МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКИЙ КОЛЕГІУМ»
імені Т.Г. ШЕВЧЕНКА

Основи інформаційного моделювання

МЕТОДИЧНІ ВКАЗІВКИ
З ДИСЦИПЛІНИ «Інформаційне моделювання»

Чернігів НУЧК 2025

УДК 004.94 (072)
0-75

Укладачі:
Горошко Юрій Васильович, доктор педагогічних наук, професор кафедри інформатики і обчислювальної техніки Національного університету «Чернігівський колегіум» імені Т.Г. Шевченка
Цибко Ганна Юхимівна, кандидат педагогічних наук, доцент кафедри інформатики і обчислювальної техніки Національного університету «Чернігівський колегіум» імені Т.Г. Шевченка

Горошко Ю.В., Цибко Г.Ю.

0-75 Основи інформаційного моделювання. Методичні вказівки з дисципліни "Інформаційне моделювання" / укладачі: Горошко Ю.В., Цибко Г.Ю., Чернігів: НУЧК, 2024, 85 с.

Затверджено вченою радою природничо-математичного факультету Національного університету «Чернігівський колегіум» імені Т.Г. Шевченка, протокол №5 від 16.12.2024 р.

Рецензенти:
Кандидат фізико-математичних наук, доцент, доцент кафедри інформаційних управляючих систем та технологій ДВНЗ "Ужгородський національний університет" Шапочка І.В.
Кандидат педагогічних наук, доцент, доцент кафедри інформатики і обчислювальної техніки Національного університету "Чернігівський колегіум" імені Т.Г.Шевченка Вінниченко Є.Ф.

Методичні рекомендації складено для здобувачів освіти, які навчаються за освітньою програмою 122 Комп'ютерні науки першого (бакалаврського) рівня вищої освіти. Основною метою методичних вказівок є формування у студентів знань та практичних навичок з дисципліни "Інформаційне моделювання".

Зміст

1. Поняття системи. Ознаки складних систем....................................4

2. Принципи структурно-функціонального моделювання..............6

3. Загальні принципи структурних методів та структурного аналізу...9

4. Об’єктно-орієнтоване проектування..12

5. Засади структурно-функціональної методології.........................15

6. Концепція розробки інформаційної системи підприємства на основі структурного підходу..18

7. Інструментальні засоби структурного проектування.................21

8. Универсальна мова моделювання (UML)...................................24

8.1. Діаграма класів...28

8.2. Діаграма прецедентів (Use Case Diagram)..................................45

8.3. Діаграма послідовності..56

8.4. Діаграма діяльності..64

9. StarUML..71

10. Проєктування інтерфейсів інформаційних систем....................73

11. Завдання до заключної лабораторної роботи.............................76

Список використаних джерел..82

1. Поняття системи. Ознаки складних систем

Систе́ма (віддав.-гр. Σύστημα — «сполучення», «ціле», «з'єднання») — множина взаємопов'язаних елементів, що утворюють єдине ціле, взаємодіють із середовищем та між собою, і мають мету.

Складні системи часто є ієрархічними та складаються із взаємозалежних підсистем, які в свою чергу також можуть бути розділені на підсистеми, і т.д., аж до „найнижчого рівня”. Архітектура системи складається як із компонентів, так і з (ієрархічних) відношень цих компонентів.

Вибір, які компоненти в даній системі вважаються елементарними, є відносно довільним та в великій мірі покладається на дослідника. Нижній рівень для одного користувача може виявитись достатньо високим для іншого.

Внутрішньокомпонентний зв’язок зазвичай є сильнішим, ніж зв’язок між компонентами. Це дозволяє відділяти „високочастотні” взаємодії всередині компонент від „низькочастотної” динаміки взаємодії між компонентами .

Ієрархічні системи зазвичай складаються з небагатьох типів підсистем, по-різному скомбінованих та організованих.

Будь-яка працююча складна система є результатом розвитку більш простої системи, що працювала раніше. Складна система, спроектована „з нуля”, ніколи не запрацює. Слід починати з працюючої простішої системи.

Приклади складних систем:

- «астрономічний» всесвіт;
- літак;
- диспетчерська система управління повітряним рухом;
- ринкова економіка.

Великі програмні продукти також є складними системами. Рівень складності задається складністю задач, які беруться з життя. Наступні фактори поглиблюють проблему:

- „Неузгодженість”між користувачами і розробниками: користувачу дуже важко пояснити розробнику, що треба робити. У них різні знання і досвід, їм не вистачає знань співрозмовника. Користувач на початку лише приблизно уявляє, що йому треба.
- Як наслідок вимоги до системи змінюються уже в процесі розробки. В основному це відбувається тому, що виконання проекту змінює проблему: розгляд перших результатів (прототипів) та тим більше використання системи дозволяє користувачам краще зрозуміти потреби. Це стосується і розробників.
- За винятком тривіальних випадків, всебічне тестування таких систем провести неможливо. Ми повинні задовольнитись зваженим рівнем упевненості в їхній правильності.

Щодо розробки *промислових програмних продуктів, то* вони характеризуються:

- Великою тривалістю життєвого циклу (ERP – 15 років),
- Великою кількістю користувачів, які сильно залежать від нормального функціонування системи.

Приклади:

- Системи керування складними технічними об’єктами (літак, корабель, електростанція).
- Системи контролю за діяльністю людини (диспетчеризація повітряного та залізничного транспорту; система обліку та управління підприємством (класу ERP)).
- Система управління базою даних (сотні тисяч і мільйони записів, десятки одночасних оновлень).

Головна риса промислового ПЗ щодо даної дисципліни – *рівень складності*: один розробник практично не в змозі охопити всі аспекти системи. Рівень складності промислового ПЗ перевищує можливості людського інтелекту.

Спосіб управління складними системами був відомий ще з давніх часів – *divide et imperia* (поділяй і владарюй).” – Дейкстра.

У процесі проектування складної програмної системи необхідно поділяти її на все менші та менші підсистеми, кожну з яких можна вдосконалювати незалежно. В цьому разі ми не перевищимо пропускної здатності нашого обмеженого мозку (7+-2 об’єкти одночасно): для розуміння будь-якого рівня системи нам треба одночасно тримати в голові лише кілька її частин.

З курсу програмування відомо про структурне проектування „зверху вниз”, і декомпозиція в цьому разі сприймається як просте розділення алгоритмів, де кожний модуль системи виконує один етап процесу.

Розглянемо альтернативний спосіб декомпозиції. Перш ніж розділяти задачу на кроки, спочатку потрібно визначити об’єкти предметної галузі.
У цій декомпозиції світ представлено сукупністю автономних діючих осіб, які взаємодіють один з одним для забезпечення поведінки системи.

Таким чином,

- кожний об’єкт має власну поведінку,
- кожний об’єкт моделює певний об’єкт реального світу.

Об’єкти дещо можуть робити, і якщо послати об’єкту повідомлення, можна попросити його виконати ту чи іншу дію.

Більш детально про це буде викладено у наступних параграфах.

2. Принципи структурно-функціонального моделювання

Принципи структурно-функціонального моделювання в інформаційному проєктуванні допомагають створювати інформаційні системи, які є добре

організованими, функціональними та здатними до адаптації. Ці принципи визначають, як потрібно структурувати елементи системи та надавати їм відповідні функції, щоб забезпечити оптимальне виконання завдань і легке впровадження змін.

Основні принципи структурно-функціонального моделювання в інформаційному проєктуванні

1. **Принцип цілісності системи**

• Система розглядається як цілісна структура, в якій усі елементи взаємодіють для досягнення спільної мети. Кожен компонент відіграє свою роль у контексті всієї системи, забезпечуючи її функціонування як єдиного організму.

• **Приклад**: В інформаційній системі управління персоналом (HRM) модулі для зберігання даних співробітників, обробки зарплат і оцінки ефективності працюють разом, щоб забезпечити злагоджену роботу системи.

2. **Принцип ієрархічної структури**

• Інформаційна система організовується як ієрархія, де на різних рівнях є підсистеми та компоненти з власними функціями, що забезпечує чітку організацію і відповідальність кожного рівня.

• **Приклад**: У складній корпоративній системі є рівні доступу для адміністраторів, менеджерів і користувачів, кожен з яких має різний доступ до даних і функцій системи.

3. **Принцип функціональної диференціації**

• Кожен елемент системи має конкретну функцію, що забезпечує виконання певної частини завдань. Ця диференціація дозволяє системі працювати ефективніше та надійніше.

• **Приклад**: В інтернет-магазині функціональні модулі для управління замовленнями, обробки платежів та обслуговування клієнтів мають чітко визначені ролі та функції.

4. **Принцип взаємозв’язку елементів**

• Всі компоненти системи взаємопов’язані, а зміна одного елемента може вплинути на інші. Цей взаємозв’язок забезпечує ефективну комунікацію між модулями та підвищує продуктивність системи.

• **Приклад**: Зміна структури бази даних може призвести до необхідності оновлення інтерфейсу користувача, щоб відображати нові дані або структуру.

5. **Принцип модульності**

• Система складається з модулів, які можна змінювати або оновлювати окремо, без значного впливу на інші частини системи. Це дозволяє легко вносити зміни, адаптувати і розширювати систему.

• **Приклад**: В системі управління контентом (CMS) модулі для управління користувачами, створення контенту та аналітики можна розвивати незалежно один від одного.

6. **Принцип адаптивності та гнучкості**

• Система має бути здатна адаптуватися до нових вимог і змін у зовнішньому середовищі. Це забезпечується завдяки гнучкій структурі та можливості швидкого оновлення функцій.

• **Приклад**: В CRM-системі повинна бути можливість легко інтегрувати нові комунікаційні канали (наприклад, чат-боти або соціальні мережі), щоб відповідати сучасним потребам клієнтів.

7. **Принцип ефективного використання ресурсів**

• Структура і функції системи повинні бути оптимізовані для ефективного використання ресурсів (часу, пам’яті, процесорної потужності). Це зменшує витрати і підвищує продуктивність.

• **Приклад**: У великих інформаційних системах використовуються оптимізовані алгоритми для пошуку та сортування даних, що скорочує час обробки запитів.

8. **Принцип стандартизації та уніфікації**

• Компоненти і функції системи повинні бути стандартизовані для забезпечення їх сумісності і взаємозамінності. Це полегшує інтеграцію нових елементів у систему.

• **Приклад**: У розподілених системах стандартизовані API дозволяють інтегрувати різні сервіси без необхідності змінювати базову структуру.

9. **Принцип безперервного вдосконалення**

• Система повинна бути побудована з урахуванням можливостей для розвитку і модернізації. Це дозволяє постійно покращувати її функціональність та відповідати новим вимогам.

• **Приклад**: Розробка програмного забезпечення з регулярними оновленнями, які включають нові функції і виправлення, забезпечує актуальність і надійність системи.

Принципи структурно-функціонального моделювання дозволяють створювати інформаційні системи, які є оптимально організованими і здатними адаптуватися до вимог користувачів та технологічних змін. У сучасних проєктах, таких як ERP-системи, CRM-системи, веб-додатки, використання цих принципів забезпечує стабільність, масштабованість і високу продуктивність.

3. Загальні принципи структурних методів та структурного аналізу

Загальні принципи структурних методів та структурного аналізу в інформаційному проєктуванні зосереджуються на системному підході до моделювання та аналізу складних інформаційних систем. Вони допомагають забезпечити чітку структуру, зручність управління, ефективне використання ресурсів і адаптивність системи до змін. Ці принципи широко застосовуються для створення модульних, масштабованих та надійних систем, що відповідають потребам користувачів.

Основні принципи структурних методів та структурного аналізу в інформаційному проєктуванні

1. **Принцип декомпозиції**

• Декомпозиція розділяє систему на більш дрібні частини або модулі, кожен з яких виконує певну функцію. Це дозволяє розробникам зосередитися на конкретних функціональних аспектах системи і полегшує процес аналізу та проектування.

• **Приклад**: При проектуванні ERP-системи можна виділити модулі для фінансів, кадрів, закупівель, що дозволяє окремим командам розробників працювати над специфічними завданнями кожного модуля.

2. **Принцип модульності**

• Система проектується як набір модулів, які можна створювати, тестувати та вдосконалювати окремо. Це полегшує підтримку, оновлення та масштабування системи.

• **Приклад**: У системі електронної комерції окремі модулі, такі як обробка замовлень, управління товаром і платіжна система, можуть бути оновлені або замінені без порушення роботи інших модулів.

3. **Принцип абстракції**

• Абстракція дозволяє спростити складну систему, виділяючи тільки ключові аспекти або функції і приховуючи деталі. Це допомагає розробникам зосередитися на важливих аспектах системи.

• **Приклад**: При створенні інтерфейсу користувача важливо абстрагувати основні функції, щоб користувач бачив тільки необхідні елементи і не взаємодіяв із зайвими технічними деталями.

4. **Принцип ієрархічності**

• Ієрархічність створює рівні в системі, де кожен рівень має певну функціональність і з’єднується з іншими за допомогою визначених інтерфейсів. Це дозволяє структурувати систему, полегшуючи управління та масштабування.

• **Приклад**: У базах даних рівні доступу можуть бути розділені на рівень адміністраторів, менеджерів і кінцевих користувачів, кожен з яких має доступ лише до відповідних даних і функцій.

5. **Принцип інформаційної прихованості**

• Кожен модуль приховує внутрішні деталі реалізації від інших модулів і надає тільки необхідний інтерфейс для взаємодії. Це знижує залежність між компонентами системи і забезпечує гнучкість.

• **Приклад**: Модуль аутентифікації може бути змінений на новий, якщо він підтримує той самий інтерфейс, що і попередній модуль, не порушуючи роботу решти системи.

6. **Принцип повторного використання**

• Компоненти і модулі проектуються так, щоб їх можна було повторно використовувати в різних частинах системи або навіть у різних проектах, що знижує витрати на розробку і тестування.

• **Приклад**: Модуль перевірки електронної пошти може використовуватися і в системі реєстрації користувачів, і в модулі відновлення паролів, зменшуючи обсяг роботи над тестуванням та впровадженням.

7. **Принцип послідовного розвитку**

• Розробка системи відбувається поступово, з поділом на етапи. Це дозволяє впроваджувати зміни в окремих частинах системи без ризику порушення її загального функціонування.

• **Приклад**: Великий веб-проект може спочатку включати базові функції для тестування, а потім можуть поступово додаватись інші функції у міру розвитку системи.

8. **Принцип контролю складності**

• Важливо уникати надмірної складності в структурі і функціонуванні системи. Це досягається за допомогою стандартизації, структуризації і зменшення зайвої функціональності.

• **Приклад**: У веб-додатку управління потоками даних стандартизоване і розподілене за простими правилами для полегшення налагодження і підтримки.

9. **Принцип стандартизації**

• Використання стандартів і єдиних підходів у проектуванні забезпечує сумісність компонентів і полегшує інтеграцію, підтримку та масштабування системи.

• **Приклад**: Веб-додаток може використовувати стандартизовані протоколи API (наприклад, REST або GraphQL), що спрощує інтеграцію з іншими сервісами.

10. **Принцип тестування та валідації**

• Кожен компонент тестується окремо і як частина системи. Це дозволяє виявити помилки на ранніх етапах розробки і забезпечити узгодженість роботи всіх елементів.

• **Приклад**: Автоматизоване тестування проводиться для кожного нового модуля системи, щоб виявити проблеми, перш ніж він буде впроваджений у реальну експлуатацію.

Дотримання цих принципів забезпечує структурованість, зручність і гнучкість інформаційних систем, сприяє кращому управлінню проєктами, знижує ризики і підвищує надійність системи. Вони особливо корисні в великих проєктах, таких як ERP, CRM-системи, де структура та функції повинні бути чітко визначеними, щоб досягти бажаних результатів.

4. Об'єктно-орієнтоване проектування

Об'єктно-орієнтоване проектування (ООП) – це підхід до розробки програмного забезпечення, який використовує концепції об'єктів та класів для створення гнучких, масштабованих і легко підтримуваних систем. ООП дозволяє моделювати реальні процеси, структури та взаємозв'язки, відображаючи їх у вигляді об'єктів з певними властивостями та поведінкою. Цей підхід широко використовується в сучасному програмуванні завдяки своїй

здатності організовувати та структурувати код, роблячи його зрозумілішим і простішим у підтримці.

Основні концепції об’єктно-орієнтованого проектування

1. **Клас і Об’єкт**

• **Клас** — це шаблон або тип, що визначає властивості (атрибути) та поведінку (методи), яку можуть мати об’єкти цього класу.

• **Об’єкт** — це конкретна реалізація класу, екземпляр, який має власні значення атрибутів.

• **Приклад**: Клас “Автомобіль” може мати властивості, як-от колір, марка, модель, а також методи, наприклад, “завести двигун” або “зупинитися”. Об’єктом цього класу може бути конкретний автомобіль з унікальними характеристиками.

2. **Інкапсуляція**

• Інкапсуляція означає приховування внутрішнього стану об’єкта та доступ до нього лише через методи класу. Це дозволяє захистити дані від зовнішнього втручання та забезпечити контрольований доступ.

• **Приклад**: Клас “Банк” має атрибут “баланс рахунку”, до якого можна отримати доступ лише через методи для внесення або зняття коштів, а не напряму.

3. **Наслідування**

• Наслідування дозволяє створювати нові класи на основі наявних, успадковуючи їхні властивості та методи. Це сприяє повторному використанню коду і створенню ієрархії класів.

• **Приклад**: Клас “Електромобіль” може наслідувати клас “Автомобіль” і додавати свої властивості, такі як ємність акумулятора, при цьому використовуючи базові методи класу “Автомобіль”.

4. **Поліморфізм**

• Поліморфізм дозволяє використовувати один і той самий метод для різних типів об'єктів. Завдяки цьому можна реалізовувати одну дію по-різному залежно від конкретного об'єкта.

• **Приклад**: Метод "рухатися" може бути визначений для класу "Транспортний засіб", а в класах "Автомобіль" і "Корабель" він може виконуватися по-різному, відповідно до їхньої специфіки.

5. **Абстракція**

• Абстракція дозволяє спрощувати складні системи, виділяючи лише основні деталі, що мають значення, і приховуючи складності реалізації.

• **Приклад**: Клас "Транспортний засіб" є абстракцією, яка включає спільні для всіх видів транспорту властивості та методи, а деталі для кожного окремого транспорту реалізуються в підкласах.

Етапи об'єктно-орієнтованого проектування

1. **Аналіз вимог**

• На цьому етапі визначаються вимоги до системи, зокрема функціональні та нефункціональні. Цей процес часто включає збір інформації від користувачів і визначення бізнес-процесів.

2. **Визначення об'єктів та класів**

• На основі вимог визначаються основні об'єкти, що будуть використані в системі. Об'єкти згруповуються в класи з визначенням їхніх атрибутів і методів.

