Міністерство освіти і науки України
Донбаська державна машинобудівна академія (ДДМА)

Д. О. Картамишев

ТЕХНОЛОГІЯ ПРОГРАМУВАННЯ СКЛАДНИХ СИСТЕМ

Методичні вказівки до виконання курсової роботи з дисципліни «Технологія програмування складних систем»

для здобувачів 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» першого бакалаврського рівня за ОПП «Автоматизація та комп'ютерно-інтегровані технології»

Затверджено
на засіданні методичної ради кафедри АВП
Протокол № 9 від 19.05.2025 р.

Краматорськ-Тернопіль,
ДДМА
2025

УДК 681.3.06 (075.8)

Картамишев Д.О.

Технологія програмування складних систем: Методичні вказівки до виконання курсової роботи з дисципліни «Технологія програмування складних систем» для студентів 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» першого бакалаврського рівня за ОПП «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» / Д. О. Картамишев – Краматорськ : ДДМА, 2025. – 66 с.

Розглядаються структура і склад курсового проєкту. Наведено методику моделювання засобами UML та рекомендації щодо розробки програмної системи з використанням CASE-засобу Visual Paradigm. Демонструється приклад повного циклу розробки програмної системи – від початкової стадії аналізу до етапу генерації програмного коду.

ЗМІСТ

ВСТУП

Курсова робота з дисципліни «Технологія програмування складних систем» виконується з метою набуття вмінь системного сприйняття та вирішення задачі розробки складної програми, освоєння методики об'єктно-орієнтованого моделювання програмних систем мовою UML (Unified Modeling Language), а також отримання навичок роботи з сучасним CASE-засобом Visual Paradigm.

Предметом проектування є програмна система, призначена для автоматизації процесів у певній предметній області.

Курсова робота повинна сформувати наступні програмні результати навчання, що передбачені освітньо-професійною програмою підготовки бакалаврів «Автоматизація та комп'ютерно-інтегровані технології»:

- Вміти застосовувати сучасні інформаційні технології та мати навички розробляти алгоритми та комп’ютерні програми з використанням мов високого рівня та технологій об’єктно-орієнтованого програмування, створювати бази даних та використовувати інтернет-ресурси.

- Вміти проектувати багаторівневі системи керування і збору даних для формування бази параметрів процесу та їх візуалізації за допомогою засобів людино-машинного інтерфейсу, використовуючи новітні комп’ютерно-інтегровані технології

- Оцінювати ризики та здійснювати запобіжні дії їх уникнення, вести професійну діяльність з урахуванням доброчесності та авторського права.

- Усвідомлювати необхідність навчання та саморозвитку продовж усього життя з метою поглиблення знань .

- Вміти оцінювати отримані результати та аргументовано захищати прийняті рішення, дотримуючись принципу неприпустимості корупції та будь-яких інших проявів недоброчесності.

У результаті виконання курсової роботи з навчальної дисципліни «Технологія програмування складних систем» студент повинен продемонструвати достатній рівень сформованості певних результатів навчання через здобуття наступних програмних компетентностей:

- Здатність розв’язувати складні спеціалізовані задачі та практичні проблеми, що характеризуються комплексністю та невизначеністю умов, під час професійної діяльності у галузі автоматизації, або у процесі навчання, що передбачає застосування теорій та методів галузі.

- Здатність застосовувати знання у практичних ситуаціях.

- Здатність до пошуку, опрацювання та аналізу інформації з різних джерел.

- Здатність ухвалювати рішення та діяти, дотримуючись принципу неприпустимості корупції та будь-яких інших проявів недоброчесності.

- Здатність діяти свідомо та соціально-відповідально за результати прийняття стратегічних рішень.

- Здатність до навчання та саморозвитку.

У технології створення програмного продукту використовуються методи, засоби та процедури.

Методи забезпечують аналіз системних і програмних вимог, проектування структури програми та структури даних, генерацію програмного коду та тестування готової програми. Під час розробки проекту слід керуватися об'єктно-орієнтованою методологією, стандартною мовою якої є UML.

Засоби технології програмування – це інструменти, за допомогою яких створюється програма. Під час розробки проекту доцільно застосовувати CASE-технологію (Computer Aided Software Engineering), а з відомих інструментів цієї технології – CASE-засіб Visual Paradigm, що забезпечує інструментальну підтримку UML.

Процедури є "клеєм", що поєднує методи та засоби так, що весь процес проектування перетворюється на послідовність певних операцій, які забезпечують порядок моделювання та документування, а також контроль якості програми.

Процес програмування підпорядковується основній парадигмі технології розробки програм – класичному життєвому циклу. Класичний життєвий цикл становить базову платформу RUP. Він є каскадною моделлю розробки програми, що включає наступні етапи:

1. Системний аналіз, що забезпечує формування загального уявлення про інтерфейс програми з людьми, апаратурою та базами даних.
2. Аналіз вимог до програмної системи, що дозволяє деталізувати функції системи, її характеристики та інтерфейс.
3. Проектування на рівні моделей структури та поведінки системи.
4. Кодування результатів моделювання, тобто створення тексту програми мовою програмування.

З існуючих стратегій розробки програмних систем (водоспадна, інкрементна та еволюційна) для курсового проектування доцільно прийняти водоспадну стратегію – одноразове проходження всіх етапів створення програми. Незважаючи на те, що така стратегія зазвичай реалізує лише частину можливостей системи, для першого самостійного кроку у створенні складної програмної системи цього достатньо. Слід врахувати, що подальше вдосконалення програми можливе в курсовому проектуванні з дисципліни «Цифрові системи управління та обробки інформації», а також під час виконання кваліфікаційної роботи бакалавра.

Зміст курсової роботи має розкривати сутність етапів створення програми для заданої предметної області. Кожен етап роботи над проектом повинен бути описаний за схемою:

– цілі та завдання етапу;
– обґрунтування дій, виконуваних на цьому етапі;
– реалізація дій та результат етапу.

Основна частина розрахунково-пояснювальної записки має повністю розкривати тему та складатися з наступних розділів:

– Вступ.

– Опис і аналіз предметної області.
– Моделювання функціонального призначення системи.
– Розробка діаграм взаємодії об'єктів.
– Об'єктно-орієнтоване моделювання структури та поведінки системи.
– Моделювання реалізації та розгортання програмної системи (за потреби).
– Генерація програмного коду.
– Висновки.

Приблизна тематика курсових робіт наведена у додатку А, зразок титульного аркуша пояснювальної записки представлений у додатку Б.

Результати проектування мають бути представлені у пояснювальній записці на аркушах формату А4 та додатках, що включають UML-діаграми та текст програми. Пояснювальна записка повинна бути оформлена відповідно до ДСТУ 3008-95 «Документація. Звіти у сфері науки і техніки».

Захист проекту здійснюється перед комісією з викладачів кафедри у терміни, встановлені графіком засідань комісії. Доповідь щодо проекту має бути побудована так, щоб були висвітлені головним чином цілі, завдання та мотиви прийняття рішень, а не сам зміст діаграм. Доповідь має бути ілюстрована плакатами або слайдами.

Результати захисту проектів оцінюються за такими критеріями:

– Оцінка A «відмінно» – проекти, виконані згідно із завданням, самостійно, з оригінальними технічними рішеннями.
– Оцінка B «добре» – проекти з незначними недоліками або прогалинами у розробці.
– Оцінка C «задовільно» – проекти з помилковою концепцією системного сприйняття, але правильно застосованою методикою моделювання.
– Оцінка D «задовільно» – проекти з недостатньою розробкою, помилками у методиці моделювання та слабо аргументованими відповідями.
– Оцінка E «задовільно» – проекти з суттєвими недоліками, слабкою розробкою теми та невпевненими відповідями.

1 ОПИС ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Проектування починається з фази досліджень. На першому етапі необхідно провести ретельний аналіз предметної області для формулювання вимог до програмної системи.

Завданням етапу аналізу предметної області є складання опису в контексті «Що повинна робити система?», що підводить до досягнення мети – формулювання вимог.

Рекомендації щодо виконання курсового проєкту будуть супроводжуватися прикладом відомої (в загальних рисах) предметної області – автоматизованого банківського термінала (банкомата).

Тема прикладу: Розробити модель та каркас програмного коду для системи керування банкоматом.

Опис предметної області «Банкомат»

Банкомат – це автоматичний банківський термінал (АТМ) для видачі готівки за кредитними пластиковими картками. До його складу входять такі пристрої: екран, панель керування з кнопками (клавіатура), приймач карток із пристроєм зчитування, пристрій видачі готівки, пристрій блокування карток, принтер для друку чеків (довідок про залишок коштів на рахунку). Банкомат підключено через локальну мережу до сервера (контролера) банку, де зберігаються дані про рахунки клієнтів.

Обслуговування клієнта починається з моменту введення пластикової картки у приймач карток банкомата. Після розпізнавання типу картки банкомат виводить на дисплей повідомлення з проханням ввести персональний код. Після введення ПІН-коду банкомат перевіряє його правильність. Якщо код введено неправильно, користувачу надається ще дві спроби для введення правильного коду. У разі повторної невдачі картка блокується та переміщується у сховище карток, а сеанс обслуговування завершується. Після введення правильного коду банкомат пропонує клієнту вибрати операцію. Клієнт може або зняти готівку з рахунку, або перевірити залишок коштів на рахунку.

Під час зняття готівки клієнт банкомата вводить суму, після чого банкомат запитує, чи потрібно друкувати чек. Потім банкомат надсилає запит на зняття вибраної суми до сервера банку. У разі отримання дозволу банкомат перевіряє наявність необхідної суми в касеті, вилучає картку з приймача та видає вказану суму у лоток видачі. Якщо клієнт обрав друк чека, банкомат друкує довідку про виконану операцію.

Якщо клієнт хоче дізнатися залишок на рахунку, банкомат надсилає запит до сервера банку та виводить суму на дисплей.

Аналіз предметної області

Перед розробником програмної системи завжди постає одне з принципових питань: “Які об’єкти реального світу необхідно врахувати в моделі та як вони взаємопов’язані?”.

Проектуючи об’єктно-орієнтовану систему, врешті-решт потрібно створити модель абстракцій предметної області, тобто отримати уявлення

про структуру системи та її поведінку. Починати створення цієї моделі слід із аналізу опису предметної області.

Іменні групи та іменники в наведеному описі можуть стати об'єктами або атрибутами класів.

Склавши список можливих об'єктів, слід проаналізувати його для виключення зайвих класів.

Зі списку необхідно видалити:

– *Надлишкові назви*, які передають однакову інформацію (залишити лише одну з назв);

– *Нерелевантні терміни*, що не мають безпосереднього відношення до проблеми і представлення яких у майбутній системі є сумнівним;

– Нечітко визначені вирази, які не мають конкретного зв'язку з розглянутою проблемою;

– *Атрибути*: деякі іменники більше відповідають атрибутам класів (наприклад, ім'я, вік, вага, адреса тощо), а не самим класам;

– *Операції*: деякі іменники більше відповідають операціям, а не класам (наприклад, “телефонний виклик” навряд чи є класом);

– *Ролі*: деякі іменники визначають ролі в об'єктній моделі (наприклад, власник, водій, керівник, співробітник);

– *Реалізаційні конструкції*: назви, які більше пов’язані з програмуванням або апаратним забезпеченням (підпрограма, процес, алгоритм, переривання тощо), не слід на цьому етапі пов'язувати з класами, оскільки вони не відображають особливостей прикладної системи, що проектується.

Після видалення всіх зайвих (надлишкових) імен буде отримано попередній список об'єктів (класів), які становлять основу проєктованої програмної системи.

