розроблювач апаратури створює систему з готових апаратних компонентів (мікросхем), що виконують певні функції, що й надають набір послуг через ясні інтерфейси. Завдання конструкторів спрощується за рахунок повторного використання результатів, отриманих іншими.

Повторне використання - магістральний шлях розвитку програмного інструментарію. Створення нового ПЗ з існуючих, працездатних програмних компонентів приводить до більш надійного й дешевого коду. При цьому строки розробки суттєво скорочуються.

Основна мета програмних компонентів - допускати складання системи із двійкових замінних частин. Вони повинні забезпечити початкове створення системи з компонентів, а потім і її розвиток - додавання нових компонентів і заміну деяких старих компонентів без перебудови системи в цілому. Ключ до втілення такої можливості - інтерфейси. Після того як інтерфейс визначений, до виконуваної системи можна підключити будь-який компонент, який задовольняє йому або забезпечує цей інтерфейс. Для розширення системи проводяться компоненти, які забезпечують додаткові послуги через нові інтерфейси. Такий підхід ґрунтується на особливостях компонента:

- Компонент фізичний. Він живе у світі бітів, а не логічних понять і не залежить від мови програмування
- Компонент - замінний елемент. Властивість заменяемості дозволяє замінити один компонент іншим компонентом, який задовольняє тим же інтерфейсам. Механізм заміни застережений сучасними компонентними моделями (COM, COM+, CORBA, Java Beans), що вимагають незначних перетворень або, що надають утиліти, які автоматизують механізм

- Компонент є частиною системи, він рідко автономний. Частіше компонент співпрацює з іншими компонентами й існує в архітектурній або технологічному середовищі, призначеної для його використання. Компонент зв'язаний і фізично, і логічно, він позначає фрагмент великої системи
- Компонент відповідає набору інтерфейсів і забезпечує реалізацію цього набору інтерфейсів

Різновиди компонентів

Мир сучасних компонентів досить широкий і різноманітний. У мові UML для позначення нових різновидів компонентів використовують механізм стереотипів. Стандартні стереотипи, передбачені в UML для компонентів:

"executable" - Компонент, який може виконуватися у фізичному вузлі (має розширення .exe)

"library" - Статична або динамічна об'єктна бібліотека (має розширення .dll)

"file" - Компонент, який представляє файл, що містить вихідний код або дані (має розширення .ini)

"table" - Компонент, який представляє таблицю бази даних (має розширення .tbl)

"document" - Компонент, який представляє документ (має розширення .hlp)

У мові UML не визначені піктограми для перерахованих стереотипів, застосовувані на практиці піктограми компонентів показані на рис. 3-7.

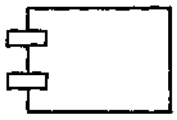


Рисунок 3 – Піктограма елемента що виконується.

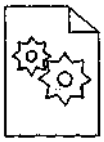


Рисунок 4 – Піктограма об’єктної бібліотеки

Рисунок 5 – Піктограма документу з ісходним кодом

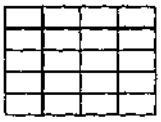
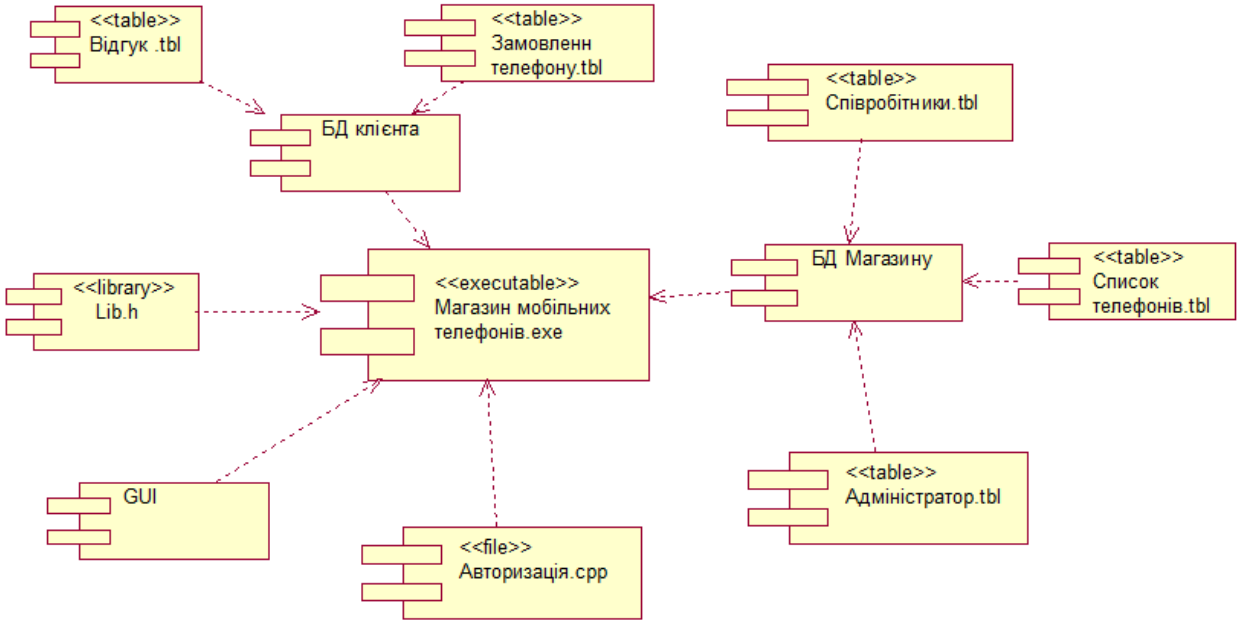


Рисунок 6 – Піктограма таблиці даних БД

Рисунок 7 – Піктограма документу

Приклад:



II Завдання до практичної роботи №8

1. Зробити моделювання реалізації системи з використанням відомих видів компонентів. Для компонентів вказати інтерфейси.
2. Зробити моделювання реалізації системи з використанням спеціальних піктограм.