3. **Побудова діаграм**

• Для моделювання взаємозв'язків між об'єктами створюються UML-діаграми, наприклад, діаграми класів, об'єктів, послідовностей, які показують взаємодію об'єктів і класів у системі.

4. **Визначення зв'язків між класами**

• Встановлюються зв'язки між класами, такі як асоціація, наслідування, агрегація та композиція, щоб відобразити, як об'єкти взаємодіють між собою.

5. **Реалізація методів**

• Розробляються методи для класів, які визначають поведінку об’єктів. Це включає написання коду для кожного методу, враховуючи специфіку класів і взаємодію між ними.

6. **Тестування та верифікація**

• Завершену систему перевіряють на відповідність початковим вимогам, функціональність та коректність роботи. Об’єкти і класи тестуються як окремо, так і в сукупності для забезпечення правильності роботи системи.

Переваги об’єктно-орієнтованого проектування

• **Модульність**: ООП дозволяє розділити програму на окремі модулі, які можна розробляти та тестувати незалежно один від одного.

• **Повторне використання коду**: Наслідування та абстракція дозволяють створювати ієрархію класів, що полегшує повторне використання вже створених компонентів.

• **Зручність підтримки**: Код, побудований на ООП, легше підтримувати і модифікувати, оскільки зміни в одному класі не впливають на інші класи.

• **Гнучкість**: Поліморфізм та інкапсуляція роблять систему гнучкою, забезпечуючи просте додавання нових функцій без зміни існуючого коду.

5. Засади структурно-функціональної методології

Структурно-функціональна методологія в інформаційному проєктуванні фокусується на організації, аналізі та створенні систем на основі взаємопов’язаних елементів і їх функцій. У контексті інформаційного проєктування цей підхід допомагає побудувати чітко структуровані, оптимальні й функціональні інформаційні системи, які задовольняють конкретні бізнесові чи технічні потреби. Ось ключові засади цієї методології, адаптовані до інформаційного проєктування:

Основні засади структурно-функціональної методології в інформаційному проєктуванні

1. **Принцип системної цілісності**

• Інформаційна система розглядається як єдине ціле, в якому всі компоненти (програмне забезпечення, апаратне забезпечення, бази даних, користувачі) працюють разом для досягнення загальної мети. Кожен елемент виконує свою функцію, але в межах єдиної інтегрованої системи.

• **Приклад**: В е-комерції система керування товарами, платіжна система та система обліку клієнтів мають взаємодіяти для забезпечення безперебійної роботи онлайн-магазину.

2. **Принцип ієрархічної структури**

• Компоненти інформаційної системи організовуються на різних рівнях ієрархії, що дозволяє виділити підсистеми і компоненти, з якими можна працювати окремо. Це дає можливість структурувати систему, розділивши її на рівні для зручного управління та розробки.

• **Приклад**: У банківській системі можна виділити рівні обробки транзакцій, клієнтських даних, аналітики, звітності тощо.

3. **Принцип функціональної спрямованості**

• Кожен елемент інформаційної системи виконує певну функцію, яка має значення для досягнення загальної мети системи. Розподіл функцій допомагає визначити роль кожного компонента та забезпечити оптимальне виконання його завдань.

• **Приклад**: База даних виконує функцію зберігання даних, тоді як модуль обробки запитів виконує функцію обробки та відповіді на запити користувачів.

4. **Принцип взаємозалежності компонентів**

• Усі компоненти системи є взаємопов'язаними, а зміна одного компонента може впливати на інші. Це дозволяє створити систему, в якій взаємодія елементів є узгодженою, а процеси – інтегрованими.

• **Приклад**: Зміна в структурі бази даних (наприклад, додавання нових полів) може потребувати оновлення інших компонентів системи, зокрема програмного забезпечення для обробки даних.

5. **Принцип адаптивності та гнучкості**

• Інформаційна система повинна мати можливість адаптуватися до змін зовнішніх та внутрішніх умов. Це забезпечує можливість оновлення, додавання нових функцій та масштабування системи.

• **Приклад**: Впровадження нового типу платежів у фінансовій системі потребує адаптації системи без необхідності її повної перебудови.

6. **Принцип стабільності**

• Інформаційна система повинна залишатися стабільною і забезпечувати постійну роботу навіть під час внесення змін або підвищеного навантаження. Стабільність забезпечує надійність системи для користувачів.

• **Приклад**: У системі, що обробляє велику кількість запитів (наприклад, соціальні мережі), стабільність забезпечується через резервування даних та застосування методів управління навантаженням.

7. **Принцип узгодженості функцій та структури**

• Функціональні можливості кожного компонента мають відповідати його структурі та можливостям. Цей принцип гарантує, що функції, які виконує система, оптимально відповідають її структурі та використовують доступні ресурси.

• **Приклад**: У системі керування проєктами структура має відповідати функціональним потребам, забезпечуючи можливість співпраці, розподілу завдань, відстеження прогресу.

8. **Принцип модульності**

• Компоненти інформаційної системи повинні мати чітку структуру і бути розділені на модулі, які можна оновлювати або замінювати незалежно один від одного. Модульність дозволяє легко вносити зміни в окремі частини системи.

• **Приклад**: В інформаційній системі CRM (управління взаємодією з клієнтами) модулі комунікації, управління продажами та аналітики можуть працювати як окремі компоненти.

9. **Принцип оптимізації**

• Система повинна бути оптимізована для ефективного використання ресурсів і виконання завдань. Це досягається завдяки ретельному проектуванню структури і функцій.

• **Приклад**: Оптимізація запитів до бази даних або мінімізація обсягів передаваних даних у мережі знижує затрати ресурсів та підвищує швидкість обробки інформації.

Використання структурно-функціональної методології в інформаційному проєктуванні дозволяє створювати інформаційні системи, які є не тільки зручними та надійними, але й стійкими до змін. Ці засади є важливими для забезпечення масштабованості та гнучкості, необхідних для динамічного середовища сучасних технологій, де інформаційні системи мають здатність адаптуватися до постійних змін і зберігати ефективність.

6. Концепція розробки інформаційної системи підприємства на основі структурного підходу

Концепція розробки інформаційної системи підприємства на основі структурного підходу в інформаційному проєктуванні базується на принципах структурного аналізу та структурного моделювання, що дозволяє систематично підходити до створення комплексної, масштабованої, ефективної та надійної системи управління інформаційними процесами. Мета такого підходу — забезпечити точне виконання бізнес-вимог підприємства, підвищення ефективності його роботи, а також зручне впровадження та обслуговування системи.

Основні етапи розробки інформаційної системи підприємства за структурним підходом

1. **Аналіз вимог та специфікація системи**

• На першому етапі проводиться детальний збір вимог від користувачів і зацікавлених сторін, аналізуються бізнес-процеси підприємства, визначаються цілі системи та функціональні і нефункціональні вимоги. Важливо також визначити обмеження проєкту (часові, бюджетні, технічні).

• **Результат**: документ специфікації вимог до системи, що визначає, які саме функції повинна виконувати інформаційна система.

2. **Декомпозиція та структурний аналіз**

• На основі вимог система поділяється на компоненти або модулі, кожен з яких виконує окрему функцію. Декомпозиція дозволяє виділити базові елементи системи та їх взаємозв’язки, що забезпечує чітку структуру системи.

• **Результат**: створення діаграм, таких як діаграма потоків даних (DFD) і діаграма компонентів, що показують зв’язки та взаємодію між компонентами.

3. **Проектування структури системи**

• Створення модульної структури системи, де кожен модуль відповідає за певний набір функцій. При цьому кожен модуль повинен мати зрозумілі інтерфейси для взаємодії з іншими модулями.

• **Результат**: архітектурна схема системи з визначенням ієрархічної структури компонентів.

4. **Проектування бази даних**

• На цьому етапі розробляється структура бази даних, яка має відповідати потребам системи. Структурний підхід передбачає нормалізацію даних і побудову зв’язків між таблицями для забезпечення цілісності, оптимізації пошуку й обробки інформації.

• **Результат**: створення концептуальної моделі даних, яка визначає зв’язки між таблицями і атрибутами, а також логічної моделі бази даних.

5. **Проектування взаємодії користувача з системою**

• Розробка інтерфейсу користувача з врахуванням зручності, інтуїтивності і простоти використання. Інтерфейс користувача повинен відображати структуру системи, забезпечувати легкий доступ до всіх функцій.

• **Результат**: прототипи інтерфейсу, що відповідають специфікації вимог і дозволяють користувачам взаємодіяти з основними функціями системи.

6. **Впровадження та тестування компонентів**

• Розробка модулів системи, їх інтеграція і тестування для перевірки функціональності. Структурний підхід дозволяє тестувати кожен компонент окремо, а потім проводити інтеграційне тестування для забезпечення сумісності.

• **Результат**: тестована система, готова до впровадження, в якій перевірено функціональні й нефункціональні вимоги.

7. **Впровадження системи та навчання користувачів**

• Після завершення тестування інформаційна система впроваджується на підприємстві. Цей процес включає також навчання користувачів і надання документації, що полегшує подальшу експлуатацію системи.

• **Результат**: система, впроваджена на підприємстві, та підготовлені користувачі, які можуть ефективно взаємодіяти з новою системою.

8. **Підтримка та оновлення**

• Система потребує постійної підтримки та оновлення відповідно до змін бізнес-процесів підприємства або нових вимог. Структурна модель спрощує процес оновлення, оскільки зміни можна внести в окремі модулі без суттєвого порушення роботи системи.

• **Результат**: стабільна робота системи та відповідність новим вимогам підприємства.

Принципи структурного підходу в інформаційному проєктуванні

1. **Чітке розподілення функцій** – Кожен модуль виконує окрему функцію, що полегшує управління та підтримку системи.

2. **Ієрархічність структури** – Система організована за ієрархічною схемою, що дозволяє легко масштабувати систему.
3. **Модульність** – Вся система поділена на модулі, які можна розробляти, тестувати й оновлювати незалежно.
4. **Інтерфейси для взаємодії** – Між компонентами системи визначені інтерфейси, що стандартизують обмін даними.
5. **Гнучкість і адаптивність** – Завдяки структурованій модульності, систему легко адаптувати до нових вимог.

Переваги структурного підходу

- **Підвищена продуктивність**: завдяки чіткій структурі зменшуються помилки і пришвидшується розробка та впровадження.
- **Зручність в управлінні**: ієрархічна структура дозволяє легко керувати кожним компонентом.
- **Масштабованість**: завдяки модульній структурі система може розширюватись з урахуванням потреб підприємства.
- **Зниження витрат на підтримку**: зміни або оновлення можна впроваджувати без великих витрат часу на зміну всієї системи.

Загалом, структурний підхід в інформаційному проєктуванні дає змогу створювати системи, що задовольняють потреби підприємства, легко адаптуються до змін і забезпечують ефективну підтримку для користувачів.

7. Інструментальні засоби структурного проектування

Інструментальні засоби структурного проєктування — це спеціальні програми та середовища, які допомагають аналітикам і розробникам створювати, аналізувати та підтримувати структуровані проєкти систем і програмного забезпечення. Вони включають різні засоби для моделювання,

тестування та документування системи на основі структурного підходу, що допомагає забезпечити чітку організацію, модульність і взаємодію компонентів у межах складних інформаційних систем.

Основні інструментальні засоби структурного проєктування

1. **CASE-засоби (Computer-Aided Software Engineering)**

• CASE-засоби є комплексними інструментами для автоматизації процесів розробки ПЗ, включаючи структурний аналіз і проєктування. Вони допомагають проєктувати, аналізувати, тестувати та документувати структуру системи.

- **Приклади**: Rational Rose, Oracle Designer, PowerDesigner.
- **Функції**:
  - Моделювання та створення діаграм (DFD, ERD, діаграми діяльності).
  - Генерація коду з моделі.
  - Автоматичне документування.

2. **Інструменти для створення діаграм потоків даних (DFD)**

• Діаграми потоків даних дозволяють моделювати, як дані проходять через систему, включаючи процеси, що трансформують дані, зберігання та інтерфейси з іншими системами. DFD дозволяють чітко визначити вхідні й вихідні дані для кожного модуля.

- **Приклади**: Lucidchart, Microsoft Visio, SmartDraw.
- **Функції**:
  - Побудова ієрархічної структури процесів.
  - Візуалізація потоків даних і зв’язків.

3. **Інструменти моделювання ERD (Entity-Relationship Diagrams)**

• ERD використовується для створення концептуальних та логічних моделей даних, відображення зв’язків між сутностями та атрибутами, що є основою структури бази даних.

- **Приклади**: MySQL Workbench, ER/Studio, dbForge Studio.

- **Функції**:
  - Моделювання сутностей, зв’язків і атрибутів.
  - Створення схеми бази даних і зв’язків.

4. **Уніфіковані мови моделювання (UML)**

• UML забезпечує універсальний стандарт для створення структурних і поведінкових моделей. UML-діаграми допомагають визначити основні об’єкти, їхні взаємодії, а також компоненти та модулі, які формують структуру системи.

- **Приклади**: Enterprise Architect, Visual Paradigm, StarUML.
- **Функції**:
  - Створення різних типів діаграм (класів, компонентів, взаємодії, послідовності).
  - Підтримка візуалізації структури системи на різних рівнях абстракції.

5. **Інструменти для проєктування архітектури системи**

• Програми для архітектурного проєктування допомагають визначити загальну структуру і взаємозв’язки компонентів системи. Вони полегшують процес планування і проєктування системної архітектури.

- **Приклади**: ArchiMate, C4 Model, System Architect.
- **Функції**:
  - Побудова високорівневих архітектурних моделей.
  - Ієрархічне структурування компонентів системи.
  - Відображення взаємодій і зв’язків між модулями.

6. **Системи контролю версій та управління конфігураціями**

• Інструменти контролю версій зберігають всі зміни в структурі системи, що особливо важливо для великих проєктів, де над різними модулями працює кілька команд.

- **Приклади**: Git, SVN, Mercurial.
- **Функції**:
  - Відстеження і керування змінами в проєкті.
  - Синхронізація роботи кількох команд.

- Історія версій і можливість повернення до попередніх змін.

7. **Інструменти для автоматизованого тестування**

- Автоматизоване тестування є критичним етапом у структурному проєктуванні, оскільки дозволяє перевіряти правильність роботи окремих модулів і взаємодію між ними.
- **Приклади**: Selenium, JUnit, TestComplete.
- **Функції**:
  - Автоматизація тестування функціональності, продуктивності, інтеграції.
  - Виявлення дефектів на етапі розробки.
  - Підтримка тестування компонентів, відповідно до структурного дизайну.

8. **Інструменти для документування**

- Інструменти для документування забезпечують повне документування всіх етапів проєктування, що важливо для підтримки й оновлення системи.
- **Приклади**: Confluence, Doxygen, Microsoft Word/Excel.
- **Функції**:
  - Створення і збереження технічної документації, діаграм, специфікацій.
  - Ведення журналів змін і поліпшень.
  - Забезпечення доступу до документів для розробників та інших зацікавлених сторін.

Переваги використання інструментальних засобів структурного проєктування

- **Підвищення продуктивності**: інструменти дозволяють зекономити час на рутинних задачах завдяки автоматизації проєктування, створення діаграм та документування.

• **Якість та надійність**: структуровані підходи допомагають краще контролювати процес розробки, що знижує ризик помилок.

• **Масштабованість і гнучкість**: завдяки поділу на модулі та структурування, систему можна легко адаптувати й масштабувати.

• **Полегшення підтримки**: детальне документування всіх компонентів і зв’язків системи полегшує підтримку й оновлення.

Інструментальні засоби структурного проєктування є важливими компонентами для розробників і архітекторів, оскільки вони підтримують ефективність, надійність і якість інформаційних систем.

8. Универсальна мова моделювання (UML)

Unified Modeling Language (UML, уніфікована мова моделювання) — це стандартна мова моделювання для опису, проектування та документування програмних систем. UML широко використовується в розробці об’єктно-орієнтованих систем для створення графічних діаграм, що відображають різні аспекти системи, включаючи її структуру, поведінку і взаємодію компонентів.

Це не мова програмування, скоріше набір правил та стандартів для створення діаграм. Вони дозволяють розробникам програмного забезпечення та інженерам «говорити однією мовою», не заглиблюючись у фактичний код свого продукту. Складання діаграм за допомогою UML — це чудовий спосіб допомогти іншим швидко зрозуміти складну ідею чи структуру.

Основні особливості UML

1. **Універсальність**: UML можна використовувати для різних типів систем, включаючи програмне забезпечення, бізнес-процеси, складні технічні системи та навіть організаційні структури.

2. **Візуальність**: UML забезпечує наочний спосіб подання складних концепцій, полегшуючи спілкування між учасниками проєкту, включаючи програмістів, аналітиків, архітекторів та інших зацікавлених осіб.

3. **Стандартизація**: UML є стандартом, підтримуваним Object Management Group (OMG), що гарантує сумісність і узгодженість між різними інструментами і підходами.

Класифікація діаграм UML

UML-діаграми поділяються на 14 типів:

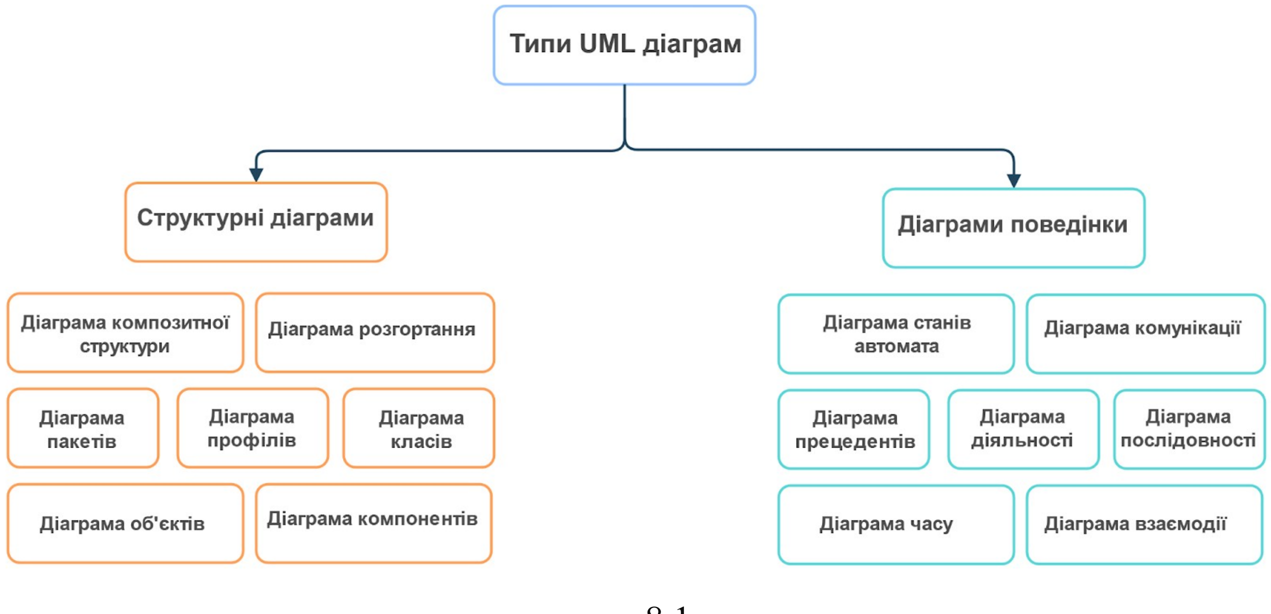


рис 8.1

UML складається з різних типів діаграм, які можна поділити на дві основні групи: **структурні діаграми** та **поведінкові діаграми**.