Застосування до опису предметної області «Банкомат»

У попередній список об'єктів, з яких можливе створення класів, можна включити такі елементи:

1. Екранна форма банкомата.
2. Панель керування (клавіатура) банкомата.
3. Приймач карток із пристроями зчитування та блокування.
4. Пристрій видачі готівки.
5. Принтер для друку чеків.
6. Контролер банкомата.
7. Інтерфейс сервера банку.
8. Транзакція банкомата, яка забезпечує зв’язок із зовнішньою системою – банком і непрямо присутня в описі.

Також до списку можна додати абстрактний клас **“Контролер”**, який дозволить визначити системні атрибути, зв’язки та операції.

Дотримуючись методики “рух із середини назовні”, що застосовується для побудови статичної моделі, визначаємо об’єкти, з якими взаємодіє система банкомата. З опису банкомата можна виділити зовнішні

об'єкти – **Клієнт банкомата** та **Банк**. Вони є акторами по відношенню до системи банкомата.

Визначення структури системи

Після визначення структури системи слід перейти до аналізу другого принципового питання: **“Що повинна робити система?”**

Щоб визначити поведінку системи, необхідно виділити прецеденти, наявні в описі, та провести граматичний аналіз тексту кожного прецеденту.

Текст прецеденту повинен відповідати очікуваній поведінці системи та формулюватися з точки зору діючої особи, використовуючи дієслова теперішнього часу в дійсному способі.

Застосовуючи це до розглянутої задачі, клієнт може або **зняти готівку**, або **перевірити залишок коштів на рахунку**. З цього опису формулюються тексти двох прецедентів:

1. **Зняття готівки.**
2. **Перевірка рахунку.**

Для реалізації кожного з цих двох прецедентів знадобиться відповідна поведінка клієнта та системи банкомата. У цю поведінку повинні бути включені процедури ідентифікації параметрів картки, перевірка ПІН-коду, друк чека (на вимогу клієнта), видача готівки (клієнт повинен ввести суму) та інші дії.

Опис поведінки має представляти собою **специфікацію потоків подій**, яка необхідна для побудови моделі поведінки з використанням діаграм варіантів використання (**use case diagram**) та діаграм послідовностей (**sequence diagram**).

Специфікація потоку подій являє собою текстовий сценарій, складений на основі шаблону. Приклад сценарію для прецеденту “Зняття готівки” наведено в таблиці 1.1.

Таблиця 1.1 – Дії акторів та відгук системи

Дії акторів	Відгук системи
1. Клієнт вводить кредитну картку у пристрій читання банкомату. **Виняток 1.** Кредитна картка недійсна.	1. Банкомат перевіряє кредитну картку. 2. Банкомат пропонує запровадити ПІН-код.
2. Клієнт запроваджує ПІН-код. **Виняток 2**. ПІН-код неправильний.	3. Банкомат перевіряє ПІН-код. 4. Банкомат відображає опцію меню.
3. Клієнт вибирає опцію "Зняти готівку".	5. Банкомат пропонує запровадити необхідну суму.
4. Клієнт вводить необхідну суму. **Виняток 3.** Необхідна сума перевищує поточний стан рахунку.	6. Система робить запит до банку та з'ясовує поточний стан рахунки. 7. Банкомат змінює стан рахунки, видає картку, готівкою та чек.

Завершуючи аналіз предметної області, слід переконатися, що потрібна поведінка може бути забезпечена структурою, представленою попереднім переліком об'єктів системи.

2 МОДЕЛЮВАННЯ ФУНКЦІОНАЛЬНОГО ПРИЗНАЧЕННЯ СИСТЕМИ

Рекомендації щодо створення діаграм варіантів використання.

Діаграма варіантів використання відображає формалізацію функціональних вимог до поведінки проєктованої програмної системи. Кожен варіант використання має представляти завершену транзакцію між користувачем і системою.

Семантика побудови діаграми варіантів використання повинна відповідати таким правилам:

1. Екземпляр окремого варіанта використання ініціалізується (запускається) повідомленням від екземпляра актора. У відповідь варіант використання виконує послідовність дій, встановлених для цього сценарію, які продовжуються до того моменту, коли відповідний актор отримає необхідний сервіс.

2. Варіанти використання можуть бути представлені у вигляді тексту, що містить передумови та результати (передумови та постумови). Взаємодія між варіантами використання та акторами може бути деталізована за допомогою діаграм взаємодії.

3. З точки зору системного аналізу, діаграми варіантів використання мають не тільки специфікувати функціональні вимоги до поведінки системи, що проєктується, але й виконувати первинну структуризацію предметної області. Для цього на діаграмі варіантів використання мають бути виділені основні функції системи.

Створення діаграм варіантів використання у Visual Paradigm

Діаграма варіантів використання (use case) дозволяє візуалізувати ролі користувачів та їхню взаємодію з системою. Вона демонструє, які функції може виконувати система з точки зору користувача, але не відображає послідовність кроків. Така діаграма може бути представлена у текстовій формі або графічно.

Основними елементами діаграми є чотири об’єкти:

1. **Актор**
2. **Прецедент (варіант використання)**
3. **Система**
4. **Зв’язок**

Актор — це особа, система або пристрій, що взаємодіє із системою, але не є її частиною. У графічному вигляді актор зображається як стилізована фігура людини або прямокутник з написом <<actor>>.

Актори поділяються на:

– **Первинних** — ініціюють взаємодію з системою (зазвичай розташовані зліва на діаграмі).

– **Вторинних** — реагують на дії системи (розміщуються праворуч).

– Актор не обов'язково представляє людину. Це може бути інша система або пристрій, з яким взаємодіє основна система.

Важливо розрізняти поняття актора та користувача:

– **Актор** — це узагальнений клас користувачів, який може охоплювати різні ролі. Наприклад, актором може бути співробітник компанії, що виконує ролі інженера, менеджера або директора.

– **Користувач** — конкретна реалізація актора. Декілька користувачів можуть представляти одного актора, виконуючи схожі функції. Наприклад, студенти Іван та Роман можуть бути одним актором — «Студент». Водночас один користувач може належати до кількох акторів, виконуючи різні ролі, наприклад, викладача або наукового керівника в університетській системі.

Цей підхід дозволяє структурувати функціональні вимоги та зрозуміти, як система має взаємодіяти з різними зовнішніми учасниками (рис. 2.1).

Випадок використання (прецедент)

Прецеденти описують очікувану поведінку системи, відповідаючи на питання, що саме вона робить. Вони представляють набір функцій, дій або завдань, які може виконувати система. Графічно прецедент зображується у вигляді еліпса з назвою дії (зазвичай у формі дієслова) всередині.

Прецедент показує, що має відбутись, але не деталізує, яким чином це буде реалізовано. Наприклад, у банківському додатку прецедентами можуть бути:

– Перевірка балансу
– Переказ коштів
– Оплата рахунків

Система

Система – це об'єкт моделювання, який може бути вебсайтом, мобільним додатком або модулем програмного забезпечення. На діаграмі система зображується у вигляді прямокутника з її назвою у верхній частині.

Коментар

Коментарі розширюють функціональність діаграми, надаючи додаткові пояснення, підказки або умови виконання дій. Вони допомагають уточнити важливі аспекти прецедентів та акторів.

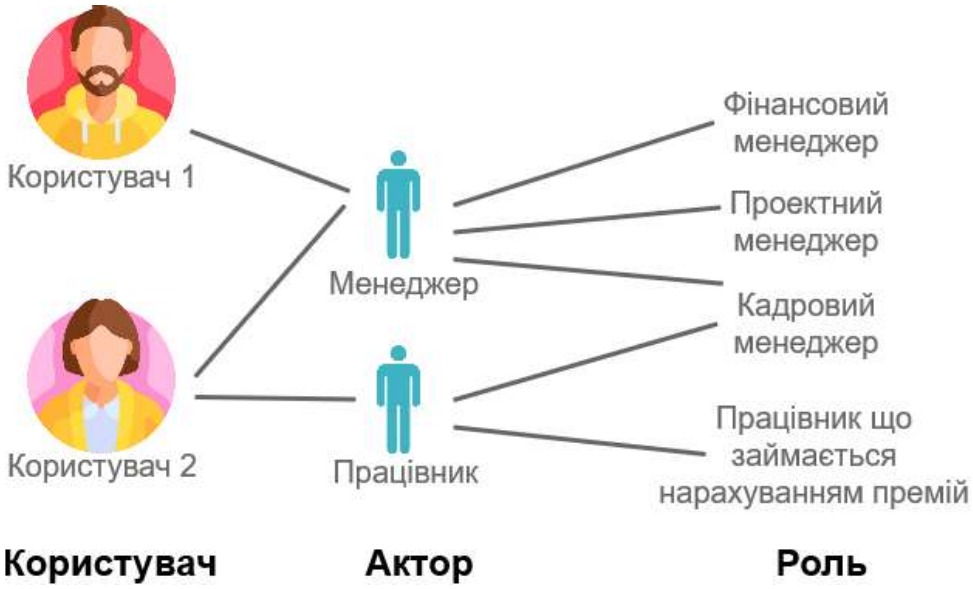


Рисунок 2.1 – Приклад користувача, актора та ролі в UML

Усі складові діаграми прецедентів відображені на рисунку 2.2.

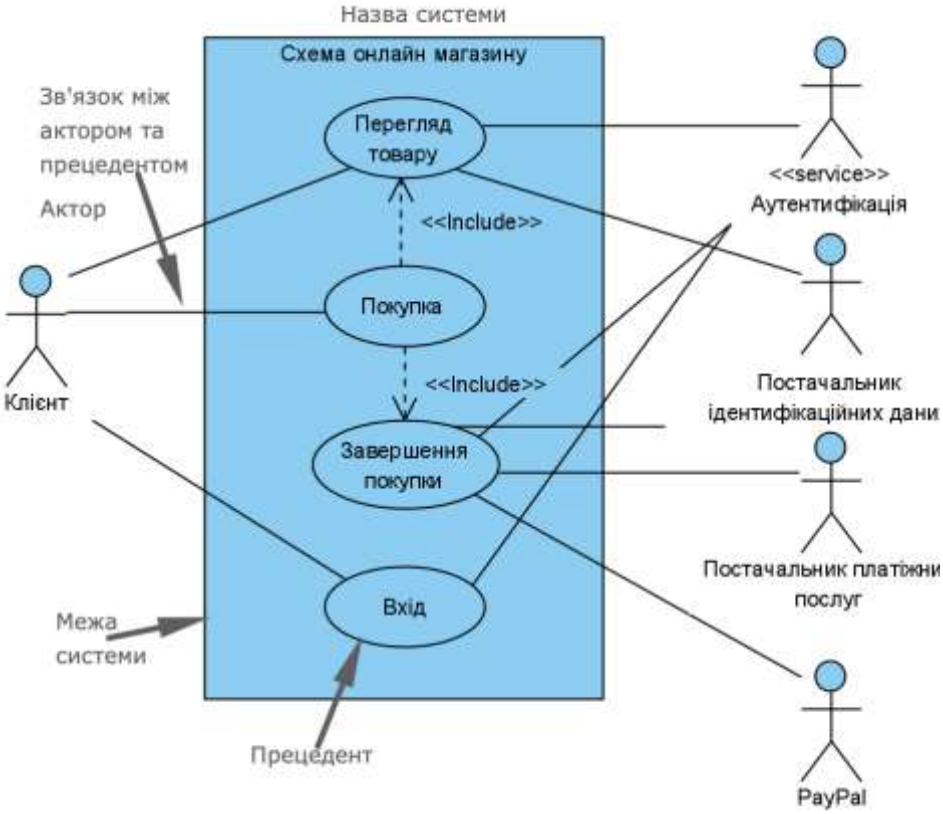


Рисунок 2.2 – Складові діаграми прецедентів

Зв’язки (відношення)

Усі актори на діаграмі повинні мати зв’язок із хоча б одним прецедентом, тоді як не всі прецеденти обов’язково мають бути пов’язані з

акторами. Для позначення зв’язків найчастіше використовують суцільні лінії.