III Зробити висновки по виконанню практичної роботи №8

IV Контрольні питання для підготовки к практичній роботі №8

1. Як задається ім'я компонента?
2. Що прийнято використовувати в якості простих імен?
3. Що таке компонент?

Практична робота №9

Тема: Побудова діаграми розгортання (Deployment Diagram).

Мета: Отримати практичні навики побудови діаграми розгортання.

Хід роботи:

I. Теоретичні відомості

Діаграма розміщення (розгортання) - друга із двох різновидів діаграм реалізації UML, що моделюють фізичні аспекти об'єктно-орієнтованих систем. Діаграма розміщення показує конфігурацію обробних вузлів у період роботи системи, а також компоненти, "живучі" у них.

Елементами діаграм розміщення є вузли, а також відносини залежності й асоціації. Як і інші діаграми, діаграми розміщення можуть включати примітки й обмеження. Крім того, діаграми розміщення можуть включати компоненти, кожний з яких повинен жити в деякому вузлі, а також містити пакети або підсистеми, використовувані для угруповання елементів моделі у великі

фрагменти. При необхідності візуалізації конкретного варіанта апаратної топології в діаграми розміщення можуть міститися об'єкти.

Вузли

Вузол - фізичний елемент, який існує в період роботи системи й представляє комп'ютерний ресурс, що має пам'ять, а можливо, і здатність обробки. Графічно вузол зображується як куб з іменем (рис. 1).

Рисунок 1 - Позначення вузла

Як і клас, вузол може мати додаткову секцію, що відображає розташовувані в ньому елементи (рис. 2).

Рисунок 2 - Розміщення компонентів у вузлі

Зрівняємо вузли з компонентами. Звичайно, у них є подібні характеристики:

- наявність імені;
- можливість бути вкладеним;
- наявність екземплярів.

Остання характеристика говорить про те, що на попередньому рисунку зображений тип Контролера. Зображення конкретного екземпляра, що належить цьому типу, презентовано на рис. 3.

Тепер обговоримо відмінності вузлів від компонентів. По-перше, вони належать до різних рівнів ієрархії у фізичній реалізації системи. Фізично система складається з вузлів, а вузли - з компонентів. По-друге, у кожного з них своє призначення. Компонент призначений для фізичного впакування й матеріалізації набору логічних елементів (класів і кооперацій). Вузол же є тем місцем, де фізично розміщаються компоненти, тобто відіграє роль "квартири" для компонентів.

Рисунок 3 - Екземпляр вузла

Відношення між вузлом і компонентами, які він розміщує, можна відобразити явно. Відношення залежності між вузлом Контролер і компонентами Введення.exe, Висновки.exe ілюструє рис. 4. Правда, найчастіше такі відносини не відображаються. Їх зручно представляти в окремій специфікації вузла.

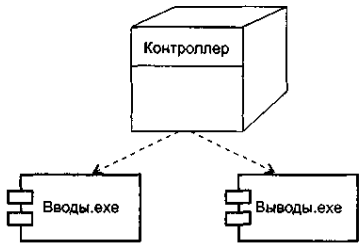


Рисунок 4 - Залежність вузла від компонентів

Угруповання набору об'єктів або компонентів, розташованих у вузлі, звичайно називають *розповсюджуваним модулем*.

Графічно діаграма розміщення - це граф з вузлів (або екземплярів вузлів), з'єднаних асоціаціями, які показують існуючі комунікації. Екземпляри вузлів можуть містити екземпляри компонентів, що живуть або запускаються у вузлах. Екземпляри компонентів можуть містити об'єкти. Як показано на рис. 5, компоненти з'єднуються один з одним пунктирними стрілками залежностей (прямо або через інтерфейси).

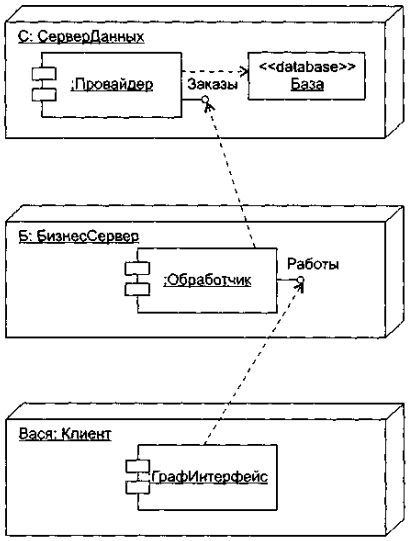


Рисунок 5 - Моделювання розміщення компонентів

На цій діаграмі зображена типова трирівнева система:

1. рівень бази даних реалізований екземпляром С вузла СерверДанных;
2. рівень бізнес-логіки представлений екземпляром Б вузла БизнесСервер;
3. рівень графічного інтерфейсу користувача утворений екземпляром Вася вузла Клієнт.

II Завдання до практичної роботи №9

Зробити побудову діаграми розгортання, базуючись на клієнт-серверному проекті, створеним раніше на курсі дисципліни БД.

III Зробити висновки по виконанню практичної роботи №9

IV Контрольні питання для підготовки к практичній роботі №9

1. Які вершини й ребра утворюють діаграму розгортання?
2. Чим відрізняється вузол від компонента?
3. Де можна використовувати й де не можна використовувати екземпляри компонентів?