1. Структурні діаграми

Ці діаграми використовуються для опису статичної структури системи, тобто компонентів системи та зв’язків між ними.

• **Діаграма класів**: Відображає класи в системі, їхні атрибути, методи та зв’язки між класами (наслідування, асоціації, залежності).

• **Діаграма об’єктів**: Зображає окремі об’єкти з їх станами на конкретний момент часу. Використовується для аналізу стану системи на певному етапі.

• **Діаграма компонентів**: Показує, як фізичні компоненти (наприклад, бібліотеки, модулі, файли) пов'язані один з одним і як вони формують систему.
• **Діаграма розгортання**: Відображає фізичне розташування компонентів у системі, зокрема розподілені сервери, вузли, мережеві пристрої.
• **Діаграма пакетів**: Використовується для організації елементів у логічні пакети або модулі, показуючи структуру та залежності між різними пакетами.

2. Поведінкові діаграми

Ці діаграми описують динамічні аспекти системи, такі як процеси, взаємодії між компонентами та потоки даних.
• **Діаграма прецедентів (Use Case Diagram)**: Показує функціональність системи з точки зору кінцевого користувача, включаючи зв'язки між користувачами (акторами) та їхні дії.
• **Діаграма діяльності (Activity Diagram)**: Моделює бізнес-процеси або логіку дій у вигляді потоків діяльності, включаючи розгалуження, цикли, синхронізацію.
• **Діаграма послідовності (Sequence Diagram)**: Відображає взаємодію між об'єктами в системі, показуючи порядок виклику методів і обмін повідомленнями.
• **Діаграма комунікацій (Communication Diagram)**: Подібна до діаграми послідовності, але більше фокусується на зв'язках між об'єктами.
• **Діаграма станів (State Machine Diagram)**: Відображає стани об'єкта в системі і події, що викликають переходи між цими станами.
• **Діаграма часових діаграм (Timing Diagram)**: Спеціалізована діаграма, яка показує зміни станів об'єктів у часі. Використовується для аналізу систем реального часу або для систем, де важлива часова послідовність подій.

Основні елементи UML

1. **Клас**: Представляє концепцію або об'єкт у системі, має атрибути (властивості) та методи (функції).
2. **Об'єкт**: Конкретний екземпляр класу, який має певний стан.
3. **Актор**: Зовнішній користувач або система, що взаємодіє із системою.
4. **Прецедент**: Визначає, яку функціональність система повинна виконувати для користувачів (акторів).
5. **Повідомлення**: Описує обмін інформацією або виклик функцій між об'єктами.
6. **Пакет**: Логічне угрупування елементів моделі для полегшення організації та навігації.
7. **Стан**: Відображає конкретний стан об'єкта або системи в певний момент часу.
8. **Перехід**: Вказує на зміну стану, що відбувається через подію.

Переваги UML

- **Підтримка об'єктно-орієнтованого підходу**: UML розроблена для об'єктно-орієнтованих систем, що робить її зручною для опису об'єктів і їхньої взаємодії.
- **Гнучкість і універсальність**: UML можна використовувати для моделювання різних видів систем, не тільки програмного забезпечення, але й бізнес-процесів.
- **Стандартизація**: UML є міжнародним стандартом, що полегшує комунікацію між командами розробників, бізнес-аналітиками і замовниками.
- **Зручність у спільній роботі**: Графічні моделі є простими для розуміння і спільного використання, що робить UML зручним для командного проєктування.

Приклади використання UML

- **Аналіз вимог**: Використання діаграми прецедентів для моделювання потреб користувачів і виявлення основних функцій системи.
- **Проєктування архітектури**: Використання діаграм класів і компонентів для побудови архітектури системи.

• **Документування бізнес-процесів**: Діаграми діяльності дозволяють моделювати логіку бізнес-процесів або операцій.

• **Опис сценаріїв використання**: Діаграми послідовності і комунікацій моделюють конкретні сценарії взаємодії компонентів або об’єктів.

• **Моделювання поведінки**: Діаграми станів і часові діаграми можуть використовуватися для проєктування систем реального часу або для опису поведінки об’єктів у динамічних умовах.

Інструменти для роботи з UML

Існує багато інструментів для створення UML-діаграм, серед яких:

- **Enterprise Architect**
- **Visual Paradigm**
- **StarUML**
- **Lucidchart**
- **Microsoft Visio**

Отже UML — це потужний і універсальний інструмент для створення, аналізу та візуалізації програмних систем і бізнес-процесів, який дає змогу будувати системи з чітко структурованими взаємозв’язками та забезпечувати високий рівень прозорості на всіх етапах розробки.

Розглянемо, як будувати найуживаніші типи діаграм.

8.1. Діаграма класів

Діаграма класів — це один із найпопулярніших типів діаграм UML, який використовується для моделювання структури системи, відображаючи класи, їхні атрибути, методи та зв’язки між ними. Діаграми класів допомагають розробникам, архітекторам і аналітикам зрозуміти і спроєктувати систему, визначивши основні об’єкти та їх взаємозв’язки.

Клас (class) - абстрактний опис множини однорідних об􀀀єктів, що мають однакові атрибути, операції й відносини з об􀀀єктами інших класів. Графічно клас зображується у вигляді прямокутника, що додатково може бути розділений

горизонтальними лініями на розділи або секції (рис. 8.2.). У цих секціях можуть вказуватися ім’я класу, атрибути й операції класу.

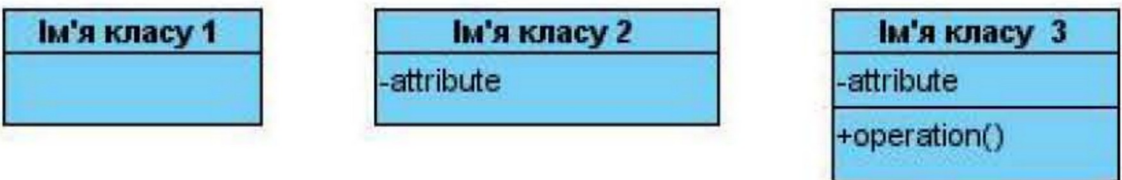


Рис. 8.2.

Навіть якщо секції атрибутів й операцій порожні, у позначенні класу вони повинні бути виділені горизонтальною лінією, для того, щоб відрізнити клас від інших елементів мови UML. Приклади графічного зображення конкретних класів наведені на рис. 8.2. У першому випадку для класу Коло (Circle) зазначені тільки його атрибути - точка на координатній площині, що визначає розташування його центра й радіус. Для класу Вікно (Window) зазначені тільки його операції.

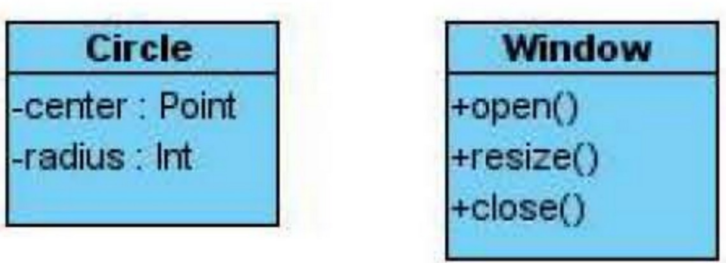


Рис. 8.3.

Клас може мати або не мати екземплярів або об’єктів. Залежно від цього в мові UML розрізняють конкретні й абстрактні класи.

Конкретний клас (concrete class) - клас, на основі якого можуть бути безпосередньо створені екземпляри або об’єкти.

Абстрактний клас (abstract class) - клас, що не має екземплярів або об’єктів.

Розглянуті вище позначення відносяться до конкретних класів.

Ім’я класу

Ім’я класу має бути унікальним у межах пакета, що може містити одну або кілька діаграм класів. Ім’я вказується в самій верхній секції прямокутника, тому

вона часто називається секцією імені класу. На додаток до загального правила іменування елементів мови UML, ім'я класу записується у центрі секції імені напівжирним шрифтом і повинне починатися із великої літери.

Для позначення імені абстрактного класу використається похилий шрифт (курсив). У мові UML прийнята загальна угода про те , що будь-який текст, що відноситься до абстрактного елемента, записується курсивом. Це має принципове значення, оскільки є семантичним аспектом опису абстрактних елементів мови UML.

У деяких випадках необхідно явно вказати, до якого пакета відноситься той або інший клас. Для цього використається спеціальний символ роздільник — подвійна двокрапка (::). Синтаксис рядка імені класу в цьому випадку буде наступний: «Ім'я пакету>::<Ім'я класу >. Інакше кажучи, перед ім'ям класу повинне бути явно зазначене ім'я пакета, до якого його варто віднести. Наприклад, якщо визначено пакет з ім'ям Банк, то клас Рахунок у цьому банку може бути записаний у вигляді: Банк :: Рахунок.

Атрибути класу

Атрибут (attribute) - змістовна характеристика класу, що описує множину значень, яких можуть набувати окремі об'єкти цього класу.

Атрибут класу призначено для опису окремої властивості або ознаки, що є загальною для всіх об'єктів даного класу. Атрибути класу записуються в другій зверху секції прямокутника класу. Цю секцію часто називають секцією атрибутів.

Ім'я атрибута являє собою рядок тексту, що використається як ідентифікатор відповідного атрибута й тому повинне бути унікальним в межах даного класу. Ім'я атрибута - єдиний обов'язковий елемент синтаксичного позначення атрибута, повинне починатися з маленької літери й не повинне містити пробілів. Знак "/" перед ім'ям атрибута вказує на те, що цей атрибут є похідним від деякого іншого атрибута цього ж класу.

Похідний атрибут (derived element) - атрибут класу, значення якого для окремих об'єктів може бути обчислене за допомогою значень інших атрибутів цього ж об'єкта .

У мові UML прийнята певна стандартизація запису атрибутів класу. Кожному атрибуту класу відповідає окремий рядок тексту, що складається із квантора видимості атрибута, імені атрибута, його кратності, типу значень атрибута й, можливо, його вихідного значення. Загальний формат запису окремого атрибута класу наступний:

```
«квантор видимості > ‹ім’я атрибута> [кратність) :
‹тип атрибута > = ‹ вихідне значення > (рядок-властивість ) .
```

Видимість (visibility) - якісна характеристика опису елементів класу, що характеризує потенційну можливість інших об`єктів моделі впливати на окремі аспекти поведінки.

Видимість у мові UML специфікується за допомогою квантора видимості (visibility), що може набувати одного з 4- х можливих значень і відображатися за допомогою спеціальних символів.

• Символ "+" - позначає атрибут з областю видимості типу загальнодоступний (public). Атрибут із цією областю видимості доступний або видимий з будь-якого іншого класу пакета, у якому визначена діаграма.

• Символ "#" - позначає атрибут з областю видимості типу захищений (protected). Атрибут із цією областю видимості недоступний або невидимий для всіх класів , за винятком підкласів даного класу.

• Символ "-" - позначає атрибут з областю видимості типу закритий (private). Атрибут із цією областю видимості недоступний або невидимий для всіх класів без винятку.

• Символ "~" - позначає атрибут з областю видимості типу пакетний (package). Атрибут із цією областю видимості недоступний або невидимий для всіх класів за межами пакета, у якому визначений клас-власник даного атрибута.

Class
-Age : int
+Company : String
#Name : String
-OtherDetails : String
+getAge() : int
~setName() : String
+updateOther() : Amount,CustomerID

Рис. 8.4.

Вважається, що якісна об’єктно-орієнтована архітектура обмежує безпосередній доступ до атрибутів класу та пропонує методи, які дозволяють оперувати атрибутами за викликом. Такий підхід гарантує, що зміни даних відбуваються в одній точці та за певними правилами.

Кратність (multiplicity) атрибута характеризує загальну кількість атрибутів даного типу, що входять до складу окремого класу. У загальному випадку кратність записується у формі рядка тексту із цифр у квадратних дужках після імені відповідного атрибута, при цьому цифри розділяються двома крапками: [нижня границя..верхня границя], де нижня й верхня границі позитивні цілі числа. Кожна така пара призначена для позначення окремого замкнутого інтервалу цілих чисел. У якості верхньої границі може використовуватися спеціальний символ "*" (зірочка), що означає довільне позитивне ціле число, тобто необмежене зверху значення кратності відповідного атрибута. Інтервалів кратності для окремого атрибута може бути кілька. У цьому випадку їхне спільне використання відповідає теоретико-множинному об’єднанню відповідних інтервалів. Значення кратності з інтервалу монотонно зростають без пропуску окремих чисел, що лежать між нижньою та верхньою границями. При цьому дотримуються наступного правила: відповідні нижні й верхні границі інтервалів включаються в значення кратності. Якщо як кратність указується одне значення, то кратність атрибута приймається рівною даному числу. Якщо ж указується єдиний знак "*", то це означає, що кратність атрибута може бути довільним позитивним цілим числом або нулем. У мові UML кратність широко використається також для завдання

ролей асоціацій, складених об’єктів і значень атрибутів. Якщо кратність атрибута не зазначена, то за замовчуванням у мові UML приймається її значення рівне [1..1].

Операції класу

Операція (operation) - це сервіс, що надається кожним екземпляром або об’єктом класу на вимогу своїх клієнтів, якими можуть виступати інші об’єкти, у тому числі й екземпляри даного класу.

Операції класу записуються в третій зверху секції прямокутника класу, яку часто називають секцією операцій. Сукупність операцій характеризує функціональний аспект поведінки всіх об’єктів даного класу. Кожній операції класу відповідає окремий рядок, що складається із квантора видимості операції, імені операції, виразу типу, що повертає операція, значення й, можливо, рядок-властивість даної операції. Запис окремої операції класу має наступний вигляд:

```
<квантор видимості> <ім’я операції> (список параметрів): <вираз
типу значення, що повертає> {рядок-властивість}
```

Ім’я операції - рядок тексту, що використовується як ідентифікатор відповідної операції. Воно повинно бути унікальним в межах даного класу. Ім’я операції — єдиний обов’язковий елемент синтаксичного позначення операції, повинне починатися з маленької літери і, як правило, записуватися без пробілів.

Список параметрів - перелік розділених комою формальних параметрів.

Параметр (parameter) - специфікація змінної операції, що може бути змінена, передана або повернута.

Параметр може включати ім’я, тип, напрямок і значення за замовчуванням.

Ім’я параметра - ідентифікатор відповідного формального параметра, при записі якого додержуються правил завдання імен атрибутів.

Вираз типу - специфікація типу даних для припустимих значень відповідного формального параметра.

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

Значення за замовчуванням у загальному випадку - деяке конкретне значення для цього формального параметра.

Вираз типу значення, що повертається, також вказує на тип даних значення, що повертається об'єктом після виконання відповідної операції. Дві крапки та вираз типу значення, що повертається, можуть бути опущені, якщо операція не повертає ніякого значення.

Розширення діаграми класів

Однією з характерних особливостей мови UML є наявність механізмів розширення, які дозволяють додати до розгляду додаткові графічні позначення, орієнтовані на розв'язування завдань із певної предметної області. Одним з таких розширень є профіль для процесу розробки програмного забезпечення (The UML Profile for Software Development Processes).

У рамках вищезазначеного профілю запропоновано три спеціальних графічних примітиви, які можуть бути використані для уточнення семантики окремих класів при побудові різних діаграм:

• Граничний клас (boundary class) — клас, що розташовується на границі системи із зовнішнім середовищем і безпосередньо взаємодіє з акторами, але є складовою частиною системи. Граничний клас може бути зображений у формі прямокутника класу зі стереотипом <<boundary>> (рис. 8.5.).

• Клас-сутність (entity class) - пасивний клас, що містить інформацію, яка має зберігатися постійно й не повинна знищуватися зі знищенням об'єктів даного класу або припиненням роботи системи, що моделюється. Зазвичай, цей клас відповідає окремій таблиці бази даних. У цьому випадку його атрибути є полями таблиці, а операції - приєднаними або збереженими процедурами. Цей клас лише приймає повідомлення від інших класів моделі. Клас-сутність може бути зображений формі прямокутника класу зі стереотипом <<entity>> (рис. 8.5.).

• Керуючий клас (control class) - клас, відповідальний за координацію дій інших класів. На кожній діаграмі класів повинен бути хоча б один керуючий клас, причому кількість повідомлень, що посилають об'єктам керуючого класу - мала, у порівнянні із кількістю, що розсилають вони. Керуючий клас відповідає

за координацію дій інших класів. Зазвичай, даний клас є активним й ініціює розсилання множини повідомлень іншим класам моделі. Крім спеціального позначення керуючий клас може бути зображений у формі прямокутника класу зі стереотипом <<control>> (рис. 8.5.).

Рис. 8.5.

Інтерфейс

Інтерфейс (interface) - іменована множина операцій, які характеризують поведінку окремого елемента моделі.

Інтерфейс у контексті мови UML є спеціальним випадком класу, у якого є операції, але відсутні атрибути. Для позначення інтерфейсу використовується стандартний спосіб - прямокутник класу зі стереотипом <<interface>> (рис 8.6.). Як ім'я може бути використаний іменник, що характеризує відповідну інформацію або сервіс, наприклад, "Датчик температури", "Форма вводу", "Сирена", "Відеокамера" (рис. 8.6.). З урахуванням мови реалізації моделі ім'я інтерфейсу, як й імена інших класів, рекомендується записувати англійською й починати із великої літери I, наприклад, ITemperatureSensor, IsecureInformation.

Рис. 8.6.

Відношення на діаграмі класів

Крім внутрішніх принципів організації класів, важливу роль при розробці проектованої системи мають різні відношення між класами, які також можуть бути зображені на діаграмі класів.

Базові відношення, що можуть бути зображені на діаграмі класів :

• Відношення асоціації (association relationship)

• Відношення узагальнення (generalization relationship)

• Відношення агрегації (aggregation relationship)

• Відношення композиції (composition relationship)

Кожне з цих відношень має власне графічне зображення, що відображує семантичний характер взаємозв’язку між об’єктами відповідних класів.