У діаграмах варіантів використання існує чотири основні типи зв’язків:

1. **Асоціація (Association)** – це базовий зв’язок між актором і прецедентом. Він позначається простою суцільною лінією без додаткових позначок чи стереотипів.

Незалежно від типу зв’язку, кожен актор повинен бути пов’язаний щонайменше з одним варіантом використання. Один актор може взаємодіяти з кількома прецедентами, так само як і кілька акторів можуть бути пов’язані з одним варіантом використання.

Цей тип зв’язку допомагає візуалізувати, хто саме ініціює або виконує певні функції в системі (рис. 2.3).

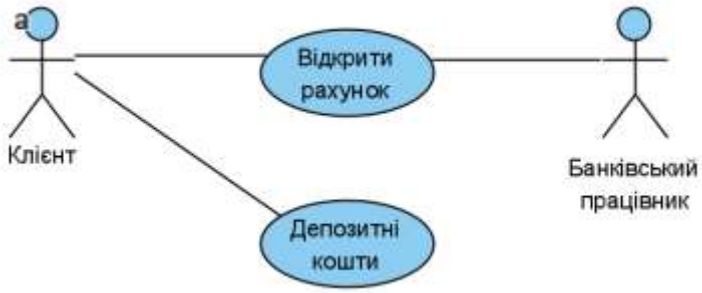


Рисунок 2.3 – Асоціація між актором та варіантом використання

2. Розширення (Extend)

Зв’язок типу **«Розширення»** позначає додаткову функціональність або необов’язкову поведінку системи. Базовий прецедент є самостійним і може функціонувати незалежно від прецеденту-розширення.

Відношення розширення позначається пунктирною лінією зі стрілкою, що вказує на базовий прецедент. Над лінією розміщується стереотип (напис) **«<<extend>>»**. Прецедент-розширення активується лише за певних умов.

Приклад:

Прецедент **«Хибний пароль»** спрацьовує тільки тоді, коли користувач вводить невірний пароль. Цей зв’язок відповідає секції **extensions** у текстовому описі сценарію використання.

Зв’язок **Extend** дозволяє моделювати варіанти поведінки системи, які не є обов’язковими, але можуть розширювати її можливості в залежності від конкретних ситуацій.

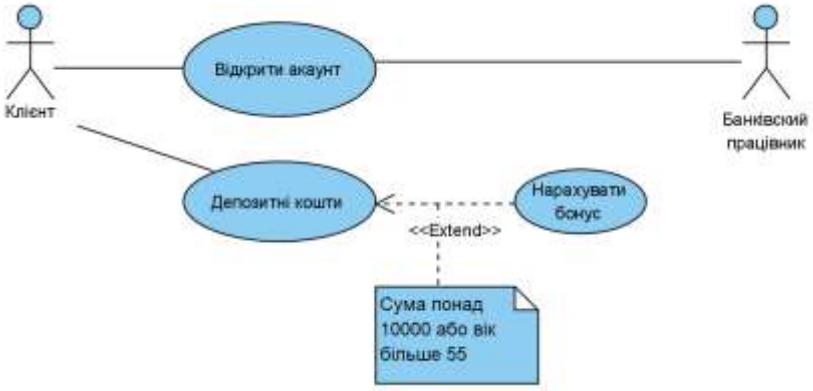


Рисунок 2.4 – Приклад зв'язку розширення з умовою

Точки розширення (extension points)

При використанні зв'язку **«Розширення»** базовий прецедент містить спеціальні точки розширення. Це місця, де може бути активований прецедент-розширення. У текстовому описі такі точки відповідають секції **extensions**.

За необхідності точки розширення можна деталізувати в окремому розділі базового прецеденту, що допомагає чітко визначити умови та місця їхнього виклику.

Точки розширення дозволяють створювати гнучкі діаграми, які демонструють альтернативну або додаткову поведінку системи без порушення основного сценарію (рис. 2.5).

Рисунок 2.5 – Приклад зв'язку розширення з точками розширення

Умови розширення (conditions)

Якщо для прецеденту-розширення існує певна умова, її можна описати у вигляді **коментаря**. У текстовому варіанті ця умова відповідає секції **preconditions**.

Коментар з умовою розширення з'єднується з відповідним прецедентом за допомогою пунктирної лінії. Це дозволяє чітко вказати, за яких обставин активується прецедент-розширення, роблячи діаграму більш зрозумілою та деталізованою.

Такий підхід допомагає відобразити логіку роботи системи, де певні дії виконуються лише за наявності специфічних умов (рис. 2.6).

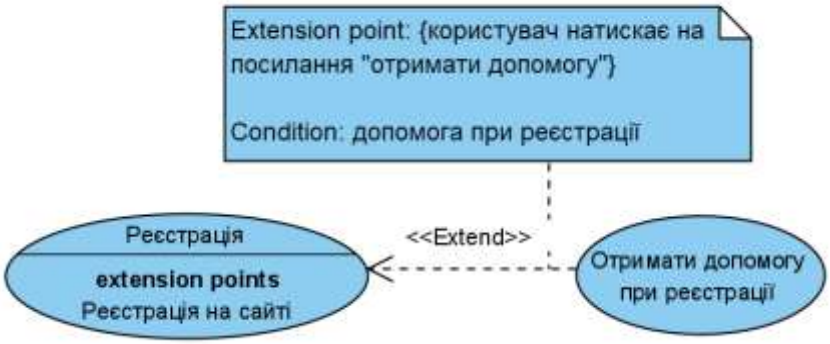


Рисунок 2.6 – Приклад використання коментаря

3. Включення (Include)

Зв'язок **«Включення»** показує, що поведінка одного прецеденту є складовою частиною іншого прецеденту. Це ілюструє, які саме дії базовий прецедент використовує для виконання операції.

На відміну від зв'язку **«Розширення»**, включений прецедент є обов'язковим для виконання базового сценарію. Зв'язок **Include** дозволяє уникнути дублювання однакових прецедентів, додаючи функціональність, яка є критичною для базового прецеденту.

Відношення включення позначається пунктирною лінією зі стрілкою, спрямованою на включений прецедент, та стереотипом **«<<include>>»**.

Приклад:

У сценарії відновлення працездатності комп'ютера прецедент «Відновлення системи» може включати наступні дії:

- Ремонт або заміна апаратних компонентів;
- Виявлення та видалення вірусів;
- Перевстановлення операційної системи.

Зв'язок **Include** демонструє, що ці кроки є обов'язковими для повного виконання основного сценарію (рис 2.7).

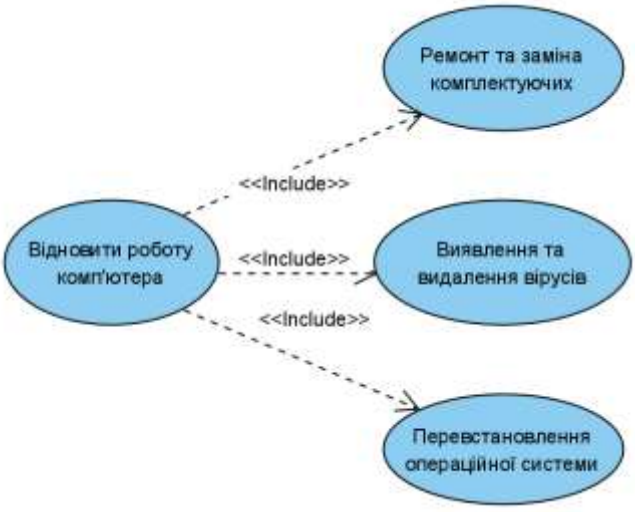


Рисунок 2.7 – Включення між варіантами використання

На рисунку 2.8 зображено випадки використання для банківської системи, де клієнт та банківський працівник взаємодіють із системою.

Опис зв’язків include:

- **Депозитні кошти** включають в себе прецеденти:
  - **Зняти кошти**
  - **Оновити баланс**

Це означає, що під час виконання операції з депозитом система обов’язково виконує дії з оновлення балансу та можливого зняття коштів. Прецеденти «Зняти кошти» та «Оновити баланс» є необхідною частиною для повного виконання базового прецеденту «Депозитні кошти».

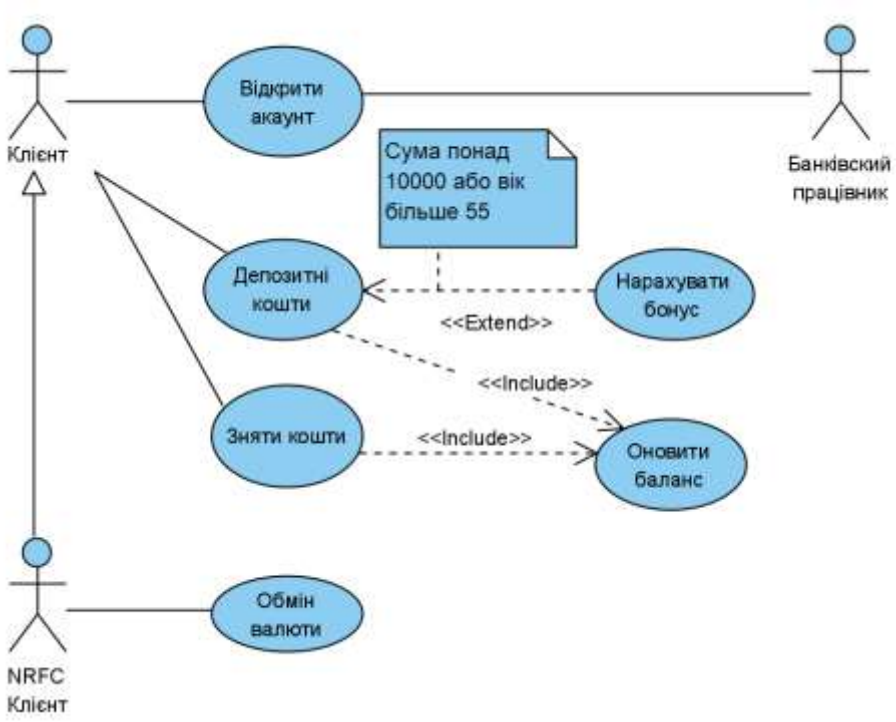


Рисунок 2.8 – Включення дочірнього прецеденту «оновити баланс» до декількох прецедентів-предків

4. Генералізація (Generalization)

Генералізація (узагальнення або успадкування) — це відношення між батьківським та дочірніми прецедентами. Воно показує, що дочірній прецедент успадковує всі властивості батьківського прецеденту, але може також мати власні унікальні риси.

Позначення:

Генералізація зображується у вигляді суцільної лінії з трикутним, незаповненим стрілоподібним вказівником, що спрямований на прецедент-предок.

Основні властивості генералізації:

– Дочірні прецеденти успадковують усі характеристики батьківського прецеденту.

– Один батьківський прецедент може мати кілька дочірніх прецедентів.

– У деяких випадках можливо множинне успадкування, коли дочірній прецедент має кілька батьків.

Генералізація актора:

Відношення генералізації може бути застосоване і до акторів. У такому випадку один актор (нащадок) успадковує ролі іншого актора (предка). Це означає, що нащадок має доступ до всіх прецедентів предка, але може також мати власні додаткові варіанти використання.