Відношення асоціації

Відношення асоціації (association) відповідає наявності довільного відношення або взаємозв’язку між класами. Це відношення позначається суцільною лінією зі стрілкою або без неї й з додатковими символами, які характеризують спеціальні властивості асоціації.

Бінарна асоціація (binary association) - найбільш простий вид відношення асоціації, служить для зображення довільного відношення між двома класами. Вона зв’язує в точності два різних класи й може бути ненаправленим (симетричним) або направленим відношенням. Окремий випадок бінарної асоціації — рефлексивна асоціація, що зв’язує клас із самим собою. Ненаправлена бінарна асоціація зображується лінією без стрілки. Для неї на діаграмі може бути зазначений порядок читання класів з використанням значка у формі трикутника поруч із ім’ям даної асоціації.

Як простий приклад ненаправленої бінарної асоціації можна розглянути відношення між двома класами - класом Компанія й класом Співробітник (рис. 8.7.). Вони зв’язані між собою бінарною асоціацією “працює”, ім’я якої зазначено на рисунку поруч із лінією асоціації, тобто співробітник працює в компанії.

Рис. 8.7.

Ім'я асоціації - необов'язковий елемент її позначення. Але, якщо воно задано, то має бути записано поруч із лінією асоціації. При генерації коду з діаграми класів, іменований кінець асоціації стає змінною екземпляру цільового класу.

Спрямована бінарна асоціація зображується суцільною лінією із простою стрілкою на одній з ї кінцевих точок. Напрямок цієї стрілки вказує на те, який клас є першим, а який - другим.

Як простий приклад спрямованої бінарної асоціації можна розглянути відношення між двома класами - класом Клієнт і класом Рахунок (рис. 8.8.). Вони зв'язані між собою бінарною асоціацією з ім'ям Має , для якої визначений порядок проходження класів. Це означає, що конкретний об'єкт класу Клієнт завжди повинен указуватися першим при розгляді взаємозв'язку з об'єктом класу Рахунок. Інакше кажучи, ці об'єкти класів утворять кортеж елементів, наприклад, ‹клієнт, рахунок 1, рахунок 2,., рахунок п›.

Рис. 8.8.

Наступний елемент позначень - кратність асоціації. Кратність ставиться до кінців асоціації і позначається у вигляді інтервалу цілих чисел, аналогічно до кратності атрибутів й операцій класів, але без прямих дужок. Цей інтервал записується поруч із кінцем відповідної асоціації і означає потенційне число окремих екземплярів класу, які можуть мати місце, коли інші екземпляри або об'єкти класів фіксовані.

Наприклад, кратність "1" для класу Компанія означає, що кожен співробітник може працювати тільки в одній компанії. Кратність "1..*" для класу Співробітник означає, що в кожній компанії можуть працювати кілька співробітників, загальне число яких заздалегідь невідомо й нічим не обмежено. (рис. 8.7.) Замість кратності "1.*" не можна записати тільки символ "*", оскільки останній означає кратність "0.*". Для даного прикладу це означало б, що окремі компанії можуть зовсім не мати співробітників у своєму штаті .

Для відношення залежності призначені ключові слова, що позначають деякі спеціальні види залежностей. Ці ключові слова (стереотипи) записуються в лапках поряд із стрілкою, яка відповідає даній залежності. Приклади представлені нижче:

• "access" - служить 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-клієнта можуть бути обчислені за атрибутами класу-джерела;

• "import" - відкриті атрибути і операції класу-джерела стають частиною класу-клієнта, ніби вони були оголошені безпосередньо в ньому;

• "refine" - указує, що клас-клієнт служить уточненням класу-джерела через причини історичного характеру, коли з'являється додаткова інформація в ході роботи над проєктом.

Відношення узагальнення

Узагальнення (generalization) - відношення між більше загальним поняттям і менш загальним поняттям.

Менш загальний елемент моделі повинен бути погоджений з більше 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.

Стосовно до діаграми класів дане відношення описує ієрархічну будову класів і наслідування їхніх властивостей і поведінки.

Наслідування (inheritance) - спеціальний концептуальний механізм, за допомогою якого спеціальні елементи містять у собі структуру й поведінку загальних елементів, тобто клас-нащадок має всі властивості й поведінку класу-предка, а також має власні властивості й поведінку, які можуть бути відсутні у класі-предку.

Батько, предок (parent) - відносно узагальнення більш загальний елемент.

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

На діаграмах відношення узагальнення позначається суцільною лінією із трикутною стрілкою на одному з кінців (рис. 8.9.). Стрілка вказує на загальний клас (клас-предок або суперклас), а її початок - на спеціальний клас (клас-нащадок або підклас).

Рис. 8.9.

Від одного класу-предка одночасно можуть успадковувати декілька класів-нащадків.

Наступна діаграма ілюструє відношення узагальнення між батьківським класом і нащадком (рис. 8.10.). Об’єкт Коло матиме атрибути координати: x_position, y position та радіус (radius), а також метод display(). Зауважимо, що "Shape" абстрактний клас.

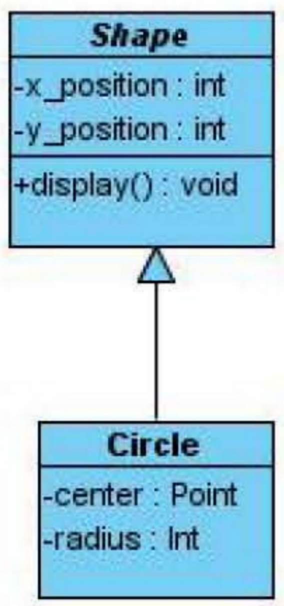


Рис. 8.10.

На додаток до простої стрілки узагальнення може бути приєднаний рядок тексту, що вказує на спеціальні властивості цього відношення у формі обмеження, що повинне бути записане у фігурних дужках

• {complete} - означає, що в даному відношенні узагальнення специфіковані всі класи-нащадки, і інших класів-нащадків у даного класу-предка бути не може.

• {incomplete} - означає випадок, протилежний першому. Тобто передбачається, що на діаграмі зазначені не всі класи-нащадки. Можливо, надалі розробник заповнить їхній перелік, не змінюючи вже побудовану діаграму.

• {disjoint} - означає, що класи-нащадки не можуть містити об□єктів, що одночасно є екземплярами двох або більше класів.

• {overlapping} - випадок, протилежний попередньому. Передбачається, що окремі екземпляри класів-нащадків можуть належати одночасно декільком класам.

Відношення агрегації

Агрегація (aggregation) - спеціальна форма асоціації, що служить для відображення відношення типу "частина-ціле" між агрегатом (ціле) і його складовою частиною .

Відношення агрегації має місце між декількома класами в тому випадку, якщо один із класів являє собою сутність, що містить у собі у якості складових частин інші сутності.

Розподіл системи на складові частини може бути розглянуто як ієрархія. Проте ієрархія такого виду принципово відрізняється від тієї, що породжується відношенням узагальнення. Відмінність полягає в тім, що частини системи не мають успадковувати її властивості й поведінку, оскільки є самостійними сутностями. Більше того, частини цілого мають власні атрибути й операції, які можуть істотно відрізняються від атрибутів й операцій цілого.

Графічно відношення агрегації зображується суцільною лінією з не зафарбованим усередині ромбом на одному з своїх кінців. Цей ромб указує на той клас, що являє собою "ціле" або клас-контейнер. Інші класи є його "частинами" (рис. 8.11.).

Рис. 8.11.

Як приклад відношення агрегації можна розглянути взаємозв’язок типу "частина-ціле ", що має місце між класом Системний блок персонального комп’ютера і його складових частин: Процесор, Материнська плата, Жорсткий диск і Відеокарта (рис. 8.12.).

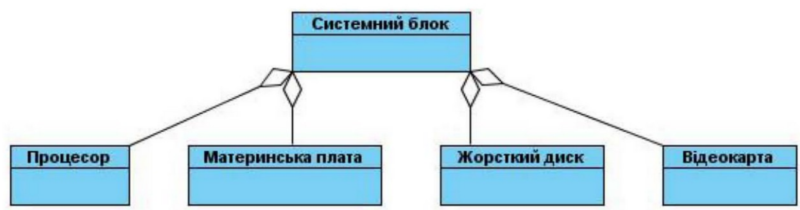


Рис. 8.12.

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.

Композит (composite) - клас, що зв'язаний відношенням композиції з одним або більшим числом класів.

Графічно відношення композиції зображується суцільною лінією, з зафарбованим усередині ромбом на одному з своїх кінців. Цей ромб указує на клас-композит. Інші класи є його "частинами" (рис. 8.13.)

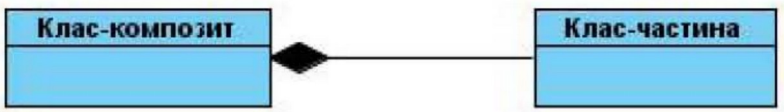


Рис. 8.13.

Практичним прикладом відношення композиції може служити вікно графічного інтерфейсу програми, що може складатися з рядка заголовка, смуг прокручування, головного меню, робочої області й рядка стану. Подібне вікно являє собою клас, проте його складові елементи також є окремими класами. Остання обставина характерна для відношення композиції, оскільки відображає різні способи зображення даного відношення.

Для відношення композиції й агрегації можуть використатися додаткові позначення, застосовувані для відношення асоціації. А саме, можуть бути зазначені кратності окремих класів, що в загальному випадку не обов'язково. Стосовно до описаного вище прикладу клас Вікно програми є класом-композитом, а взаємозв'язки складових його частин можуть бути зображені діаграмою класів наступного вигляду (рис. 8.14.).

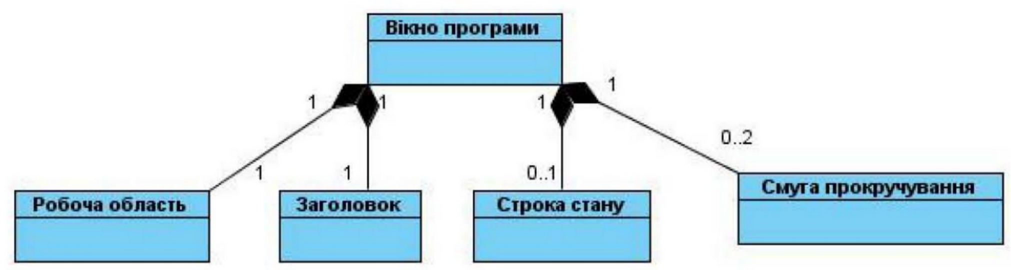


Рис. 8.14.

Прикладом відмінностей між застосуванням відношення агрегації та композиції може служити наступна діаграма взаємозв􏰀язків між будівлею, приміщенням та обладнанням, яке розташоване у приміщенні (рис. 8.15.).

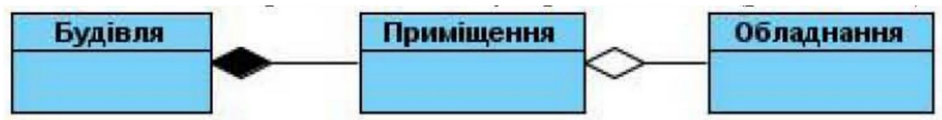


Рис. 8.15.

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 реалізації

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

Відносини 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 відносинами узагальнення та залежності (рис. 8.16.).

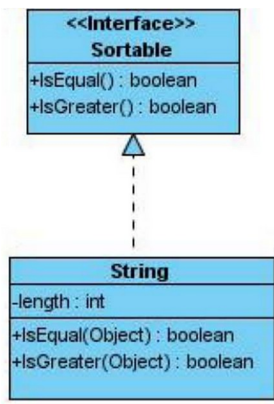


Рис. 8.16.

Приклади використання діаграм класів

1. Моделювання систем управління навчанням (LMS)

- **Опис**: У системі управління навчанням ми можемо мати класи, що представляють студента, курси, викладачів та модулі.
- **Структура**:
  - Клас Student має атрибути studentID, name, email і методи enrollCourse(), completeCourse().
  - Клас Course має атрибути courseID, title, credits і методи addModule(), assignInstructor().
  - Клас Instructor має атрибути instructorID, name, expertise і метод assignGrade().
  - Клас Module включає moduleID, name, content, а також метод addContent().
- **Зв’язки**:
  - Відношення “багато-до-багатьох” між Student і Course (студент може бути записаний на декілька курсів, і курс може мати декілька студентів).
  - Відношення “один-до-багатьох” між Instructor і Course (кожен курс може мати одного викладача, але один викладач може вести кілька курсів).

2. Інтернет-магазин

- **Опис**: У системі інтернет-магазину діаграма класів допомагає визначити основні об’єкти, такі як товари, користувачі, замовлення та оплата.
- **Структура**:
  - Клас Product з атрибутами productID, name, price, description і методами addToCart(), removeFromCart().
  - Клас User має атрибути userID, name, email і методи register(), login().
  - Клас Order має атрибути orderID, orderDate, status і методи placeOrder(), cancelOrder().
  - Клас Payment має атрибути paymentID, amount, paymentDate.
- **Зв’язки**:
  - Відношення “багато-до-багатьох” між Order і Product (одне замовлення може містити кілька товарів, і товар може бути в різних замовленнях).

• Відношення “один-до-одного” між Order і Payment (кожне замовлення має один платіж).

3. Система управління персоналом (HRM)

• **Опис**: У системі управління персоналом діаграма класів дозволяє відобразити зв’язки між працівниками, відділами та ролями.

• **Структура**:

• Клас Employee має атрибути employeeID, name, position, salary і методи applyForLeave(), submitReport().

• Клас Department має атрибути departmentID, name, location.

• Клас Role включає атрибути roleID, roleName, responsibilities.

• Клас Attendance має атрибути date, status і методи markAttendance(), checkStatus().

• **Зв’язки**:

• Відношення “один-до-багатьох” між Department і Employee (кожен відділ має кількох співробітників, але співробітник може працювати лише в одному відділі).

• Відношення “багато-до-багатьох” між Employee і Role (співробітник може мати кілька ролей, і роль може належати кільком співробітникам).

4. Бібліотечна система

• **Опис**: У бібліотечній системі діаграма класів може описувати книги, членів бібліотеки, персонал і процес позичання.

• **Структура**:

• Клас Book має атрибути bookID, title, author, ISBN, status і методи borrowBook(), returnBook().

• Клас Member має атрибути memberID, name, membershipDate і методи borrowBook(), renewMembership().

• Клас Staff включає staffID, name, role, а також метод manageInventory().

• Клас Loan описує позики з атрибутами loanID, borrowDate, dueDate.

• **Зв’язки**:

- Відношення “багато-до-багатьох” між Member і Book через клас Loan (один учасник може брати кілька книг, і одну книгу можуть брати кілька учасників у різний час).
- Відношення “один-до-багатьох” між Staff і Book (один співробітник може управляти кількома книгами).

5. Система для управління проектами

- **Опис**: У системі управління проєктами діаграма класів моделює зв’язки між проєктами, завданнями, учасниками команди та ресурсами.
- **Структура**:
  - Клас Project з атрибутами projectID, name, deadline, status.
  - Клас Task має атрибути taskID, title, priority, deadline.
  - Клас TeamMember має memberID, name, role.
  - Клас Resource має resourceID, type, availability.
- **Зв’язки**:
  - Відношення “один-до-багатьох” між Project і Task (кожен проєкт може мати багато завдань, але кожне завдання прив’язане до одного проєкту).
  - Відношення “багато-до-багатьох” між Task і TeamMember (одне завдання можуть виконувати кілька членів команди, а один член команди може працювати над кількома завданнями).

Отже діаграма класів є цінним інструментом для візуалізації структурної організації різних типів систем. Вона дозволяє легко визначити зв’язки між основними елементами системи та допомагає у створенні чіткої ієрархії класів, з якою можуть працювати різні команди та учасники проєкту.

8.2. Діаграма прецедентів (Use Case Diagram)

Діаграма прецедентів (Use Case Diagram) у UML використовується для моделювання функціональності системи з точки зору кінцевих користувачів або зовнішніх систем. Вона показує різні сценарії взаємодії користувачів із

системою через так звані прецеденти (use cases), дозволяючи візуалізувати вимоги та процеси, які система повинна виконувати.

Діаграми використання розробляються на ранній стадії проєктування системи та призначені для:

- простого пояснення роботи системи, створення та погодження ТЗ;
- формування функціональних (що має робити) вимог до системи;
- створення основи для документації та тестування.

Складові діаграми прецедентів

Діаграми прецедентів складаються з 4 об'єктів: актор, прецедент (варіант використання), система, зв'язок.

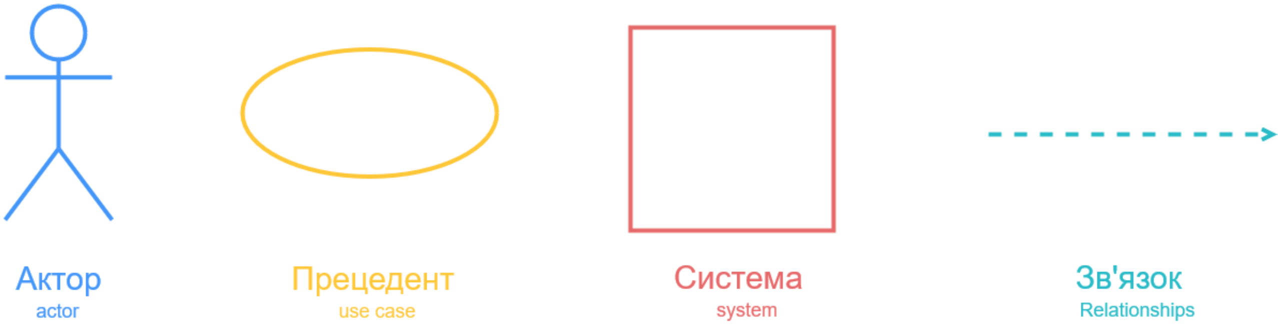


Рис. 8.17.

Актор. Поняття когось, або чогось, що взаємодіє з системою, але не належить до неї (знаходиться за межами системи). Зазвичай позначається у вигляді стилізованого чоловічка. Рідше у вигляді прямокутника класу з написом <<actor>>.

Усіх акторів умовно можна розділити на первинних (ті, що ініціюють взаємодію з системою) та вторинних (ті що реагують на взаємодію). Первинні зазвичай зображуються зліва, а вторинні — справа від системи. Оскільки будь-яка система може взаємодіяти не лише з людьми, актор не завжди позначає користувача-людину. Він може позначати іншу систему або пристрій, з яким взаємодіє.

Часто виникає плутанина між поняттями актор та користувач:

- *актор* — це поняття, що представляє клас користувачів (узагальнення групи користувачів), а не конкретного користувача, та може поєднувати в собі декілька ролей. Наприклад актор — працівник компанії може мати ролі інженер, менеджер, директор;
- *користувач* — це тип актора або його конкретна реалізація. Декілька користувачів можуть грати одну роль, тобто бути одним актором. Наприклад Іван та Роман — студенти. Також один користувач може належати до різних акторів, тобто виконувати різні ролі. Наприклад, в системі університету актор може бути викладачем в одному випадку, і науковим керівником в іншому.

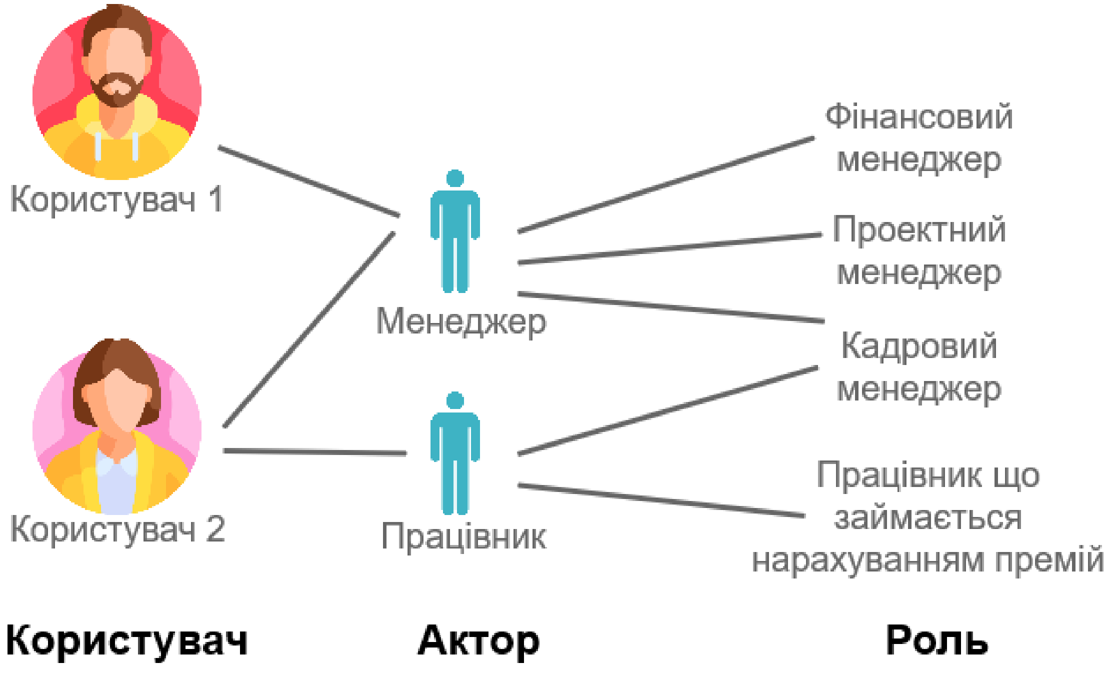


Рис. 8.18.

Випадок використання (прецедент). Прецеденти визначають очікувану поведінку та відповідають на питання, що робить система. Представляють набір можливих функцій, дій або завдань. Зображується у вигляді еліпса з назвою дії (дієсловом) у ньому. Прецедент вказує, що має трапитись, проте не відповідає на запитання, як це має статись.

До прикладу: в системі банківського додатка прецедентами можуть бути: перевірити баланс, переказати кошти, оплатити рахунок та ін.

Система. Те, що моделюється. Це може бути сайт, мобільний додаток або навіть модуль програмного забезпечення. Зображується у вигляді прямокутника з назвою системи у верхній частині.

Коментар розширює функціональність, надає підказки та умови.

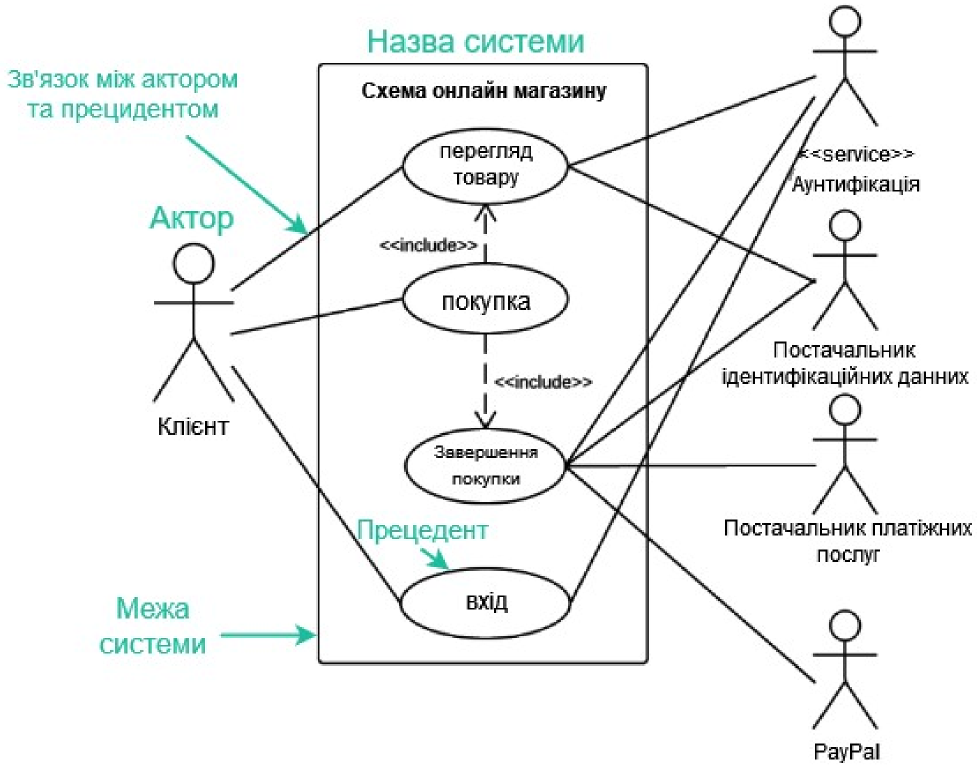


Рис. 8.19.

Зв'язки (відношення). Усі актори мають бути пов'язані з прецедентом. Проте не усі прецеденти повинні бути пов'язані з акторами. Частіше всього для зображення зв'язку використовується суцільна лінія.

У діаграмі варіантів використання існує 4 типи зв'язків:

1. **Асоціація (Association)**. Звичайний зв'язок актора та прецеденту. Позначається суцільною лінією без напису (стереотипу). Незалежно від типу зв'язку, будь-який актор повинен бути пов'язаний принаймні з одним (можна з декількома) варіантом використання. Кілька акторів можуть бути пов'язані з одним варіантом використання.

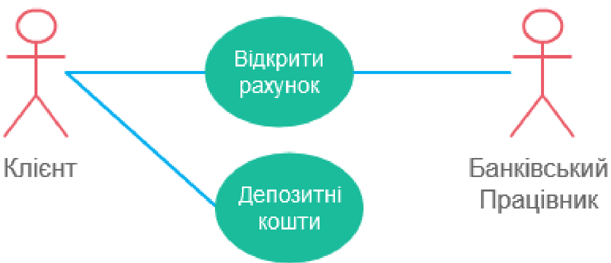


Рис. 8.20.

2. **Розширення (Extend).** Показує додаткову функціональність або можливий не обов'язковий варіант поведінки системи. Базовий прецедент має сенс сам по собі, не залежить від розширюючого і може існувати без нього. Відношення розширення позначається пунктирною лінією зі звичайним вказівником, що вказує на базовий прецедент та стереотипом (написом) <<extend>>. Розширюючий прецедент активується лише за виконання умови.

Наприклад: прецедент «хибний пароль» можливий лише при введенні невірного паролю. Відповідає extensions в текстовому варіанті.

Рис. 8.21.

Точки розширення (extension points). При використанні відношення розширення базовий варіант надає точки розширення (те саме, що extensions в текстовому описі) для розширюючого прецеденту. За бажанням вони можуть бути описані в окремому розділі базового прецеденту.

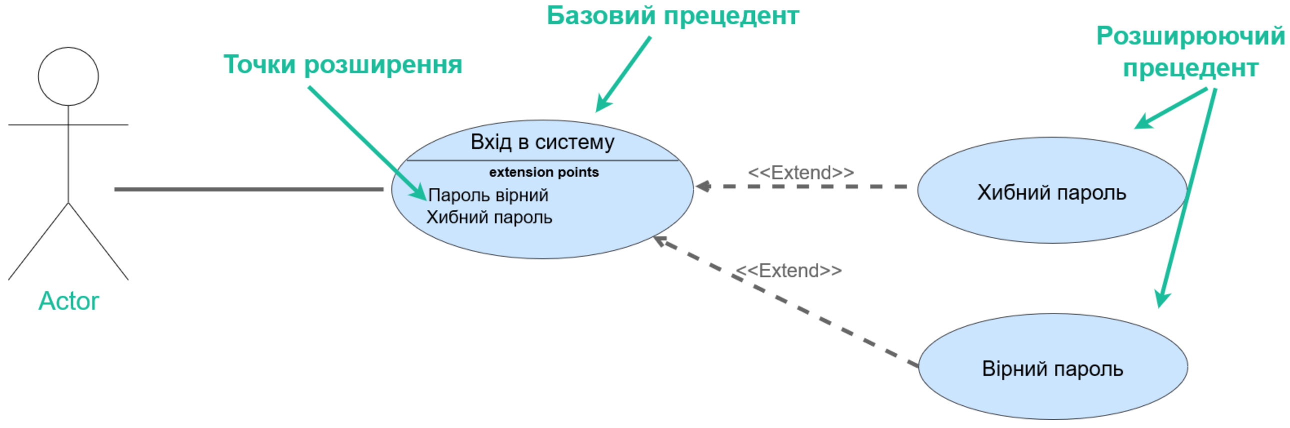


Рис. 8.22.

Якщо існує умова розширення (condition, те саме, що preconditions в текстовому варіанті) її можна описати в коментарі. Коментар з'єднується з пунктирною лінією з умовою розширення.

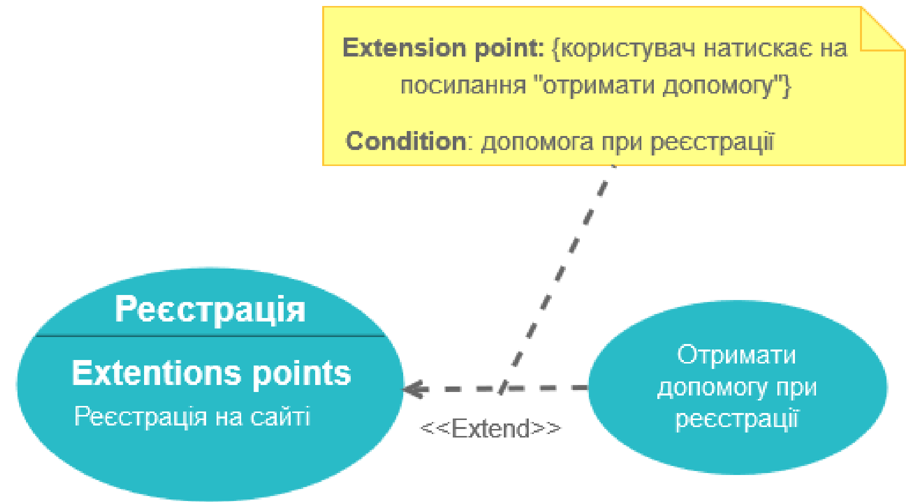


Рис. 8.23.

3. **Включення (Include)**. Показує, що поведінка одного прецеденту включається як складовий компонент у послідовність поведінки іншого прецеденту. Ілюструє, що саме використовує базовий варіант для виконання операції. На відміну від зв'язку розширення, дочірній варіант у зв'язку include має бути обов'язковим для базового. Відношення включення використовується для уникнення дублювання однакових прецедентів та додає функціональність, не вказану в базовому. Відношення позначається пунктирною лінією зі стрілкою та стереотипом <<include>>, що вказує на включений варіант.

Включення добре ілюструє сценарій відновлення працездатності комп'ютера (припустимо, що інших варіантів немає):

•ремонт або заміна апаратних компонентів;

•виявлення та видалення вірусу;

•перевстановлення системи.

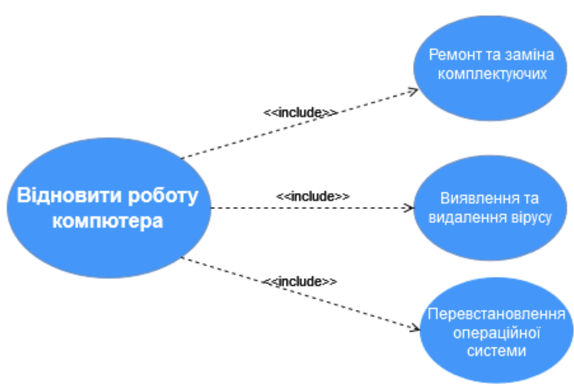


Рис. 8.24.

4. **Генералізація (Generalization).** Генералізація (успадкування, узагальнення) це батьківсько-дочірні відношення. Генералізація позначається суцільною лінією з трикутним не зафарбованим вказівником, що вказує на прецедент-предок.

Властивості відносин узагальнення:

•дочірні прецеденти мають всі властивості предків;

•для одного предка може існувати декілька дочірніх прецедентів;

•може бути кілька батьків (множинне успадкування).

Узагальнення актора. В узагальнені актора один актор може успадкувати роль іншого. Нащадок успадковує всі варіанти використання предка, проте може мати і власні унікальні варіанти.

Узагальнення варіанту використання. Схоже на узагальнення актора. В цьому випадку поведінка предка успадковується нащадком.

Використовується, коли існує спільна поведінка між двома варіантами використання. Наприклад, варіанти «Оплата банківською карткою» та «Оплата через Google Pay» можна узагальнити до «Оплата рахунку»

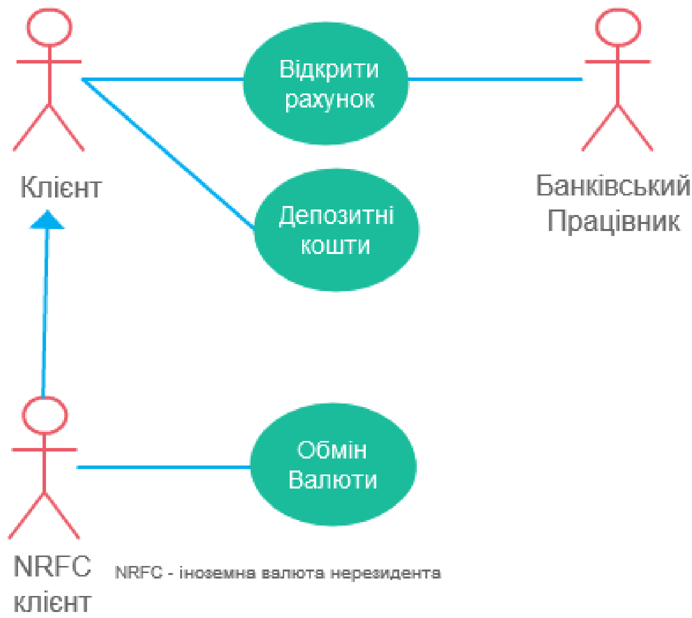


Рис. 8.25.

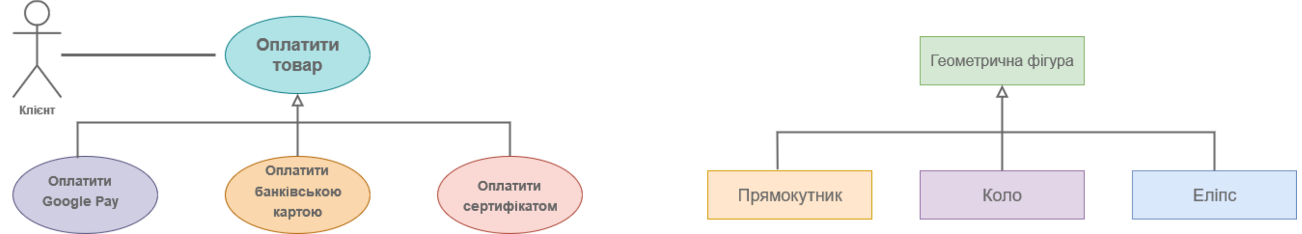


Рис. 8.26.

Приклади використання діаграм прецедентів

1. Інтернет-магазин

• **Опис**: У системі інтернет-магазину діаграма прецедентів може показувати сценарії, пов'язані з купівлею товарів, управлінням замовленнями та іншими процесами.

• **Актори**:

• **Клієнт** — користувач, який може переглядати товари, додавати їх до кошика, оформляти замовлення.

• **Адміністратор** — відповідальний за управління товарами, перегляд замовлень, обробку платежів.

• **Прецеденти**:

• **Перегляд товарів**: клієнт може переглядати список товарів і деталі кожного товару.

• **Додавання до кошика**: клієнт додає товар до кошика.

• **Оформлення замовлення**: клієнт завершує покупку, вибираючи спосіб доставки та оплати.

• **Оплата замовлення**: клієнт здійснює платіж за замовлення.

• **Обробка замовлення**: адміністратор обробляє замовлення, перевіряючи його статус і наявність товарів.

• **Зв'язки**:

• Прецедент "Оформлення замовлення" **включає** "Оплата замовлення" (платіж є обов'язковою частиною процесу оформлення замовлення).

• Прецедент "Обробка замовлення" **розширює** "Оформлення замовлення" (якщо адміністратор переглядає та обробляє замовлення після його оформлення).

2. Система управління персоналом (HRM)

• **Опис**: Діаграма прецедентів для HR-системи відображає сценарії, пов'язані з управлінням співробітниками, обліком робочого часу та зарплатою.

• **Актори**:

• **HR-менеджер** — відповідальний за управління співробітниками, перевірку інформації про них.

• **Співробітник** — має доступ до своїх даних, може переглядати графік і залишати запити на відпустку.

• **Прецеденти**:

• **Додавання співробітника**: HR-менеджер додає нового співробітника до системи.

• **Редагування інформації про співробітника**: HR-менеджер змінює дані співробітника.

• **Перегляд графіку**: співробітник може переглядати свій графік роботи.

• **Заявка на відпустку**: співробітник залишає запит на відпустку.

• **Погодження відпустки**: HR-менеджер розглядає заявку на відпустку.

• **Зв'язки**:

• Прецедент "Погодження відпустки" **включає** "Заявка на відпустку" (обробка заявки можлива лише після того, як співробітник її залишить).

• Прецедент "Редагування інформації про співробітника" **розширює** "Додавання співробітника" (HR-менеджер може внести додаткові зміни до даних після додавання співробітника).