Приклад:

Актор «Менеджер» може бути узагальнений у таких акторів, як «Фінансовий менеджер» або «Проєктний менеджер», які матимуть як спільні функції, так і свої специфічні прецеденти (рис. 2.9).

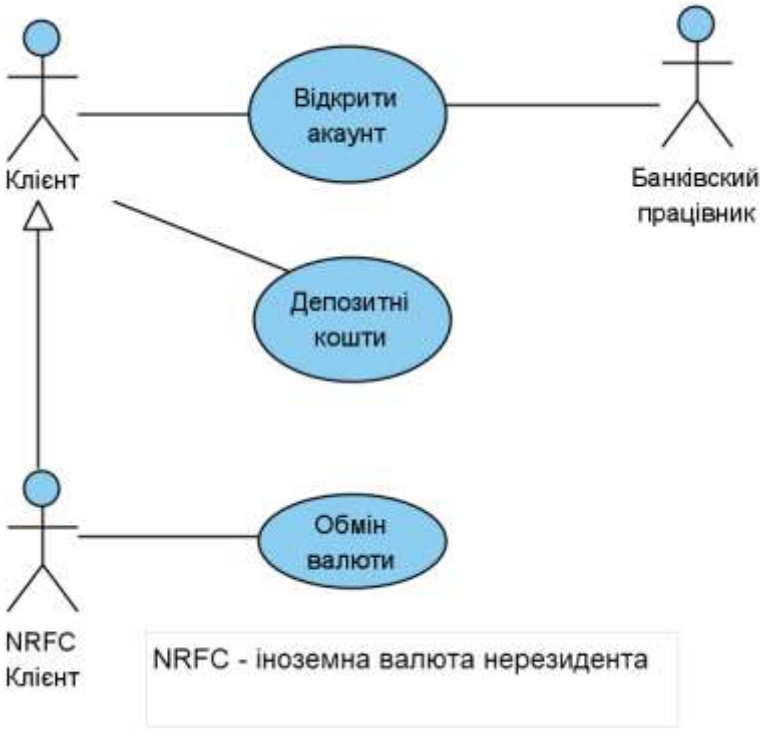


Рисунок 2.9 – Узагальнення актора

Узагальнення варіанту використання

Узагальнення варіантів використання працює за тим же принципом, що й узагальнення актора. У цьому випадку дочірній варіант використання успадковує поведінку від батьківського (предка).

Застосування:

Використовується, коли кілька варіантів використання мають спільну поведінку або логіку. Це дозволяє створити базовий варіант використання,

який об’єднує загальні функції, з можливістю уточнення чи розширення в дочірніх варіантах.

Приклад:

Прецеденти **«Оплата банківською карткою»** та **«Оплата через Google Pay»** можуть бути узагальнені в батьківський варіант **«Оплата рахунку»**. Усі специфічні варіанти оплати успадковують основну логіку процесу, але можуть мати відмінні деталі, залежно від обраного способу оплати.

Такий підхід дозволяє зменшити дублювання та спрощує управління сценаріями, де присутня спільна поведінка.

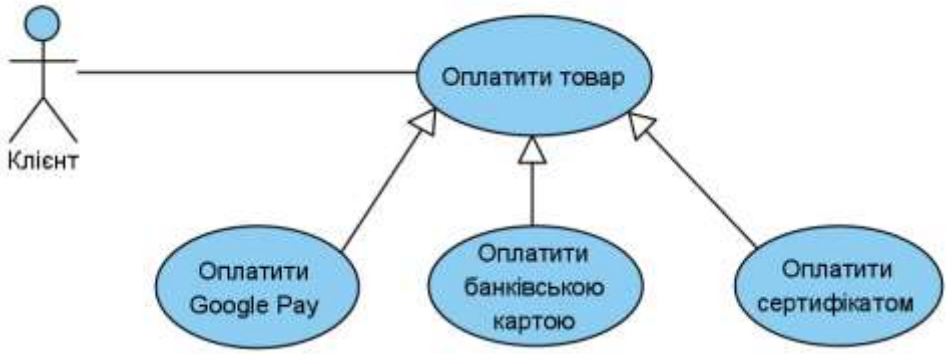


Рисунок 2.10 – Узагальнення варіанту використання

3 РЕКОМЕНДАЦІЇ З РОЗРОБКИ ДІАГРАМ КЛАСІВ

Діаграма класів представляє логічне уявлення програмної системи, описуючи її архітектуру та структуру.

Клас (class) в UML використовується для позначення групи об'єктів, які мають спільну структуру, поведінку та взаємодіють з об'єктами інших класів.

Графічне зображення класу:

Клас представлений прямокутником, розділеним на горизонтальні секції:

- **Назва класу** (обов'язкова секція);
- **Атрибути** (властивості);
- **Операції** (методи).

На початкових етапах розробки діаграми класів прямокутник може містити лише ім'я класу. З часом він доповнюється атрибутами та методами у міру деталізації системи.

Атрибути (attributes):

Атрибути класу записуються у середній секції прямокутника, кожен з них займає окремий рядок. Запис атрибуту містить:

- Квантор видимості (модифікатор доступу);
- Ім'я атрибуту;
- Тип даних;
- Кратність (при необхідності);
- Значення за замовчуванням.

Квантори видимості:

Квантор видимості визначає рівень доступу до атрибута та може набувати наступних значень:

- **Public (+)** – атрибут доступний з будь-якого класу системи.
- **Protected (#)** – атрибут доступний лише класам-нащадкам.
- **Private (-)** – атрибут недоступний для інших класів.
- **Package (~)** – атрибут доступний тільки в межах класів одного пакета.

У різних CASE-засобах (наприклад, Visual Paradigm) можуть використовуватися різні позначення для кванторів видимості, але їхній функціонал залишається незмінним.

Операції класу – це функції або методи, які забезпечують обробку даних і виконання певних дій над об'єктами класу. Як і атрибути, операції мають:

- Унікальні імена;
- Квантори видимості (public, protected, private, package).

Приклад:

На прикладі з Visual Paradigm створено клас **User** з такими атрибутами (рис. 3.1):

- **LastName** та **FirstName** – типу *public*;
- **Age** – типу *protected*;

– **Status** – типу *private*.
– Клас також містить наступні операції:
– **GetStatus()** – типу *protected*;
– **UpdateStatus()** – типу *private*;
– **PrintStatus()** – типу *package*.

Види зв'язків між класами в UML

Основні типи зв'язків між класами:

1. **Залежність (Dependency Relationship)**
2. **Асоціація (Association Relationship)**
3. **Узагальнення (Generalization Relationship)**
4. **Агрегація (Aggregation Relationship)**

Кожен із цих типів зв'язків має унікальне графічне позначення на діаграмі класів.

Цей тип зв'язку вказує на семантичну залежність між класами, що означає використання або взаємодію одного класу з іншим. Проте цей зв'язок не належить до таких типів, як асоціація, узагальнення чи агрегація.

Відношення залежності свідчить про те, що зміни в одному класі можуть вплинути на інший, але це не обов'язково вказує на постійний або тісний взаємозв'язок між ними.

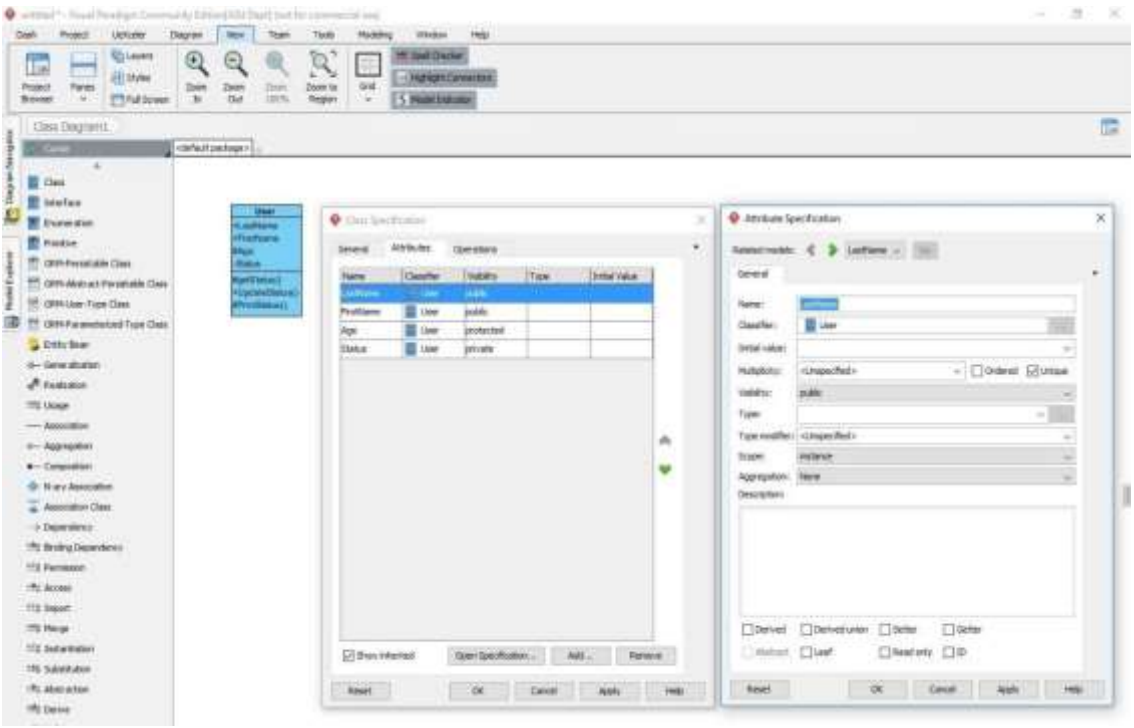


Рисунок 3.1 – Інтерфейс Visual Paradigm для визначення властивостей класу

Відношення залежності застосовується в ситуаціях, коли зміна одного класу може вимагати внесення змін до параметрів (атрибутів або операцій) іншого класу, який залежить від нього. На діаграмі це відображається у

вигляді пунктирної лінії зі стрілкою на одному з її кінців, спрямованою від залежного класу або клієнта (client) до незалежного класу, який виступає постачальником (supplier). На рис. 3.2 зображено приклад двох класів: клас *User Profile* є залежним клієнтом, тоді як клас *Application* виступає постачальником у цій залежності.

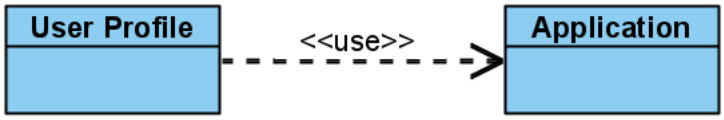


Рисунок 3.2 – Приклад відношення залежності

Відношення **асоціації** відображає наявність структурної залежності між класами. На діаграмі воно позначається суцільною лінією, доповненою спеціальними символами, які вказують на певні характеристики асоціації, зокрема її назву, кратність і напрямок. Приклад такої асоціації представлений на рис. 3.3, де показано зв'язок між класами *User* і *Account*. Ці класи поєднані бінарною асоціацією з назвою *subscribe*, яка зазначена поруч із лінією зв'язку.

Рисунок 3.3 – Графічне зображення відношення бінарної асоціації

На діаграмі класів може бути зазначена кратність входження класів у відношення. Вона відображається як інтервал цілих чисел, що записується біля кінця лінії асоціації та показує кількість об'єктів, які можуть брати участь у цьому зв'язку. Основними варіантами кратності є: **ОДИН ДО ОДНОГО (1:1)**, **ОДИН ДО БАГАТЬОХ (1:*)** та **БАГАТО ДО БАГАТЬОХ (*:*)**.