3. Система бібліотеки

• **Опис**: У бібліотечній системі діаграма прецедентів описує сценарії, пов'язані з обслуговуванням читачів, позичанням книг і їх поверненням.

• **Актори**:

• **Читач** — може шукати книги, позичати їх і повертати.

• **Бібліотекар** — відповідальний за управління книгами, обслуговування читачів.

• **Прецеденти**:

• **Пошук книги**: читач шукає книгу за назвою або автором.

• **Позичення книги**: читач бере книгу в тимчасове користування.

• **Повернення книги**: читач повертає позичену книгу до бібліотеки.

• **Реєстрація читача**: бібліотекар додає нового читача до системи.

• **Додавання нової книги**: бібліотекар додає нову книгу до каталогу.

• **Зв'язки**:

• Прецедент "Позичення книги" **включає** "Пошук книги" (читач повинен знайти книгу перед її позичанням).

• Прецедент "Повернення книги" **розширює** "Позичення книги" (повернення відбувається після того, як книга була позичена).

4. Система банківського обслуговування

• **Опис**: Діаграма прецедентів для банківської системи може відображати сценарії, пов’язані з управлінням рахунками, проведенням транзакцій та переглядом балансу.

• **Актори**:

  • **Клієнт** — користувач, який має доступ до рахунку та може проводити операції.

  • **Банківський службовець** — відповідальний за управління рахунками, відкриття та закриття рахунків.

• **Прецеденти**:

  • **Відкриття рахунку**: банківський службовець створює новий рахунок клієнта.

  • **Поповнення рахунку**: клієнт вносить кошти на рахунок.

  • **Зняття коштів**: клієнт знімає гроші з рахунку.

  • **Перегляд балансу**: клієнт переглядає баланс свого рахунку.

• **Зв’язки**:

  • Прецедент “Зняття коштів” **включає** “Перегляд балансу” (перед зняттям клієнт може переглянути баланс).

  • Прецедент “Відкриття рахунку” **розширює** “Поповнення рахунку” (після відкриття рахунку клієнт може внести кошти).

5. Система управління навчанням (LMS)

• **Опис**: Діаграма прецедентів для системи управління навчанням моделює сценарії взаємодії студентів та викладачів із системою.

• **Актори**:

  • **Студент** — має можливість переглядати курси, записуватися на них і проходити тести.

  • **Викладач** — додає курси, створює завдання, оцінює студентів.

• **Прецеденти**:

  • **Перегляд курсу**: студент переглядає доступні курси.

- **Запис на курс**: студент реєструється для проходження курсу.
- **Проходження тесту**: студент проходить тест, який створений викладачем.
- **Створення курсу**: викладач додає новий курс.
- **Оцінювання**: викладач оцінює студентів.

• **Зв'язки**:

• Прецедент "Запис на курс" **включає** "Перегляд курсу" (студент може записатися лише після перегляду курсу).

• Прецедент "Оцінювання" **розширює** "Проходження тесту" (викладач оцінює після завершення студентом тесту).

Отже діаграма прецедентів допомагає зрозуміти, як різні користувачі взаємодіють із системою і які функції їм доступні. Вона є цінним інструментом для аналізу вимог, який дозволяє легко узгодити очікування між замовниками, розробниками та користувачами, забезпечуючи чітке уявлення про функціональні можливості системи.

8.3. Діаграма послідовності

Діаграма послідовності є одним із важливих інструментів в UML для моделювання взаємодії між об'єктами або компонентами системи з часом. Вона надає візуальне представлення послідовності повідомлень, які циркулюють між об'єктами, щоб виконати конкретну функцію або бізнес-процес. Діаграми послідовності допомагають краще зрозуміти, як відбувається взаємодія між різними компонентами системи, забезпечуючи виявлення можливих проблем в комунікації або логіці.

Діаграма послідовності (Sequence Diagram) — показує часові особливості передачі і прийому повідомлень об'єктами. Впорядкованість за часом слід розуміти як послідовність дій і не плутати з часовими діаграмами.

Позначення діаграми послідовності

1. Лінія життя починається з об'єкта-прямокутника (голова) та зображується вертикальною пунктирною лінією (стеблом). Вона служить для позначення

періоду часу, протягом якого об’єкт існує в системі. Якщо об’єкт існує в системі постійно, то його лінія життя повинна продовжуватися по всій площині діаграми зверху донизу. Зазвичай об’єкти перераховуються зліва направо. На рис. 8.27. зображено загальні елементи діаграми послідовності. На рис. 8.28. зображено позначення на лінії життя.

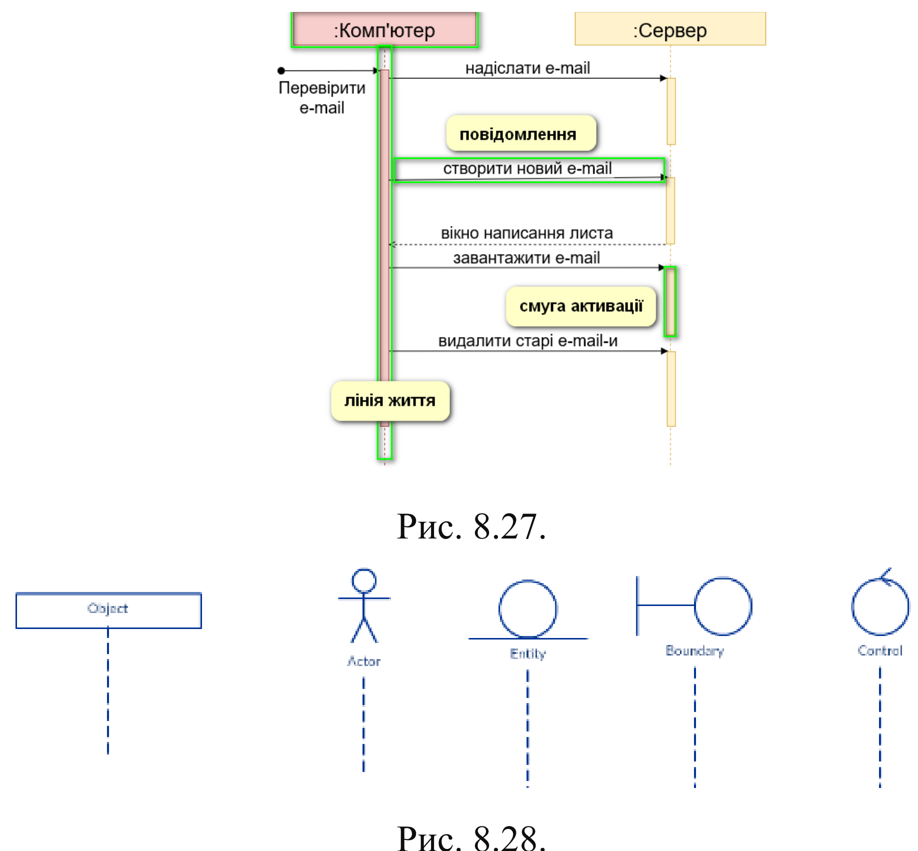


Рис. 8.27.

Рис. 8.28.

Об’єкт (учасник) — позначення лінії життя та екземпляр класу.

Актор — використовується, коли конкретна діаграма послідовності належить варіанту використання.

Сутність (entity) — представляє системні дані. Наприклад, у програмі обслуговування клієнтів суб’єкт — клієнт керує даними (сутність), пов’язаними з клієнтом.

Межа/кордон (boundary) — вказує на межу системи/граничний елемент у системі; наприклад, екрани інтерфейсу користувача, шлюзи баз даних або меню, з якими взаємодіють користувачі.

Управління (control) вказує на керівну сутність або менеджера. Він організовує та планує взаємодії між кордонами та сутностями та служить посередником між ними.

2. Смуга активації (фокус управління) — тонкий прямокутник на лінії життя, протягом якого елемент виконує операцію. Довжина прямокутника вказує на тривалість перебування об'єктів в активному режимі.

3. Повідомлення (виклики) з'являються в послідовному порядку на лінії життя. Повідомлення зображується за допомогою стрілок. Початок стрілки завжди торкається лінії життя відправника та лінії життя об'єкта, що приймає повідомлення. Підпис може знаходитись над або всередині стрілки повідомлення. Для зручності перед повідомленням можна проставляти нумерацію дій. Повідомлення можна розділити на такі категорії:

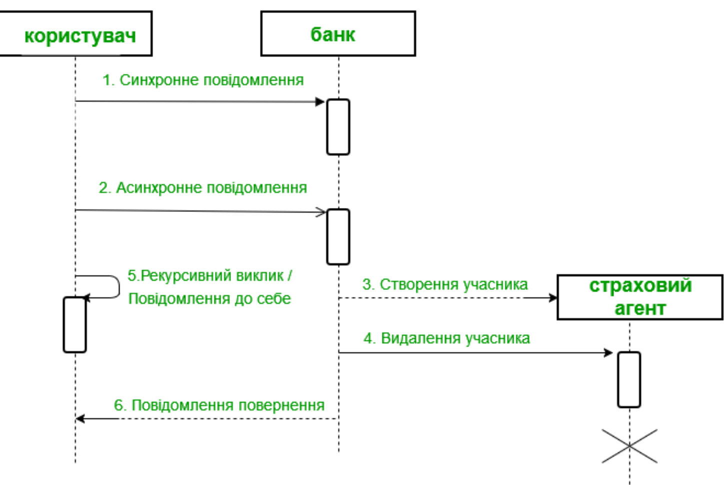


Рис. 8.29.

Повідомлення поділяються на *синхронні*, що очікують відповіді (зафарбований вказівник), та *асинхронні* — не очікують відповіді (пунктирна лінія, звичайний вказівник).

Рекурсивне повідомлення — направлене до себе (починається та закінчується на одній лінії життя). Прикладом може служити отримання доступу до камери смартфоном.

Повідомлення *повернення* (пунктирна лінія із зафарбованим вказівником) — зворотне повідомлення викликаючій стороні. Можна уникнути захаращення діаграм, вказуючи значення в самій стрілці початкового повідомлення.

Створення учасника — повідомлення, що створює нову лінію життя. Позначається пунктирною лінією, направленою до прямокутника учасника. Якщо створений учасник робить щось після його створення, слід додати вікно активації під його полем.

Видалення учасника позначається хрестом в кінці лінії життя учасника. При цьому, якщо стрілка повідомлення направлена від одного учасника до іншого, значить, один учасник видаляє іншого. Якщо стрілка відсутня, учасник самознищується.

Знайдене повідомлення — повідомлення від невідомого джерела.

4. Інші позначення:

•*Коментар* — прямокутник із загнутим кутом. Може бути з'єднаний з об'єктом пунктирною лінією.

•*Фрагмент* використовується, якщо процеси утворюють цикл або вимагають виконання умов для його закінчення. Він складається з вікна (рамки) та оператора фрагмента (напису) в п'ятикутнику зверху зліва.

Фрагмент діаграми послідовності (sd) використовується для оточення усієї діаграми. Після напису sd вказується назва діаграми.

Альтернативний фрагмент (alt) — показує один або декілька альтернативних сценаріїв, де виконується лише один, той, чия умова істинна.

Для опису двох або більше альтернативних сценаріїв використовуються пунктирні лінії — операнд взаємодії (interaction operands). Кожен операнд має умову захисту в квадратних дужках (guard condition).

Опціональний (не обов’язковий) фрагмент (opt) — виконується, лише якщо вказана умова, істинна. Він має лише одну умову і не поділяється на операнди. Використовується для опису не обов’язкового кроку робочого процесу.

Прикладом може слугувати послідовність покупок в інтернет-магазині: opt, щоб описати, як користувач може додати подарункове пакування. Та alt для опису двох варіантів оплати: за допомогою кредитної картки або грошового переказу.

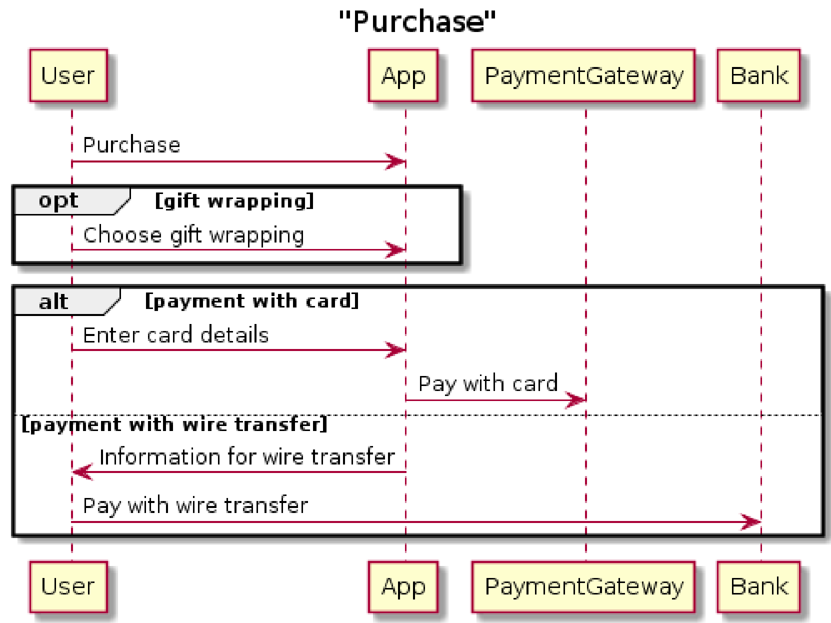


Рис. 8.30.

Фрагмент циклу (loop) — використовується для послідовності, що повторюється. Може мати межі ітерації, що пишуться біля назви loop.

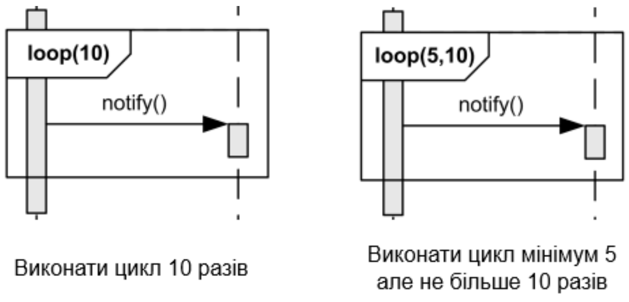


Рис. 8.31.

Фрагмент посилання (ref) — для повторного використання частини послідовності в іншому місці діаграми.

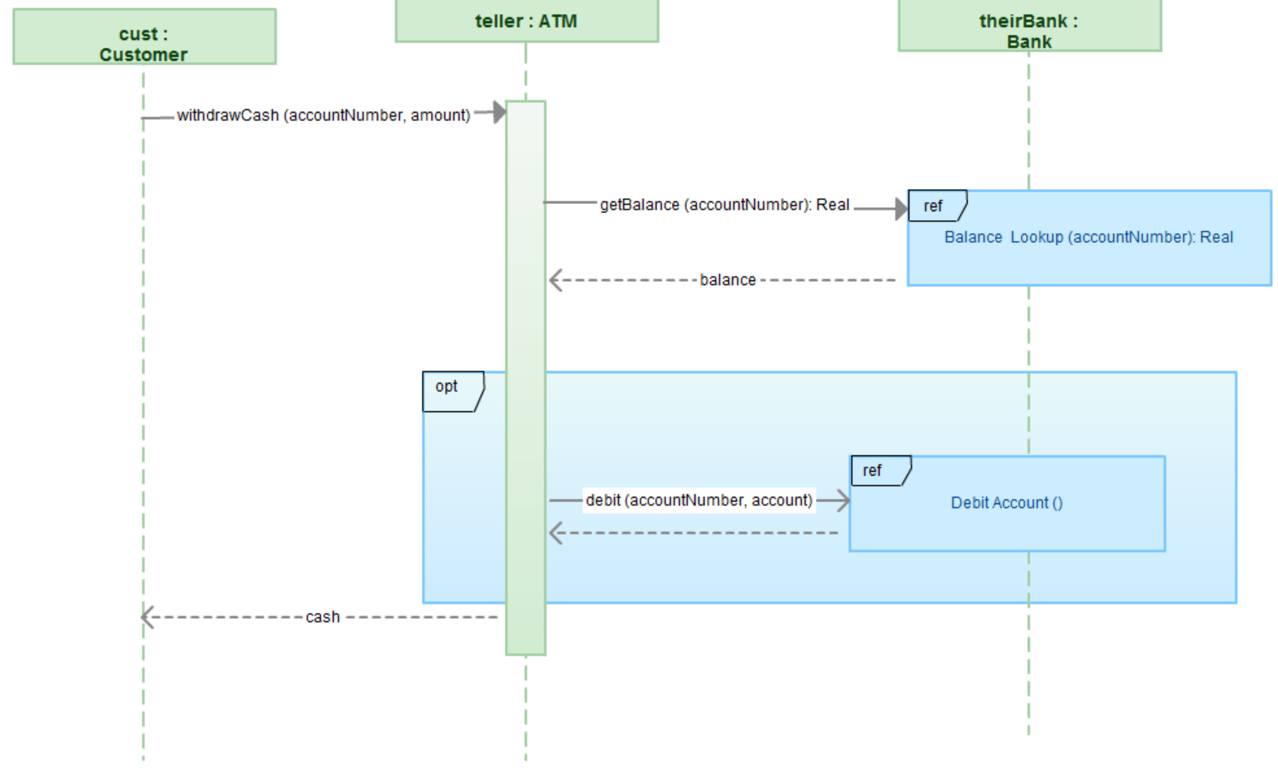


Рис. 8.32.

Рекомендації зі створення діаграми послідовності

•Оберіть тему діаграми послідовності.

•Визначте об’єкти або акторів, які будуть залучені до створення діаграми.

•Створіть короткий список взаємодій об’єктів, що відображатимуться на діаграмі послідовності.

•Визначте, якими типами повідомлень будуть обмінюватись об’єкти.

•Ви можете розфарбувати діаграму.

Поширені помилки

•Не додавайте багато деталей. Це захаращує діаграму та ускладнює її читання.

•Початок стрілки повідомлення має завжди торкатись лінії життя відправника, а вказівник — лінії життя об’єкта-одержувача.

•Повідомлення мають будуватись зліва направо.

•Не використовуйте діаграму послідовності, якщо необхідна реалізація простої логіки.

Приклади практичного використання діаграм послідовності

1. Авторизація користувача в веб-додатку

- **Контекст**: Користувач вводить логін і пароль на сторінці авторизації веб-додатку.
- **Процес**:
  - Користувач надсилає дані форми авторизації на сервер.
  - Сервер передає запит на перевірку облікових даних до бази даних.
  - База даних повертає результат (успішний вхід або помилка).
  - Якщо перевірка успішна, сервер створює сесію для користувача та надсилає відповідь з успішною авторизацією.
- **Корисність**: Діаграма послідовності допомагає зрозуміти послідовність запитів і відповідей між клієнтом і сервером, виявити можливі місця для оптимізації або покращення безпеки.

2. Процес оформлення онлайн-замовлення в інтернет-магазині

- **Контекст**: Користувач оформлює замовлення на сайті інтернет-магазину.
- **Процес**:
  - Користувач додає товари до кошика.
  - Коли користувач натискає «Оформити замовлення», система надсилає запит на перевірку наявності товару на складі.
  - Система перевіряє наявність кожного товару.
  - Якщо товари є в наявності, система створює замовлення, зберігає його в базі даних і надсилає користувачу повідомлення про успішне оформлення.
  - Якщо товарів недостатньо, система повідомляє користувача про нестачу та пропонує оновити кошик.

- **Корисність**: Діаграма дозволяє зрозуміти послідовність дій при оформленні замовлення і визначити можливі сценарії, коли товари можуть бути недоступними.

3. Використання чат-бота для обслуговування клієнтів

- **Контекст**: Клієнт звертається до чат-бота для отримання відповіді на своє питання.
- **Процес**:
  - Користувач надсилає запит в чат-бот.
  - Чат-бот перевіряє наявність готових відповідей у базі знань.
  - Якщо відповідь є, чат-бот надсилає її користувачу.
  - Якщо відповідь відсутня, чат-бот може передати запит до оператора.
  - Оператор відповідає на запит, і чат-бот надсилає відповідь користувачу.
- **Корисність**: Діаграма послідовності показує, як чат-бот взаємодіє з базою знань і оператором, що допомагає в оптимізації логіки його роботи.

4. Процес бронювання квитків на рейс

- **Контекст**: Користувач бронює квиток через онлайн-систему авіакомпанії.
- **Процес**:
  - Користувач вибирає рейс і надсилає запит на бронювання.
  - Система бронювання перевіряє наявність місць на обраному рейсі.
  - Якщо місця є, система запитує дані користувача для оплати.
  - Користувач вводить дані для оплати, які передаються у платіжну систему.
  - Платіжна система підтверджує або відхиляє оплату.
  - У разі успішної оплати система бронювання надсилає підтвердження бронювання та квиток.
- **Корисність**: Діаграма допомагає візуалізувати взаємодію між системою бронювання, платіжною системою і користувачем, а також зрозуміти, де можуть виникати потенційні збої.

5. Процес відправки повідомлення електронною поштою

- **Контекст**: Система відправляє автоматичне повідомлення користувачу після певної події.
- **Процес**:
  - Подія активує запит на відправлення повідомлення.
  - Система формує текст повідомлення та передає його на сервер електронної пошти.
  - Сервер перевіряє правильність адреси та можливість доставки.
  - Якщо перевірка успішна, повідомлення надсилається.
  - Якщо виникає помилка (наприклад, неправильна адреса), система отримує повідомлення про помилку.
- **Корисність**: Діаграма дозволяє простежити весь процес відправлення електронного листа, а також визначити можливі точки збоїв і варіанти їх обробки.

6. Процес реєстрації користувача в мобільному додатку

- **Контекст**: Користувач реєструється в мобільному додатку.
- **Процес**:
  - Користувач заповнює форму реєстрації.
  - Додаток надсилає дані на сервер для перевірки.
  - Сервер перевіряє правильність даних та їх унікальність.
  - Якщо дані коректні, сервер створює профіль користувача в базі даних.
  - Додаток отримує підтвердження, і користувач бачить повідомлення про успішну реєстрацію.
- **Корисність**: Діаграма допомагає чітко побачити, як дані реєстрації обробляються додатком і сервером, а також зрозуміти точки, де можуть виникнути проблеми з реєстрацією.

Отже діаграми послідовності є корисним інструментом для візуалізації процесів обміну повідомленнями між різними компонентами системи. Вони допомагають виявити можливі проблеми у взаємодії компонентів, підвищити ефективність роботи системи та створити оптимальні умови для виконання

певних дій. Ці діаграми використовуються у бізнес-процесах, автоматизації, розробці додатків, допомагаючи покращити взаємодію між елементами системи та підвищити якість програмного забезпечення.

8.4. Діаграма діяльності

Діаграми діяльності (або активності) є корисними для моделювання різних аспектів роботи системи або бізнес-процесів, оскільки вони відображають послідовність дій, альтернативні шляхи, розгалуження, паралельні потоки й кінцеві результати. Діаграми діяльності широко використовуються для візуалізації та аналізу алгоритмів, робочих процесів і бізнес-процедур.

Діаграма діяльності візуалізує процес використання та ілюструє потік повідомлень від однієї дії до іншої. Показує цілісну роботу системи.

Діаграма діяльності вважається розширеним варіантом блок-схем. Проте блок-схема не має стандартизованої нотації та не містить паралелізмів (паралельне виконання дій) та синхронізації.

Також часто виникає питання, у чому різниця між діаграмою діяльності та послідовності. Головна мета діаграми послідовності — показати порядок виконання, або послідовність дій. Водночас діаграма діяльності потрібна для опису роботи всієї системи, вона показує перехід від однієї дії до іншої.

Ці дії можуть виконуватися людьми, програмними компонентами або комп’ютерами. Потік керування (порядок виконання) на діаграмі діяльності переходить від однієї операції до іншої. Цей потік може бути послідовним, розгалуженим або одночасним. На рис. 8.33. показано різницю між діаграмою діяльності і діаграмою послідовності.

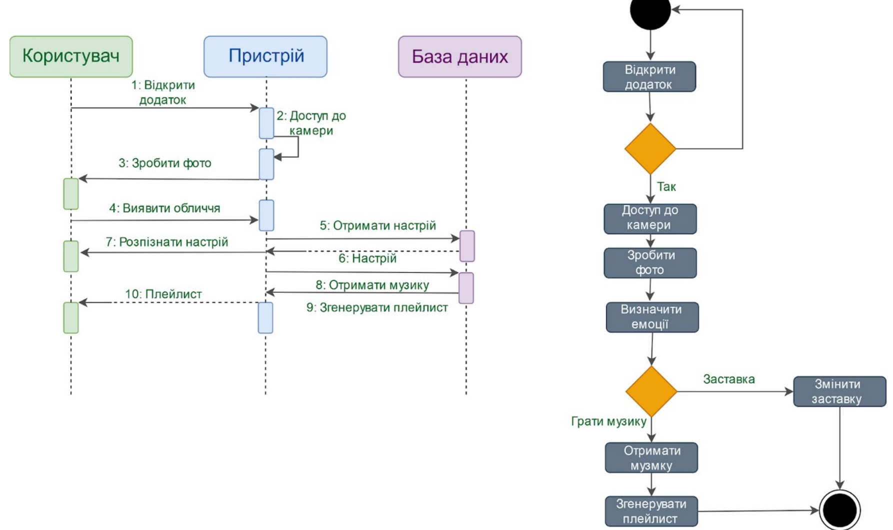
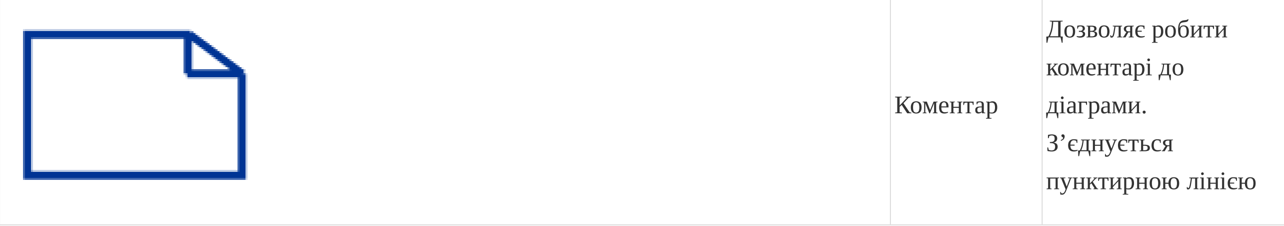


Рис. 8.33.

Позначення діаграми діяльності

Символ	**Ім’я**	**Використання**
	Початковий вузол	Відправна точка, або початковий стан
Activity	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 і 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
Signal Sending	Надсилання сигналу	Вказує на те, що сигнал надсилається на приймальну діяльність
Signal Receipt	Отримання сигналу	Вказує на отримання сигналу

	Коментар	Дозволяє робити коментарі до діаграми. З’єднується пунктирною лінією

Потік керування — з’єднує два вузли на діаграмі активності. Вузол розгалуження (вилка) — це вузол керування, який розділяє потік на кілька одночасних. Потрібен для створення декількох одночасних завдань. Вузол злиття об’єднує ці потоки.

Вузол прийняття рішень — це вузол керування, який вибирає між декількома потоками один істинний. Він схожий на оператор if в Java або C#. Умови (охоронні умови) записуються в квадратних дужках поруч з потоком.

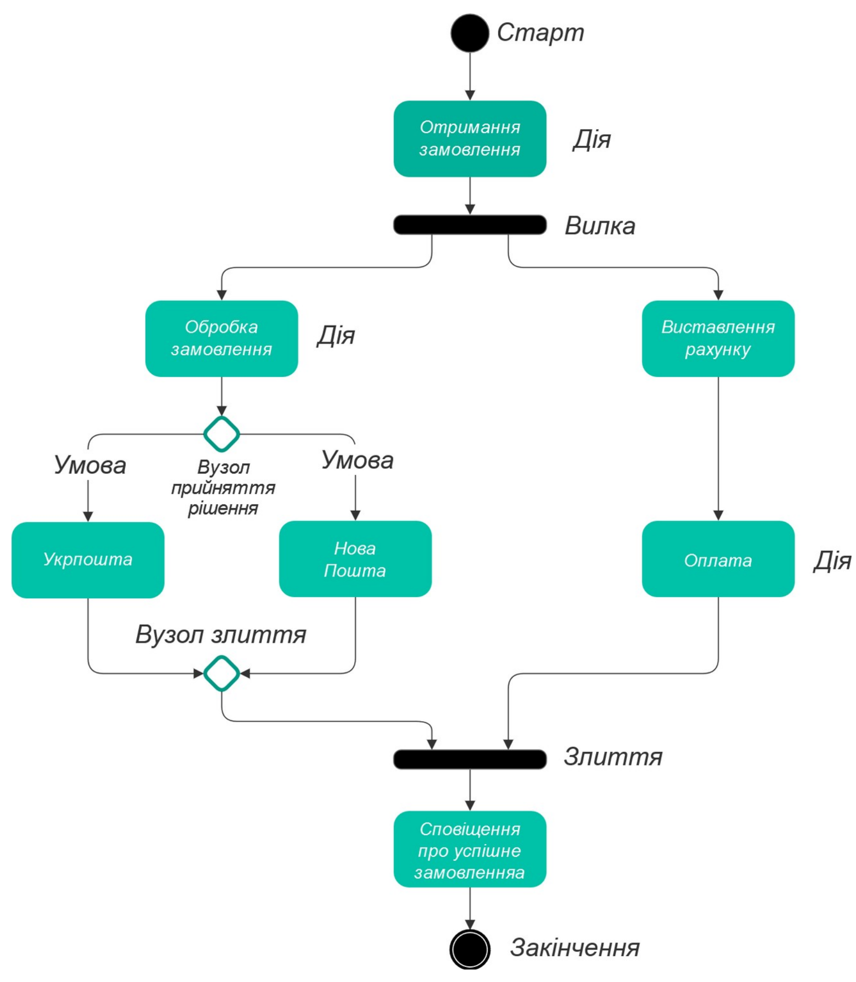


Рис. 8.34.

Доріжки (swimlanes) — це спосіб групування дій, які виконуються одним актором, або декількома акторами в одному потоці. Не додавайте більш як 5 доріжок одночасно. Розташовуйте доріжки у логічному порядку.

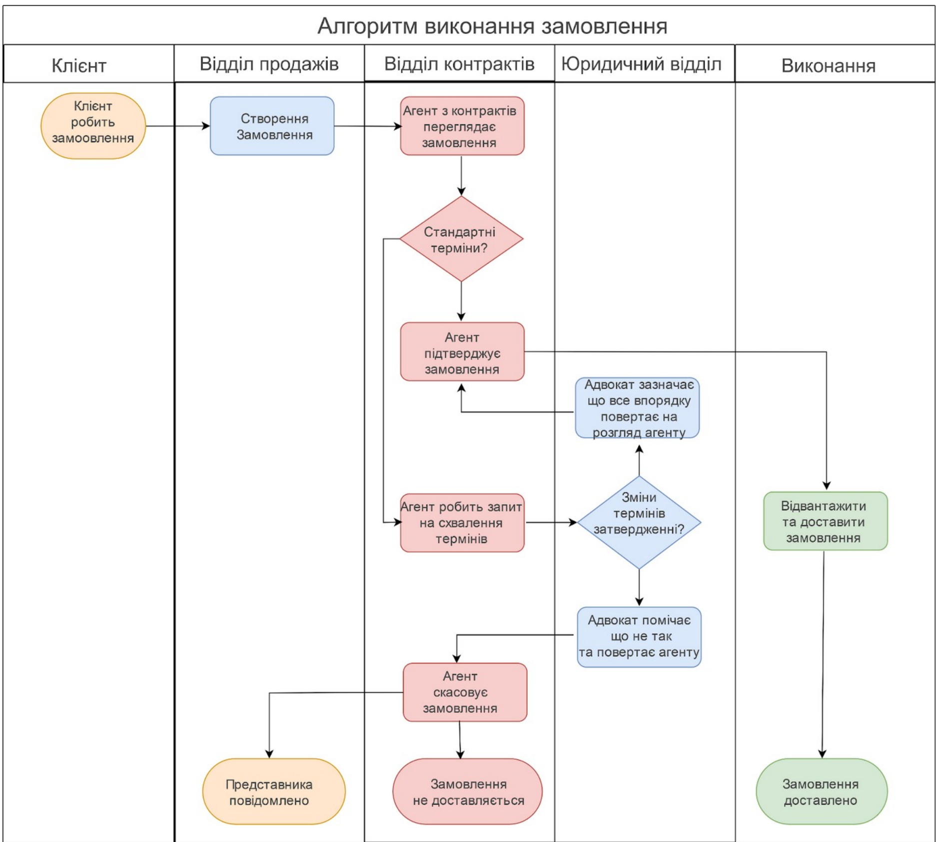


Рис. 8.35.

Практичні приклади використання діаграм діяльності

1. Онлайн-замовлення продукту

- **Контекст**: Інтернет-магазин, де користувач здійснює замовлення продукту.
- **Діаграма діяльності** відображає процес від вибору товару до отримання замовлення.
- **Процес**:
  - Користувач переглядає товари та додає потрібні до кошика.
  - Переходить до оформлення замовлення, вказує дані доставки.
  - Вибирає метод оплати та завершує покупку.
  - Замовлення обробляється, надсилається підтвердження.

- Після успішної оплати товар доставляється, а користувач отримує повідомлення про доставку.

- **Корисність**: Допомагає аналізувати та оптимізувати процес оформлення та обробки замовлення.

2. Процес реєстрації в системі

- **Контекст**: Реєстрація нового користувача в інформаційній системі.
- **Процес**:
  - Користувач заповнює форму реєстрації.
  - Система перевіряє коректність введених даних, включно з унікальністю імені користувача.
  - Якщо дані некоректні, система запитує їх повторне введення.
  - Якщо дані успішно пройшли перевірку, система створює акаунт і відправляє електронний лист для підтвердження.
  - Користувач підтверджує електронну адресу, і акаунт стає активним.
- **Корисність**: Відображає кожен етап реєстрації і показує можливі варіанти розвитку подій, наприклад, коли дані заповнені неправильно.

3. Бізнес-процес обробки замовлення на виробництві

- **Контекст**: Обробка замовлення на підприємстві, яке виготовляє продукцію.
- **Процес**:
  - Після отримання замовлення його перевіряють на відповідність можливостям виробництва.
  - Визначається необхідність додаткових матеріалів; якщо так, створюється замовлення на закупівлю.
  - Виробничий процес запускається, коли матеріали отримані.
  - Готову продукцію перевіряють на відповідність якості та упаковують.
  - Продукцію відправляють замовнику.
- **Корисність**: Відображення діяльності дозволяє ефективно управляти ресурсами та передбачати можливі затримки.

4. Алгоритм обробки запиту на технічну підтримку

- **Контекст**: Компанія, що надає послуги технічної підтримки.
- **Процес**:
    - Користувач звертається із запитом до служби підтримки.
    - Оператор отримує запит і класифікує його за рівнем складності.
    - Якщо запит простий, оператор вирішує його одразу та повідомляє користувача.
    - Якщо запит складний, його передають технічним фахівцям, які аналізують і усувають проблему.
    - Після розв’язання проблема відзначається як вирішена, а користувач отримує повідомлення про закриття запиту.
- **Корисність**: Діаграма дозволяє візуалізувати процес обробки запитів і покращити комунікацію між різними відділами підтримки.

5. Процес обробки банківського платежу

- **Контекст**: Проведення банківського платежу в онлайн-системі.
- **Процес**:
    - Користувач вводить платіжні дані та ініціює переказ.
    - Система перевіряє баланс на рахунку.
    - Якщо баланс достатній, платіж відправляється на обробку.
    - Якщо баланс недостатній, користувач отримує повідомлення про відмову.
    - Успішний платіж підтверджується, і користувач отримує квитанцію.
- **Корисність**: Діаграма дозволяє чітко побачити всі можливі сценарії обробки платежу, допомагаючи виявити потенційні ризики й оптимізувати роботу платіжної системи.

Отже діаграми діяльності є потужним інструментом для проєктування, документування та оптимізації процесів у різних системах. Вони дозволяють легко побачити послідовність дій, розгалуження і можливі альтернативи для кожної задачі або бізнес-процесу, що полегшує їхнє розуміння й удосконалення.

9. StarUML

StarUML — це програмний інструмент для моделювання на основі мови UML (Unified Modeling Language). Він використовується розробниками для створення діаграм, які описують структуру та поведінку програмних систем. StarUML підтримує багатий набір діаграм UML і застосовується на етапах проєктування, аналізу та документування в процесі розробки програмного забезпечення. Основною метою StarUML є забезпечення зручного середовища для побудови моделей, що допомагають візуалізувати та оптимізувати проєктування системи.

Основні характеристики та можливості StarUML

1. **Підтримка різних діаграм UML**:

 • StarUML підтримує 13 основних типів діаграм UML, включаючи діаграми класів, прецедентів, послідовностей, станів, активностей та інших. Це дозволяє користувачам повністю охопити різні аспекти моделювання системи.

2. **Модульна архітектура та розширення**:

 • StarUML дозволяє додавати модулі для підтримки додаткових функцій, наприклад, генерації коду, зворотного інжинірингу та інтеграції з іншими системами. Модульна архітектура забезпечує гнучкість, і користувачі можуть адаптувати програму під власні потреби.