Також можна вказати мінімальне та максимальне значення кратності. Наприклад, позначення **«0...1» – «1...*»** означає, що на лівій стороні асоціації може бути один або жоден екземпляр класу, який залежить від принаймні одного чи декількох екземплярів класу на правій стороні. Це дозволяє точно моделювати кількість об'єктів, які можуть бути пов'язані між собою в рамках певної асоціації.

Відношення **агрегації** описує зв'язок між класами, коли один клас складається з інших класів, які є його складовими частинами. Це відношення відіграє ключову роль у моделюванні структури складних систем, оскільки відображає взаємозв'язки типу «ціле – частина» (part-of). Агрегація дозволяє декомпозувати складну систему на простіші

компоненти, кожен з яких, за необхідності, може бути додатково розбитий на менші частини.

Типовим прикладом агрегації є зв'язок між класом «Автомобіль» та його компонентами, такими як «Двигун», «Шасі» та «Кузов». На діаграмі це відношення позначається суцільною лінією з ромбом на одному кінці, при цьому ромб залишається незаповненим. Ромб вказує на клас-агрегатор, тоді як інші класи представляють його частини.

Крім того, на діаграмі агрегації може бути зазначена кратність входження класів, що дозволяє вказати кількість компонентів, які можуть бути частиною агрегованого класу (рис. 3.4).

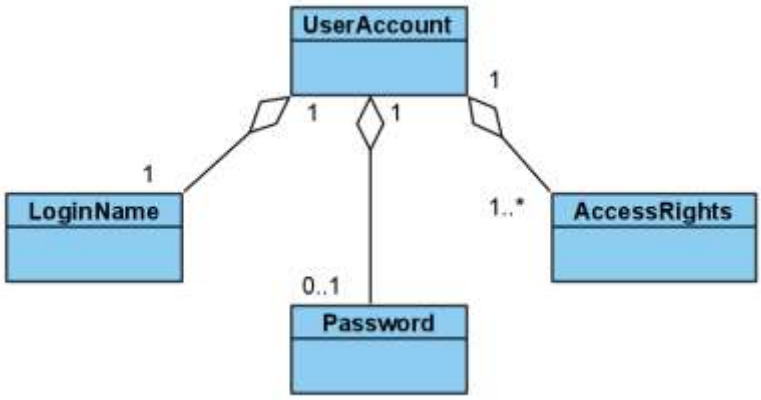


Рисунок 3.4 – Приклад діаграми класів для відношення агрегації

Композиція є окремим випадком агрегації, що використовується для моделювання класів, коли компоненти невід'ємно належать до цілого. Відмінність композиції полягає в тому, що частини не можуть існувати окремо від цілого – знищення основного класу тягне за собою знищення всіх його складових.

Прикладом композиції є вікно користувацького інтерфейсу програми, яке включає в себе заголовок, кнопки управління, смуги прокручування та робочу область. У цьому випадку вікно розглядається як основний клас, а його складові є підкласами або атрибутами, що логічно належать до нього. Графічно композиція позначається суцільною лінією з зафарбованим ромбом на кінці, який вказує на клас-композицію (рис. 3.5).

Узагальнення, своєю чергою, описує відношення між загальним класом (предком або суперкласом) та його більш специфічними підкласами-нащадками. Це відношення дозволяє візуалізувати ієрархію класів на діаграмі, демонструючи спадкування властивостей та поведінки. Клас-нащадок успадковує всі характеристики предка, але може також мати додаткові атрибути та методи, які відсутні в базовому класі. Узагальнення використовується для зв'язків між класами, пакетами або варіантами використання, що дозволяє чітко представити ієрархічну структуру системи.

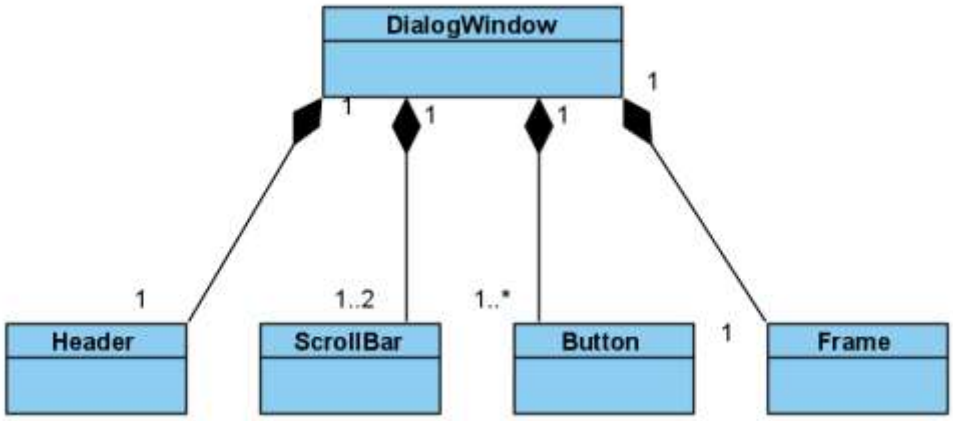


Рисунок 3.5 – Приклад діаграми класів для відношення композиції

На діаграмах узагальнення відображається у вигляді суцільної лінії, що завершується трикутною стрілкою. Ця стрілка спрямована на клас-предок або суперклас, який є загальним для кількох підкласів. Якщо стрілка відсутня, це означає, що клас є більш спеціалізованим, тобто нащадком або підкласом.

Таке позначення дозволяє чітко відобразити ієрархію та спадкування між класами, де підклас отримує властивості та методи свого предка, зберігаючи можливість додавання власних унікальних характеристик. Узагальнення спрощує представлення спільних рис між класами, сприяючи більш ефективному моделюванню складних систем (рис. 3.6).

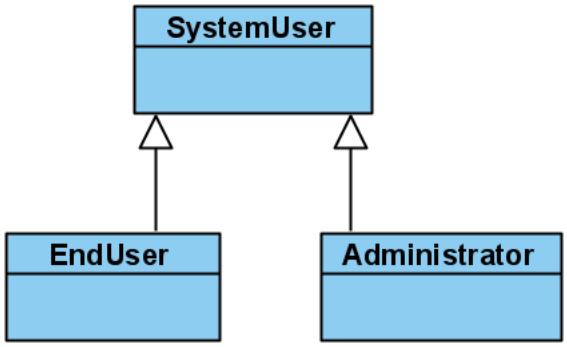


Рисунок 3.6 – Приклад діаграми класів для відношення узагальнення

4 РЕКОМЕНДАЦІЇ ЩОДО РОЗРОБКИ ЛОГІЧНОГО РІВНЯ ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

Після створення діаграми класів, яка відображає статичні аспекти програмної системи (ПС), для моделювання її динамічних характеристик використовуються інші типи діаграм UML. Вони дозволяють представити взаємодію компонентів і описати їхню поведінку в різних сценаріях роботи системи. Серед основних динамічних діаграм UML виділяють:

- **Діаграми кінцевого автомату (state machine diagram);**
- **Діаграми діяльності (activity diagram);**
- **Діаграми комунікації (communication diagram);**
- **Діаграми композитної структури (composite structure diagram);**
- **Діаграми обзору взаємодії (interaction overview diagram).**

Діаграма кінцевого автомата моделює зміну станів об’єкта певного класу протягом його життєвого циклу. Вона описує всі можливі стани, в яких може перебувати об’єкт, і умови переходу між ними. Стан об’єкта може змінюватися під впливом зовнішніх дій інших об’єктів або систем. Основна мета такої діаграми – відобразити послідовність станів і переходів, що характеризують поведінку елементів UML-моделі протягом їхнього існування.

У середовищі **Visual Paradigm** для створення діаграми кінцевого автомата необхідно обрати вкладку **UML** і вибрати пункт **State Machine Diagram** (рис. 4.1).

Стан (state) об’єкта відображає набір конкретних значень його атрибутів, причому зміна хоча б одного з них вказує на зміну стану.

На діаграмі стан представлений прямокутником із заокругленими кутами. У деяких випадках цей прямокутник розділяється горизонтальною лінією на дві секції. Якщо секція лише одна, вона містить назву стану. Якщо секцій дві, у верхній частині вказується назва стану, а в нижній – список внутрішніх дій, що виконуються в цьому стані. Назва стану є текстовим рядком, який чітко відображає суть даного стану, і завжди починається з великої літери.

У переліку внутрішніх дій зазначаються всі операції, що виконуються під час перебування об’єкта у відповідному стані. Кожна дія записується окремим рядком за форматом: «позначка дії/опис дії».

Позначка дії вказує на умови або обставини, за яких виконується дія. Перелік таких позначок є стандартним і містить фіксовані значення, що полегшує інтерпретацію діаграми.

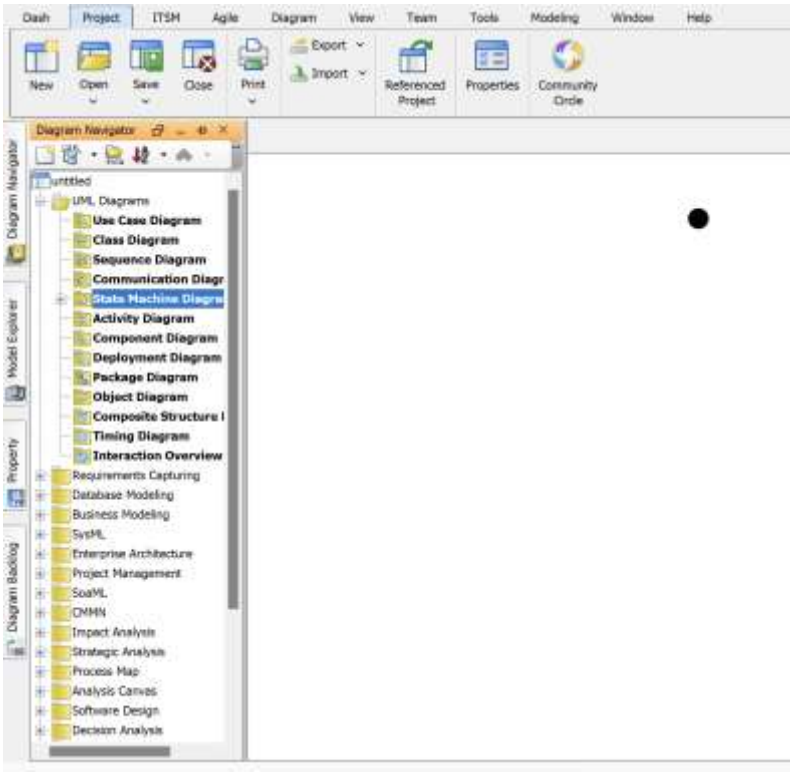


Рисунок 4.1 – Інтерфейс у середовищі Visual Paradigm при побудові діаграми кінцевого автомата

Позначки дій у діаграмі кінцевого автомата визначають специфіку операцій, які виконуються на різних етапах життєвого циклу об’єкта.

– **entry** – ця позначка вказує на дію, що виконується під час входу в стан (вхідна дія).

– **exit** – визначає дію, яка відбувається в момент виходу зі стану (вихідна дія).

– **do** – позначає діяльність, яка триває протягом усього часу перебування об’єкта в даному стані, доки не завершиться відповідне обчислення або дія.

– **event** – вказує на дію, яка виконується під час виникнення певної події у програмній системі, наприклад, натискання клавіші користувачем.

Початковий стан є особливим видом стану, що не містить внутрішніх дій. Він зображується у вигляді зафарбованого кола, з якого виходить стрілка, що вказує на перший перехід об’єкта до наступного стану.

Кінцевий стан також не передбачає внутрішніх дій і графічно зображується як зафарбоване коло, вкладене у незаповнене коло. У кінцевий стан може входити стрілка, яка символізує завершальний перехід системи.

Поняття переходу визначає зміну стану об’єкта та його позначення на діаграмі. Простий перехід (simple transition) описує взаємозв’язок між двома послідовними станами та фіксує факт їх зміни. Під час перебування

об’єкта в початковому стані можуть виконуватись певні дії, а перехід до наступного стає можливим після їх завершення та дотримання необхідних умов.

Перехід активується подією у системі, наприклад, завершенням поточної активності або отриманням повідомлення. Подія позначається на переході. Додатково можуть бути вказані дії, які виконує об’єкт у відповідь на зовнішню подію під час переходу між станами.

Перехід може залежати від **сторожової умови (guard condition)**. У такому випадку зміна стану відбувається лише за умови виконання певної події та набуття сторожовою умовою значення «істина (true)».

Перехід може бути спрямований до того ж стану, з якого він починається. Такий тип переходу називається **переходом в самого себе**. У цьому випадку вихідний і цільовий стани збігаються, що означає, що об’єкт залишає поточний стан і знову входить в нього. Під час кожного такого циклу виконуються внутрішні дії, які визначаються позначками **entry** (дії при вході) та **exit** (дії при виході).

На рис. 4.2 наведено приклад початкового стану програмної системи (ПС) та переходу в стан **System start**, що відповідає відкриттю діалогового вікна або стартової HTML-сторінки на екрані комп’ютера. При вході в цей стан система створює вікно, а при виході – вікно закривається.

Такий механізм дозволяє моделювати циклічну поведінку системи, коли для виконання певних операцій необхідно повторно проходити через один і той самий стан, оновлюючи або змінюючи його внутрішній контекст.

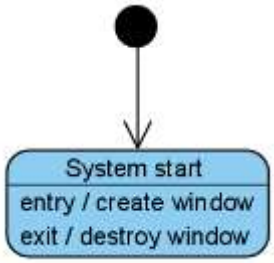


Рисунок 4.2 – Графічне зображення окремого стану з секцією опису внутрішніх дій

На рис. 4.3 зображена діаграма станів, що ілюструє процес авторизації користувача в програмній системі (ПС). Ця діаграма демонструє послідовність станів, через які проходить система під час обробки запиту на вхід.

Процес починається зі стану **«System start»**, де створюється діалогове вікно. Після завершення роботи в цьому стані та виходу з нього вікно закривається.

Із цього стану система може перейти в один із двох наступних станів залежно від виконаної дії:

– Якщо користувач виконує дію **«sign in»**, система переходить у стан **«Login and password input»**. У цьому стані користувач вводить логін (LoginName) та пароль (PW).

– При вході в цей стан виконується ініціалізація полів для введення даних (**entry/input field initialization**), а під час перебування у стані здійснюється введення

символів (**do/get Symbol**). Якщо користувач натискає клавішу **Escape**, вікно закривається (**event Escape button / Close window**).

Зі стану **«Login and password input»** можливий лише один перехід – до стану **«Authorization»**, який активується після виконання дії **«Input data submit»**.

У стані **«Authorization»** встановлюється з'єднання з базою даних, і виконується перевірка введених імені користувача та пароля (**do/check login and password**).

– Якщо авторизація проходить успішно (**[is LoginPasswordCorrect=true]**), система переходить у кінцевий стан.

– У разі невірних даних (**[is LoginPasswordCorrect=false]**) система повертається до стану **«Login and password input»**, дозволяючи користувачеві повторити спробу введення логіну та пароля.

Ця діаграма чітко демонструє логіку перевірки облікових даних, умовні переходи та реакції системи на різні дії користувача, забезпечуючи наочне відображення життєвого циклу процесу авторизації.

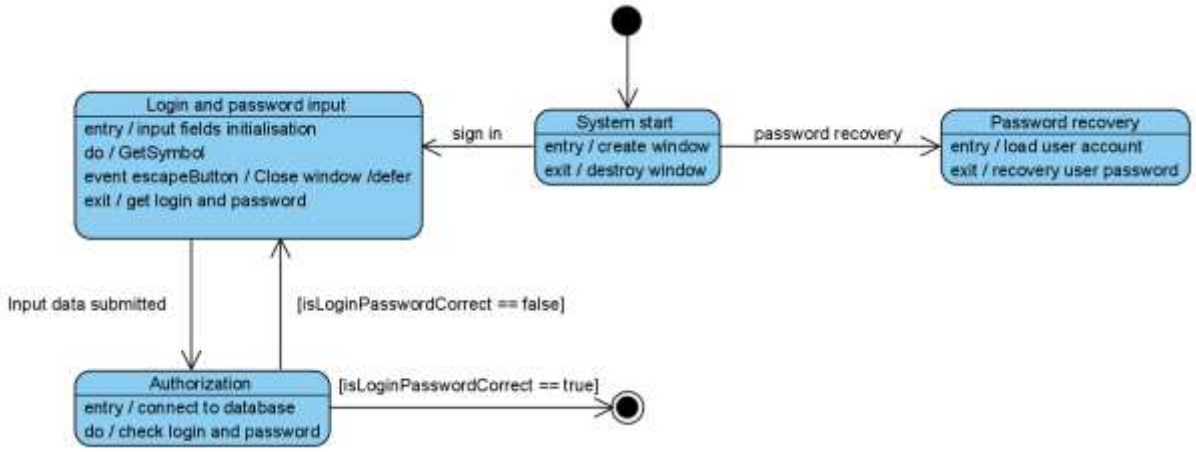


Рисунок 4.3 – Приклад побудови діаграми кінцевого автомата

При виконанні дії **«password recovery»** із стану **«System start»** система переходить у стан **«Password recovery»**. У цьому стані здійснюється пошук облікового запису користувача в базі даних (**entry/load user account**). Під час перебування в цьому стані виконується відновлення пароля користувача (**do/recovery user password**), після чого система повертається до нормального процесу роботи.

Діаграми діяльності (activity diagram)

Для моделювання поведінки програмної системи (ПС) часто необхідно не тільки відображати зміну станів, але й деталізувати алгоритмічну реалізацію операцій, що виконує система. У UML для цих цілей застосовуються **діаграми діяльності**.

Графічна нотація діаграм діяльності схожа з діаграмами станів, оскільки вони також містять позначення станів і переходів. Однак основна відмінність полягає в тому, що стани в діаграмах діяльності відповідають конкретним діям або операціям, які виконуються системою. Перехід до наступного стану відбувається лише після завершення поточної дії.

У середовищі **Visual Paradigm** створення діаграми діяльності виконується аналогічно побудові діаграми кінцевого автомата (рис. 4.1).

Стан дії (action state) є особливим типом стану, який включає початкову дію та принаймні один перехід, що сигналізує про завершення цієї дії. Важливо зазначити, що стан дії не має внутрішніх переходів, оскільки є елементарним етапом виконання процесу або алгоритму.

Графічно стан дії зображується як прямокутник із закругленими боками (рис. 4.4). Всередині цієї фігури вказується назва дії (**action-expression**), яка має бути унікальною в межах діаграми діяльності.

Дії можуть бути записані на природній мові, псевдокоді або мовою програмування.

Кожна діаграма діяльності повинна містити:

– **Початковий стан** – позначається зафарбованим колом.

– **Кінцевий стан** – позначається зафарбованим колом, вкладеним у коло.

Ці позначення аналогічні тим, що використовуються на діаграмах станів (рис. 4.2–4.3).

При створенні діаграми діяльності використовуються переходи, які не залежать від тригерних подій. Такі переходи спрацьовують автоматично після завершення дії у попередньому стані, негайно переводячи систему до наступного стану. На діаграмі ці переходи зображуються суцільними лініями зі стрілками.

На рис. 4.4, а представлено приклад дії **«System start»**, яка виконується одразу після початкового стану без додаткових умов або подій.

Якщо з певного стану виходить лише один перехід, його можна не позначати додатковими умовами – він спрацьовує автоматично.

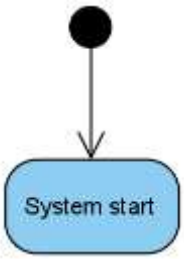
У випадку, коли зі стану виходить декілька переходів, система може обрати лише один з них для виконання. Для цього кожен перехід супроводжується **сторожовою умовою**, записаною в прямих дужках **[ ]**.

Під час виконання такого переходу перевіряється істинність умови, і спрацьовує лише той перехід, для якого вона виконується. Подібні ситуації зустрічаються, коли система має розділити послідовність виконання на кілька альтернативних гілок залежно від проміжного результату обчислень або умов у процесі роботи.

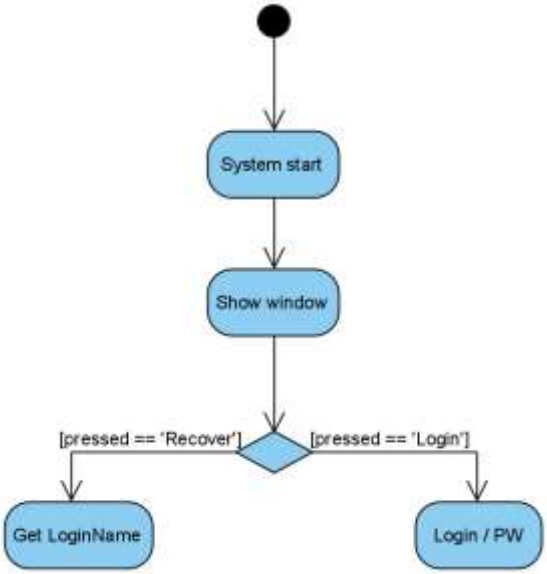
Цей механізм забезпечує гнучкість у моделюванні сценаріїв, дозволяючи детально відображати алгоритми із можливими варіантами розвитку подій.

Така ситуація відома як **альтернативне розгалуження** або просто **розгалуження (decision)**. Для її графічного представлення на діаграмі використовується спеціальний символ – **ромб без тексту всередині** (рис. 4.4, б).

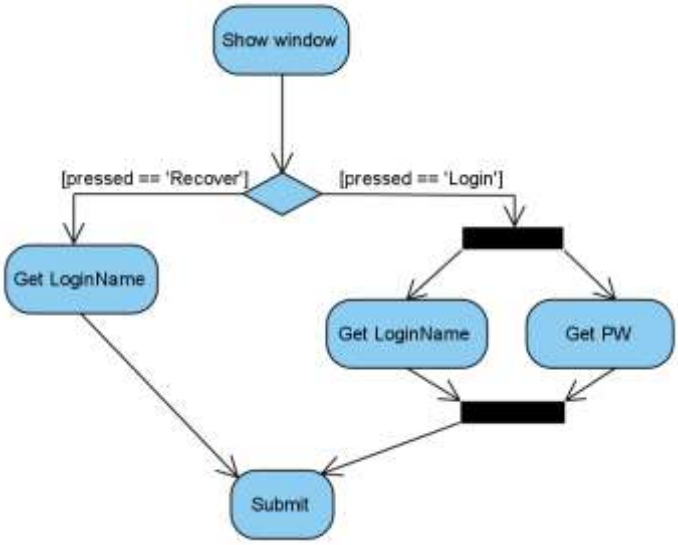
У ромб може входити лише одна стрілка від попереднього стану дії, після виконання якого система має перейти до однієї з кількох гілок, що виключають одна одну. Прийнято підключати вхідну стрілку до **верхньої або лівої** вершини ромба.