3. **Генерація коду та зворотній інжиніринг**:

 • Інструмент підтримує автоматичну генерацію коду для мов програмування, таких як Java, C#, та деяких інших, що допомагає розробникам швидко створювати шаблони програм. Зворотний інжиніринг дозволяє імпортувати існуючий код і створювати UML-діаграми на його основі, полегшуючи аналіз існуючих систем.

4. **Експорт і імпорт**:

 • StarUML дозволяє експортувати діаграми в різні формати зображень (PNG, JPG, SVG), а також у формат PDF. Це спрощує обмін документацією та модельними матеріалами з іншими учасниками проєкту.

5. **Підтримка ERD**:

• Окрім UML-діаграм, StarUML підтримує діаграми для проєктування баз даних (ERD — Entity-Relationship Diagram), що розширює його можливості у сфері моделювання інформаційних систем і баз даних.

6. **Мультиплатформенність**:

• StarUML доступний для Windows, macOS та Linux, що робить його зручним для команд, які працюють на різних операційних системах.

Переваги StarUML

• **Інтуїтивний інтерфейс**: Дозволяє легко створювати та редагувати діаграми, навіть для новачків у моделюванні UML.

• **Можливість кастомізації**: Розробники можуть використовувати скрипти та модулі для адаптації інструмента під специфічні потреби проєкту.

• **Низька вартість**: StarUML пропонує свої можливості за доступну ціну порівняно з деякими іншими інструментами для моделювання, такими як IBM Rational Rose чи Microsoft Visio.

• **Спільнота та документація**: Наявність онлайн-документації та великої спільноти користувачів, що дозволяє знайти багато ресурсів і прикладів для вивчення.

Приклади використання StarUML

1. **Проєктування архітектури програми**: Можливість створювати діаграми класів для опису об'єктно-орієнтованої архітектури, а також діаграми послідовностей для моделювання взаємодії об'єктів.
2. **Розробка баз даних**: За допомогою ERD можна проєктувати структуру бази даних, створюючи логічну модель для подальшої реалізації.
3. **Бізнес-процеси та вимоги**: Діаграми прецедентів дозволяють проєктувати сценарії використання та вимоги системи, що є важливим етапом у процесі розробки програмного забезпечення.

Отже StarUML — це потужний і зручний інструмент для моделювання, який охоплює широкий спектр завдань, від створення діаграм UML до генерації

коду та зворотного інжинірингу. Він підходить як для професіоналів, так і для студентів, які хочуть навчитися основам об’єктно-орієнтованого моделювання. Завдяки підтримці різноманітних діаграм, розширюваності та мультиплатформенності, StarUML є одним із найпопулярніших інструментів для моделювання в процесі розробки програмного забезпечення.

10. Проєктування інтерфейсів інформаційних систем

Проєктування інтерфейсів інформаційних систем є важливою частиною розробки програмного забезпечення, оскільки якість інтерфейсу користувача (UI) значною мірою визначає зручність, ефективність та продуктивність взаємодії користувачів із системою. Процес проєктування інтерфейсу орієнтується на забезпечення інтуїтивно зрозумілого, зручного та естетично приємного користувацького досвіду (UX).

Основні аспекти проєктування інтерфейсів інформаційних систем

1. **Аналіз вимог та профілю користувача**

• Визначення цільової аудиторії: розуміння, хто є основними користувачами системи, допомагає краще адаптувати інтерфейс для їхніх потреб.

• Аналіз задач користувача: виявлення основних функцій, які користувачі будуть виконувати в системі, дозволяє зосередити інтерфейс на зручному виконанні цих задач.

2. **Інтуїтивність і зручність використання**

• Інтерфейс повинен бути інтуїтивно зрозумілим, щоб користувачі могли легко освоїти його без довготривалого навчання. Для цього використовуються зрозумілі іконки, прості меню, мінімальна кількість кроків для виконання основних дій.

• Використання стандартів та шаблонів: наприклад, панелі навігації, кнопки дій (наприклад, “зберегти”, “видалити”), зрозумілі повідомлення та вказівки.

3. **Консистентність і сталість**

• Консистентність в інтерфейсі означає, що однакові елементи мають виглядати та працювати однаково по всій системі. Це зменшує криву навчання та спрощує користування.

4. **Візуальна ієрархія**

• Елементи інтерфейсу повинні бути розміщені так, щоб ключові об'єкти (кнопки, меню) легко виділялися. Застосування кольорів, розмірів і відступів сприяє фокусуванню уваги на основних елементах.

5. **Зворотний зв'язок для користувача**

• Система повинна надавати чіткий зворотний зв'язок на дії користувача (наприклад, підтвердження виконаних дій, помилки або повідомлення про успіх), щоб користувач завжди розумів, що відбувається.

6. **Адаптивний дизайн**

• Інформаційні системи часто використовуються на різних пристроях (десктопи, планшети, смартфони), тому адаптивний дизайн забезпечує зручний інтерфейс для різних розмірів екранів.

Ключові етапи проєктування інтерфейсів інформаційних систем

1. **Створення прототипу**

• Прототип дозволяє розробникам, дизайнерам та клієнтам швидко оцінити загальну структуру інтерфейсу, розташування елементів та зручність користування. Прототипи можуть бути низької деталізації (скетчі) або високої деталізації (інтерактивні макети).

2. **Тестування з користувачами**

• На основі прототипу проводять тестування з реальними користувачами. Це дозволяє виявити проблеми на ранніх стадіях і зрозуміти, як користувачі взаємодіють з інтерфейсом.

3. **Ітеративне вдосконалення**

• На основі відгуків і результатів тестування, інтерфейс постійно доопрацьовується та вдосконалюється. Це дозволяє отримати кінцевий продукт, що максимально відповідає потребам користувачів.

4. **Створення фінального дизайну**

• Після проведення тестувань і внесення всіх виправлень створюється остаточний дизайн, який потім передається на етап розробки.

Інструменти для проєктування інтерфейсів

1. **Figma** — популярний інструмент для створення прототипів та дизайну інтерфейсів з можливістю колаборації.
2. **Adobe XD** — забезпечує функціонал для проєктування, прототипування та тестування інтерфейсів.
3. **Sketch** — популярний інструмент для дизайну інтерфейсів на macOS.
4. **Axure RP** — підходить для створення прототипів високої деталізації з інтерактивністю.
5. **InVision** — дозволяє створювати інтерактивні прототипи та тестувати їх з користувачами.

Приклади інформаційних систем з ефективним інтерфейсом

1. **CRM-системи** (наприклад, Salesforce, HubSpot): забезпечують зручне управління клієнтами, мають зрозумілий інтерфейс для відстеження комунікацій та аналітики.
2. **ERP-системи** (наприклад, SAP, Oracle ERP): складні системи, які завдяки добре продуманому інтерфейсу забезпечують управління ресурсами підприємства та автоматизацію процесів.
3. **Системи для інтернет-банкінгу** (наприклад, Приват24, Revolut): зручний інтерфейс, оптимізований під мобільні пристрої, для швидкого виконання банківських операцій.

Проєктування інтерфейсів інформаційних систем — це важливий процес, який сприяє ефективній взаємодії користувача із системою, покращує якість обслуговування та продуктивність. Кожен етап розробки інтерфейсу має своє значення і разом створює системи, які відповідають сучасним вимогам зручності та адаптивності.

11. Завдання до заключної лабораторної роботи

За наданим варіантом розглянути описану у цьому варіанті предметну галузь та за допомогою програми StarUML побудувати діаграми класів, прецедентів, діяльності та послідовності.

Варіанти завдань для самостійного виконання

Варіант 1.

1. Авіарейс характеризується: місцем призначення, літаком та кількістю проданих квитків. Місце призначення, як окремий клас, характеризується: містом призначення та кількістю кілометрів до нього. Літак, як окремий клас, характеризується середньою швидкістю польоту та вантажопідйомністю (максимальна вага вантажу, який літак може взяти на борт). Описати клас "Авіарейс", в якому передбачити метод підрахунку ваги поштового вантажу (в кг), який можна розмістити на борту (враховуючи, що середня вага одного пасажира разом з багажем рівна 150 кг). Використати екземпляр класу „Авіарейс" для визначення, чи буде доставлений певний поштовий вантаж заданої ваги цим літаком до місця призначення і якщо так, то через скільки годин.

Варіант 2.

1. Вибираючи туристичну путівку, клієнт вибирає: місце відпочинку, період відпочинку, місце проживання та транспортний засіб, яким він зможе дістатися до місця відпочинку. Місце проживання, як окремий клас, характеризується назвою типу місця проживання (готель, турбаза, кемпінг, палата) та містить метод підрахунку вартості проживання за добу в залежності від вибраного місця проживання (відповідне відношення придумати самостійно). Транспортний засіб, як окремий клас, характеризується назвою типу транспортного засобу (літак, поїзд, автобус, корабель) та містить метод підрахунку вартості проїзду в один кінець в залежності від вибраного транспортного засобу (відповідне

відношення придумати самостійно). Описати клас „Туристична путівка”, в якому передбачити метод підрахунку загальної вартості путівки, що складається з вартості проживання, вартості дороги в двох напрямках та 30% від загальної суми за послуги туристичної фірми. Використати екземпляр класу „Туристична путівка” для підрахування вартості деякої туристичної путівки.

Варіант 3.

1.Овочева культура характеризується: назвою, сортом, параметрами дозрівання, висаджується на певній площі (в Га) і має врожайність. Параметри дозрівання, як окремий клас, характеризується числом і місяцем посадки, кількістю місяців і днів дозрівання та містить метод обчислення дати дозрівання. Врожайність, як окремий, клас характеризується вагою отриманого врожаю з 1 Га та можливим відсотком псування врожаю. Описати клас „Овочева культура”, в якому передбачити метод обчислення загальної врожайності з засіяної площі з урахуванням можливих втрат від псування врожаю. Використати екземпляр класу „Овочева культура” для визначення дати дозрівання та врожайності деякої культури.

Варіант 4.

1. Поштова бандероль характеризується: адресою відправника, адресою отримувача, датою відправлення та додатковими параметрами. Адреса відправника та отримувача, як окремий клас, характеризується вулицю, номерами будинку та квартири. Додаткові параметри, як окремий клас, характеризується вагою (кг) та цінністю (грн) вмісту бандеролі та містить метод обчислення вартості її доставки в залежності від ваги і цінності (відповідне відношення придумати самостійно). Дата відправлення, як окремий клас, характеризується днем і місяцем відправлення та містить метод визначення дати доставки бандеролі (бандероль доходить до одержувача через 7 днів після відправлення). Описати клас „Бандероль”, в якому передбачити метод обчислення виплати за затримку бандеролі в розмірі 10% від в її цінності за

кожен день затримки. Використати екземпляр класу „Бандероль” для визначення чи була доставлена деяка бандероль вчасно (день і місяць реальної доставки має вказати користувач) і в разі її затримки обчислити розмір повернення коштів.

Варіант 5.

1. Власник квартири характеризується: прізвищем, адресою та комунальними витратами. Адреса, як окремий клас, характеризується: назвою вулиці, номером будинку, номером квартири, розміром місячної оплати за прибирання прибудинкової території та містить метод обрахунку поверху розташування квартири (в місті побудовані будинки з трьома квартирами на поверсі та з одним під’їздом). Комунальні послуги, як окремий клас, характеризується: назвою звітного місяця, показниками лічильника на початок і кінець місяця за електроенергію (кВт*год) і воду ($м^3$) та містить методи підрахунку плати за використану електроенергію та воду (окремо для кожного ресурсу), враховуючи, що на поверхи вище 4-го вода не завжди доходить через поганий тиск, мешканці цих поверхів за воду платять на 50% менше (тарифи на електроенергію та воду придумати самостійно). Описати клас „Власник квартири”, в якому передбачити метод обчислення загальної плати за комунальні послуги. Використати екземпляр класу „Власник квартири” для визначення залишків коштів після оплати комунальних послуг при вказаних місячних доходах.

Варіант 6.

1. Метеорологічні спостереження характеризуються: датою та параметрами вимірювання погодних умов. Дата, як окремий клас, характеризується місяцем, днем, кількістю денних годин, годиною, яка вважається початком дня та містить метод, який перевіряє, чи входить вказана година в денні години. Параметри вимірювання погодних умов, як окремий клас, характеризується вимірами температури кожні 6 годин (о 1, 7, 13, 19 годинах) та містить метод обчислення

середньодобової температури. Описати клас “Метеорологічні спостереження”, в якому передбачити методи обчислення середньо-денної та середньо-нічної температури. Використати екземпляр класу „Метеорологічні спостереження” для визначення середньодобової, середньо-денної та середньо-нічної температури в деякий день.

Варіант 7.

1.Робітник характеризується: прізвищем, освітою (середня, незавершена вища, вища), місцем проживання, заробітною платнею. Місце проживання, як окремий клас, характеризується: назвою вулиці, відстанню до підприємства (в км) та містить метод обчислення компенсації за проїзд на роботу, яка становить 0,1 % від заробітної плати за 1 Км. Заробітна платня, як окремий клас, характеризується фіксованим розміром платні та містить метод обчислення надбавки за наявність освіти (розмір надбавки придумати самостійно). Описати клас “Робітник”, в якому передбачити методи підрахунку загальної заробітної плати та компенсації за проїзд. Використати екземпляр класу „Робітник” для підрахунку загальних грошових надходжень деякого робітника.

Варіант 8.

1. Замовлення на товар характеризується: замовником та товаром. Замовник, як окремий клас, характеризується назвою фірми, сумою коштів попередньо зроблених покупок, від якої залежить % знижки та методом підрахунку знижки (відповідне відношення придумати самостійно). Товар, як окремий клас, характеризується назвою товару, кількістю замовлених одиниць, ціною за одну одиницю та містить метод підрахунку загальної вартості, на яку замовлено товар. Описати клас “Замовлення”, в якому передбачити метод підрахунку суми, яку потрібно оплатити постачальнику та метод збільшення суми коштів попередньо зроблених покупок на суму поточної покупки. Використати екземпляр класу “Замовлення” для моделювання ситуації покупки деякого товару.

Варіант 9.

1.Студент характеризується: прізвищем, місцем проживання та результатами навчання. Місце проживання, як окремий клас, характеризується відміткою чи проживає студент в гуртожитку та містить метод встановлення місячної плати за гуртожиток (розмір плати за гуртожиток придумати самостійно). Результати навчання, як окремий клас, характеризується оцінками з 4-х іспитів, які отримав студент в останню сесію та методоам обчислення середнього балу та методом встановлення розміру стипендії в залежності від середнього балу. Описати клас “Студент”, в якому передбачити метод підрахунку суми коштів, які студент отримуватиме на руки. Використати екземпляр класу “Студент” для визначення отриманих коштів студентом чи заборгованості на кінець місяця.

Варіант 10.

1. Читач бібліотеки характеризується: прізвищем, адресою, назвою книги, її ціною, датою її видачі та відміткою, чи була повернена попередня книга. Адреса, як окремий клас, характеризується назвою вулиці, відстанню до бібліотеки (в Км) та методом обчислення вартості доставки читачу повідомлень, яка залежить від відстані до бібліотеки (відповідне відношення придумати самостійно). Дата видачі книги, як окремий клас, характеризується місяцем і числом видачі книги та містить метод підрахунку місяця та числа, що відповідатиме останньому дню повернення книги (книга може бути надана читачу не більше ніж на 10 днів). Описати клас “Читач”, в якому передбачити метод обчислення розміру можливого штрафу за невчасне повернення книги в розмірі 50 % від вартості книги. Використати екземпляр класу “Читач” для проведення операції взяття деякої книги (при умові, що попередня книга була повернута) та виведення сумарної вартості виплат у випадку невчасного повернення книги, яка складається з вартості доставки повідомлення про заборгованість та накладеного штрафу.

Варіант 11.

1. Пакет акцій характеризується: прізвищем власника акцій, підприємством, що випустило акції, кількістю акцій в пакеті, додатковими характеристиками. Підприємство, що випустило акції, як окремий клас, характеризується назвою, статутним капіталом (млр. грн.) та містить метод обчислення рентабельності підприємства (в %), який залежить від розміру статутного капіталу (відповідне відношення придумати самостійно). Додаткові характеристики, як окремий клас, характеризуються номінальною вартістю однієї акції та містить метод обчислення вартості пакету акцій. Описати клас “Пакет акцій”, в якому передбачити метод обчислення можливих річних дивідендів за формулою (вартість пакету акцій * рентабельність / 100%) Використати екземпляр класу “Пакет акцій” для обчислення дивідендів певного пакету акцій.

Варіант 12.

1. Лікарські засоби характеризується: виробником, складом, вартістю випуску. Виробник, як окремий клас, характеризується назвою, континентом розміщення. Склад, як окремий клас, характеризується вагою наявних наркотичних препаратів та містить метод обчислення додаткової вартості, яка залежить від ваги наркотичних препаратів (відповідне відношення придумати самостійно); в країні USA додаткова вартість не встановлюється. Описати клас „Лікарські засоби”, в якому передбачити метод обчислення загальної вартості виробництва лікарського засобу. Використати екземпляр класу „Лікарські засоби” для обчислення роздрібної вартості деякого лікарського засобу, на яку встановлюється 30 % надбавка до його вартості виробництва.

Список використаних джерел

1. Мінухін С.В. Методи і моделі проектування на основі сучасних CASE-засобів. Навчальний посібник / С. В. Мінухін, О. М. Беседовський, С. В. Знахур. — Харків: Вид. ХНЕУ, 2008. — 272 с.

2. Методичні рекомендації до виконання лабораторних робіт з дисципліни «Проектування інформаційних систем» [Електронний ресурс] / [упоряд. Оксамитна Л.П.]; М-во освіти і науки України, Черкас. держ. Технол. ун-т. Черкаси: ЧДТУ, 2021. 99 с.

3. Ушакова І. О. Основи системного аналізу об’єктів та процесів комп’ютеризації : навчальний посібник. Ч. 2 / І. О. Ушакова. – Х. : Вид. ХНЕУ, 2008. – 324 с.
4. Ушакова І. О. Практикум з навчальної дисципліни "Основи системного аналізу об’єктів і процесів комп’ютеризації": навчальний посібник / І. О. Ушакова, Г. О. Плеханова. – Х. : Вид. ХНЕУ, 2010. – 344 с.
5. Каграманова Ю., Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. Режим доступу https://dou.ua/forums/topic/40575/
6. Дідковська М. Проєктування програмного забезпечення засобами UML. Режим доступу http://mmsa.kpi.ua/sites/default/files/disciplines/Розробка%20і%20тестування%20програм/didkovska_m_v_testing_lecture_5.pdf.
7. Цибко Г. Ю., Горошко Ю. В., Костюченко А. О. Програмування у Python. Практичний курс: навчальний посібник. Ч.: ФОП Баликіна С. М., 2022. 180 с.
7. ChatGPT.