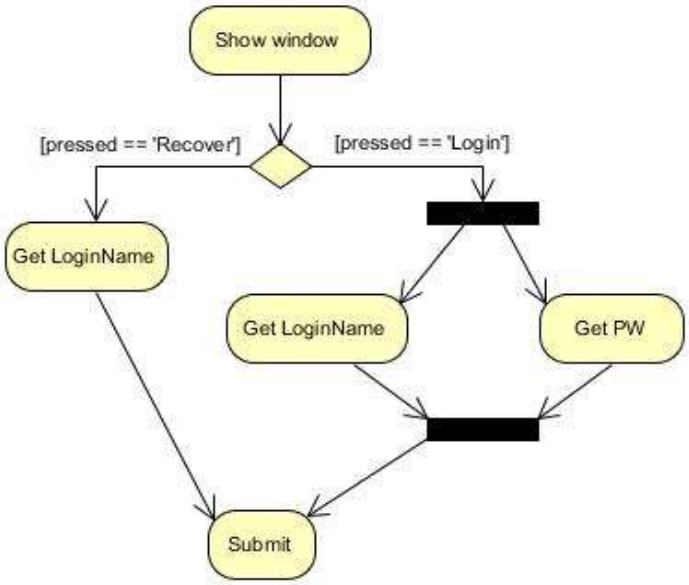

System start

а


System start
Show window
[pressed == 'Recover']
[pressed == 'Login']
Get LoginName
Login / PW

б


Show window
[pressed == 'Recover']
[pressed == 'Login']
Get LoginName
Get LoginName
Get PW
Submit



в

Рисунок 4.4 – Зображення основних графічних елементів діаграми дії:
а – окрема дія; *б* – альтернативне розгалуження; *в* – паралельне розділення та злиття (або синхронізація)

З ромба може виходити дві або більше стрілок, кожна з яких відповідає окремій гілці подальшої логіки виконання. Для кожної з цих стрілок обов'язково зазначається **сторожова умова** у вигляді логічного виразу, яка визначає, чи буде обрано відповідну гілку.

Це дозволяє системі приймати рішення на основі поточних умов або результатів попередніх операцій, забезпечуючи розгалуження виконання процесу в залежності від ситуації.

Одним із ключових недоліків традиційних блок-схем або структурних схем алгоритмів є складність у відображенні паралельних гілок обчислень. У UML ця проблема вирішується за допомогою спеціального символу, який використовується для розділення та злиття потоків управління або паралельних обчислень у програмній системі (ПС).

На діаграмі цей символ представлений у вигляді горизонтальної лінії, товщина якої більша за стандартні суцільні лінії діаграми діяльності.

– **Розділення (fork)** передбачає один вхідний перехід і декілька вихідних, що відповідають за створення паралельних потоків виконання.

– **Злиття (join)** або синхронізація, навпаки, має кілька вхідних переходів і один вихідний, що сигналізує про завершення паралельних процесів і їх об'єднання в один потік (рис. 4.4, в).

На діаграмах діяльності також можна відобразити факт, що певні дії в ПС виконуються різними суб'єктами. Наприклад, окремі бізнес-процеси можуть бути пов'язані з певними підрозділами компанії, або ж обробка даних розподіляється між клієнтським додатком та сервером.

Для відображення цих особливостей у UML застосовується концепція **доріжок (swimlanes)**. Цей підхід має візуальну аналогію з доріжками в

плавальному басейні. На діаграмі діяльності всі стани дії поділяються на групи, які розділяються вертикальними лініями. Кожна група (доріжка) відповідає окремому суб'єкту, такому як відділ компанії, група користувачів або програмний модуль.

Приклад такої діаграми представлено на рис. 4.5. У ній зображено дві доріжки:

– **Client** – відповідає за дії, що виконуються на стороні клієнта.
– **Server** – показує операції, які здійснюються сервером.

Процес починається зі стану **System start**, після чого відображається діалогове вікно (**Show window**). Далі система може розвиватися за двома варіантами залежно від виконуваних дій та взаємодії між клієнтом і сервером.

Цей підхід дозволяє чітко структурувати роботу різних частин системи та показує, як їхня взаємодія впливає на загальний хід виконання процесу.

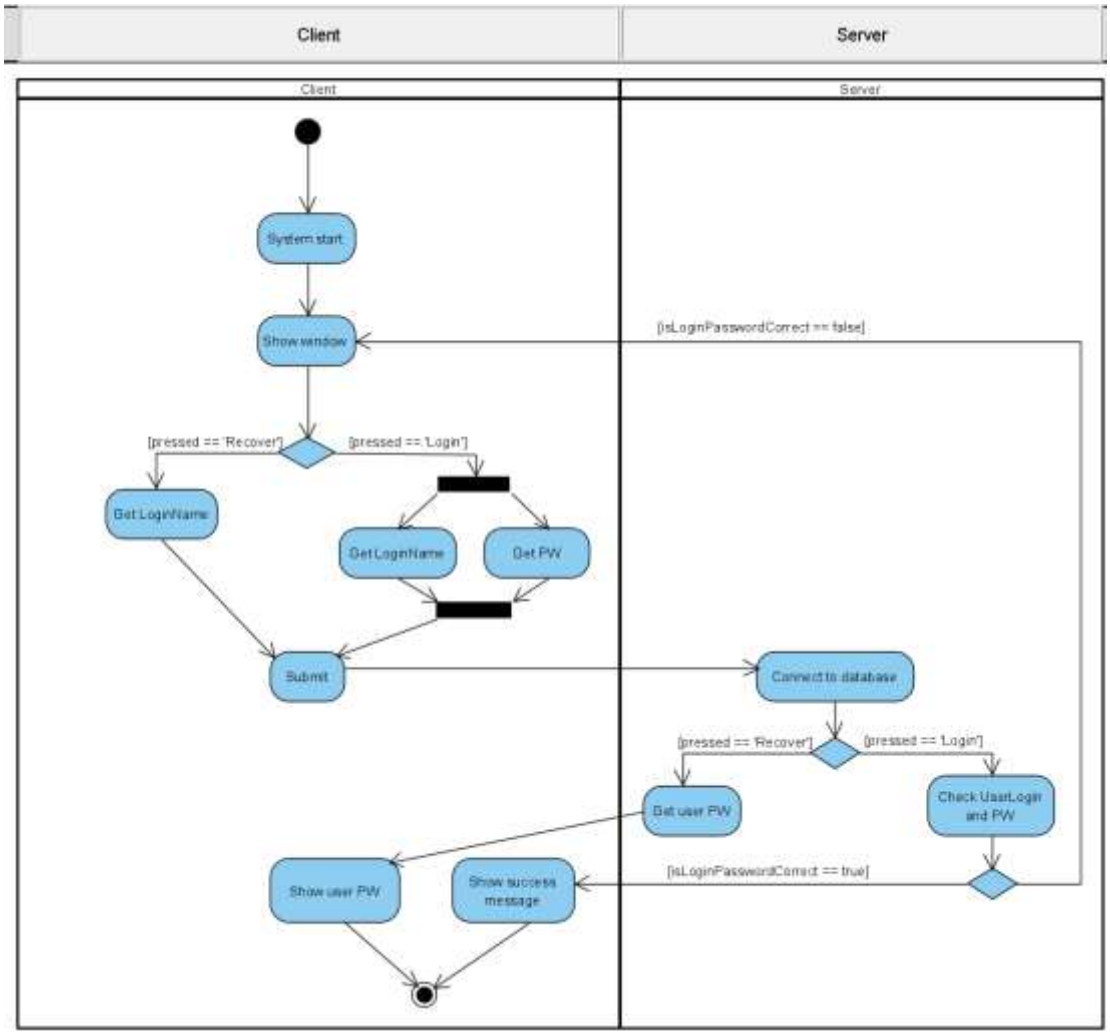


Рисунок 4.5 – Приклад побудови діаграми дії з використанням доріжок

1. Вхід у систему відбувається, якщо виконується умова **pressed == "Login"**. Після цього користувач вводить ім'я (LoginName) та пароль (Get PW). Ці дії можуть виконуватись паралельно, оскільки вони не залежать одна від одної, але завершуються синхронно під час виконання дії **Submit**. Після цього сервер встановлює підключення до бази даних (дія **Connect to database**) і виконує перевірку облікових даних (**Check Login and PW**).

– Якщо перевірка успішна (**isLoginPasswordCorrect == true**), система показує вітання користувачеві (**Show success message**) і переходить у кінцевий стан.

– У разі невдалої перевірки (**isLoginPasswordCorrect == false**) виконується перехід до активності **Show window** для повторної спроби входу.

2. Відновлення пароля виконується, якщо спрацьовує умова **pressed == "Recover"**. У цьому випадку система запитує ім'я користувача (**LoginName**), після чого сервер підключається до бази даних і виконує пошук пароля. Після завершення пошуку система виконує активність **Show user PW** і переходить у кінцевий стан (рис. 4.5).

Діаграми комунікації (communication diagram)

Для створення діаграми комунікації в середовищі **Visual Paradigm** необхідно вибрати вкладку **UML** і натиснути на пункт **Communication Diagram** (рис. 4.6).

Chec

Рисунок 4.6 – Графічне зображення об’єкта на діаграмі комунікації

Діаграма комунікації в UML має унікальну особливість – вона дозволяє не лише продемонструвати послідовність взаємодії між об’єктами, але й показати всі структурні зв’язки між ними.

На діаграмі об’єкти, які беруть участь у взаємодії, зображуються у вигляді прямокутників. У межах кожного прямокутника вказується назва об’єкта, його клас, а також можуть бути зазначені значення атрибутів. Наприклад, об’єкт із назвою **dw** належить до класу **DialogWindow** (рис. 4.6).

Цей тип діаграми дозволяє візуалізувати канали зв’язку між об’єктами та продемонструвати, як саме вони обмінюються повідомленнями під час виконання певних процесів.

Подібно до діаграми класів, на діаграмі комунікації вказуються асоціації між об’єктами у вигляді сполучних ліній. Також можна додати імена асоціацій та ролі об’єктів у цих асоціаціях. Крім статичних зв’язків, діаграма демонструє **динамічні зв’язки**, які відображають потоки повідомлень між об’єктами. Такі зв’язки зображуються у вигляді ліній зі

стрілками, що вказують напрямок передачі повідомлення. Над стрілкою розташовуються ім'я повідомлення та порядковий номер, що показує місце повідомлення в загальній послідовності взаємодії.

Поведінка системи моделюється через обмін повідомленнями між об'єктами, які прагнуть досягти певної мети або виконати конкретний програмний сервіс. Для аналітиків та розробників важливо мати чітке уявлення про структурні зв'язки між об'єктами. Саме це забезпечує **діаграма кооперації** (інша назва – діаграма комунікації).

На відміну від діаграм послідовності, діаграми кооперації фокусуються на відношеннях між об'єктами та їх ролях у процесі взаємодії. Однак, на цих діаграмах не представлено вимір часу як окремий параметр. Для визначення послідовності взаємодій та паралельних потоків використовуються порядкові номери асоціацій.

Якщо необхідно явно відобразити зв'язки між об'єктами в режимі реального часу, для цього краще використовувати **діаграми послідовності** (див. л/р 2).

На рис. 4.7 наведено приклад діаграми кооперації для процесу авторизації користувача в ПС. У цьому прикладі об'єкт **dw** класу **DialogWindow** надсилає повідомлення **setLoginName** та **setPW** об'єкту **user** класу **User**.

Ця діаграма наочно демонструє, як об'єкти взаємодіють між собою та яким чином відбувається процес авторизації через передачу ключових повідомлень.

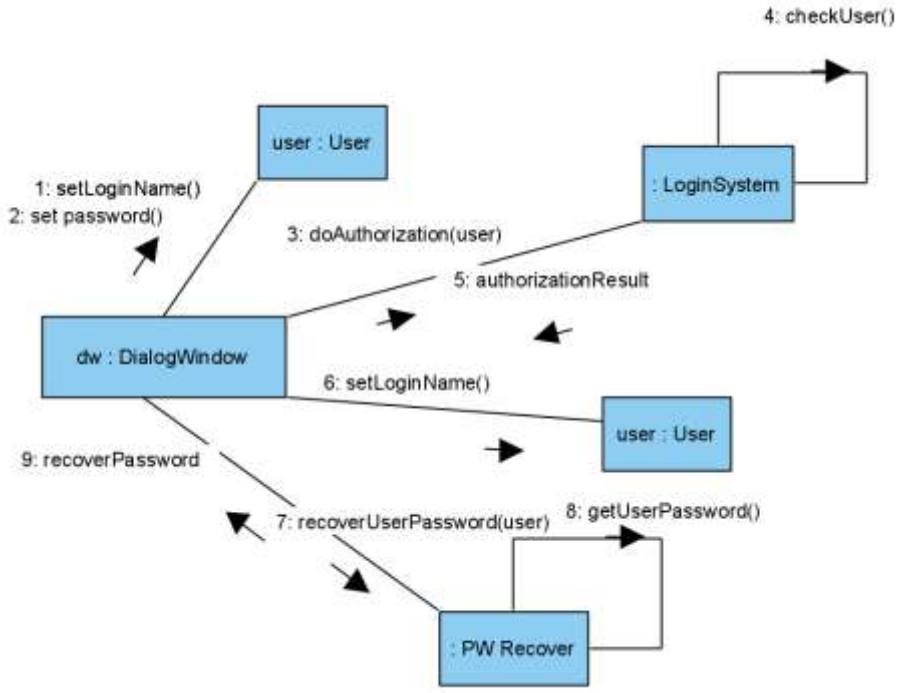


Рисунок 4.7 – Приклад побудови діаграми комунікації

Після того, як об'єкт **user** створено, він передається системі для виконання перевірки (авторизації). Об'єкт **dw** класу **DialogWindow** надсилає повідомлення **doAuthorization** з параметром **user**. У відповідь на

це об’єкт **LoginSystem** виконує метод **checkUser**, а результат авторизації передається у вигляді повідомлення **AuthorizationResult**.

У процесі відновлення пароля об’єкт **dw** класу **DialogWindow** спочатку надсилає повідомлення **setLoginName** об’єкту **user** класу **User**. Після цього об’єкт **user** передається системі для відновлення пароля. Об’єкт **dw** надсилає повідомлення **recoverUserPassword** з параметром **user**. У відповідь об’єкт **PWRecover** виконує метод **getUserPassword**, а результат відновлення пароля передається за допомогою повідомлення **RecoveredPassword** (рис. 4.7).

Цей процес ілюструє, як відбувається взаємодія між об’єктами для реалізації основних функцій авторизації та відновлення пароля в системі. Повідомлення між об’єктами чітко вказують послідовність викликів методів та передачі даних, що дозволяє системі ефективно виконувати потрібні дії.

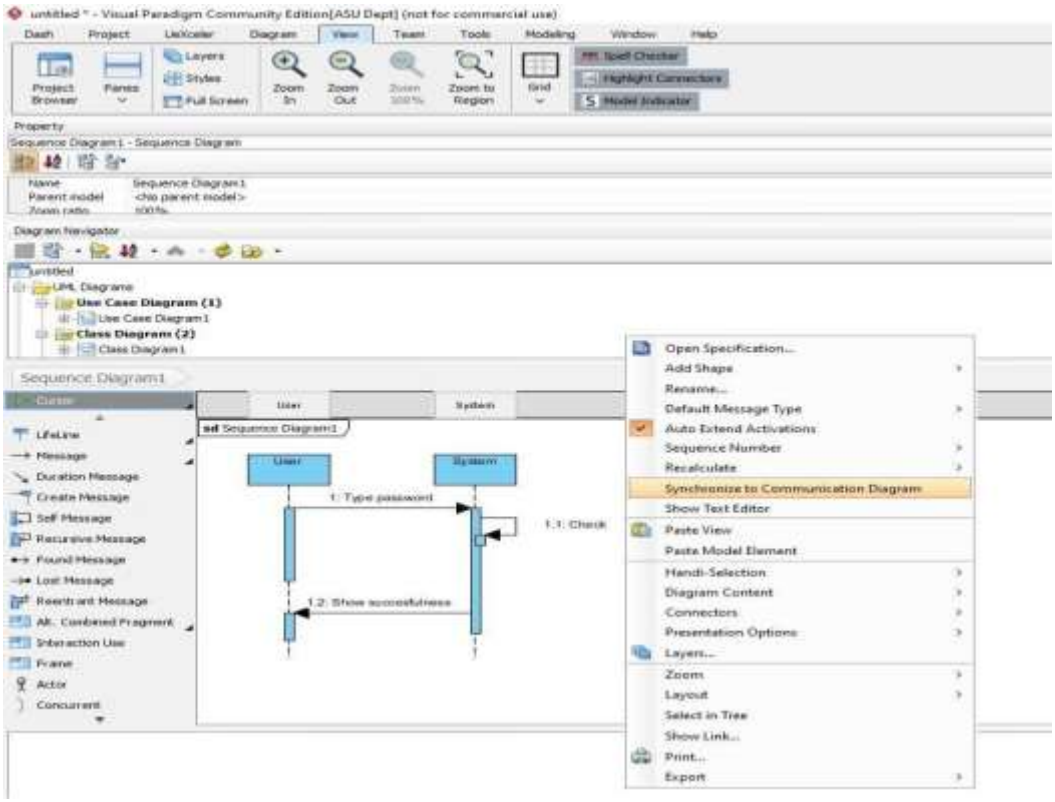


Рисунок 4.8 – Приклад інтерфейсу при автоматичній генерації діаграми комунікації

У Visual Paradigm (VP) передбачена можливість автоматичної генерації діаграм комунікації на основі існуючих діаграм послідовностей. Для цього достатньо натиснути правою кнопкою миші на обраній діаграмі послідовності та вибрати опцію **Synchronize to Communication Diagram** (рис. 4.8). Ця функція дозволяє швидко отримати діаграму комунікації, яка відображає структурні зв’язки між об’єктами, що беруть участь у взаємодії.

Діаграми огляду взаємодії (interaction overview diagram)

Діаграма огляду взаємодії є комбінацією діаграм діяльності та діаграм послідовності. Вона поєднує елементи обох типів діаграм, що дозволяє одночасно моделювати потік управління та взаємодію об’єктів у системі.

Діаграму огляду взаємодії можна розглядати як **діаграму діяльності**, в якій окремі дії замінені невеликими діаграмами послідовності. З іншого боку, її можна трактувати як **діаграму послідовності**, розділену за допомогою елементів нотації діаграм діяльності для відображення управління потоком.

У будь-якому випадку, діаграми огляду взаємодії є потужним інструментом, що забезпечує візуалізацію складних сценаріїв поведінки системи шляхом комбінування різних методів моделювання.

На рис. 4.9 представлено приклад діаграми огляду взаємодії для процедури авторизації користувача програмної системи (ПС). Цей приклад демонструє, як різні етапи процесу авторизації інтегруються в єдину діаграму, дозволяючи аналітикам та розробникам отримати цілісне уявлення про процес.

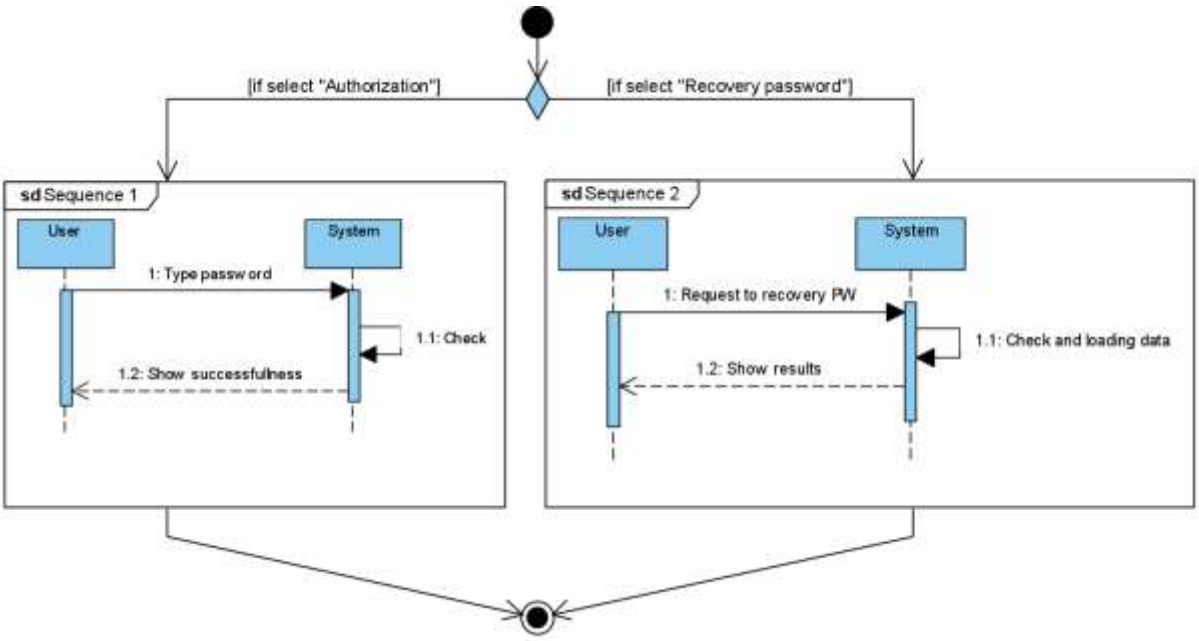


Рисунок 4.9 – Приклад побудови діаграми обзору взаємодії

5 РОЗРОБКА UML–ДІАГРАМ ФІЗИЧНОГО РІВНЯ ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

Діаграма компонентів (component diagram) в UML відображає фізичне представлення програмної системи, дозволяючи описати її архітектуру та визначити залежності між програмними компонентами, такими як вихідний код, бібліотеки та модулі.

Цей тип діаграми є важливим інструментом для системних аналітиків, архітекторів та програмістів, оскільки допомагає візуалізувати, як окремі частини системи поєднуються на рівні фізичних об’єктів.

Компонент (component) в UML – це елемент, який реалізує певний набір інтерфейсів і відображає фізичні сутності програмної системи. Графічно компонент позначається прямокутником з двома меншими прямокутниками, вставленими ліворуч. Усередині великого прямокутника вказується ім’я компонента, а за необхідності додається додаткова інформація. Імена компонентів відповідають загальним правилам іменування в UML і можуть містити літери, цифри та інші допустимі символи.

На діаграмі компонентів UML компоненти можуть представляти фізичні файли, які можна виконати безпосередньо (наприклад, файли з розширенням **.exe, .dll**), або файли, які використовуються для створення кінцевої програмної системи, такі як вихідні коди з розширеннями **.php, .cpp** для C++, HTML-сторінки (**.html**) або довідкові файли (**.hlp**).

Таким чином, діаграма компонентів демонструє, як програмні артефакти організовані та взаємодіють, що сприяє кращому розумінню структури системи на фізичному рівні.

На рис. 5.1 представлений приклад простої діаграми компонентів [4], який ілюструє базові принципи побудови такого типу діаграм.

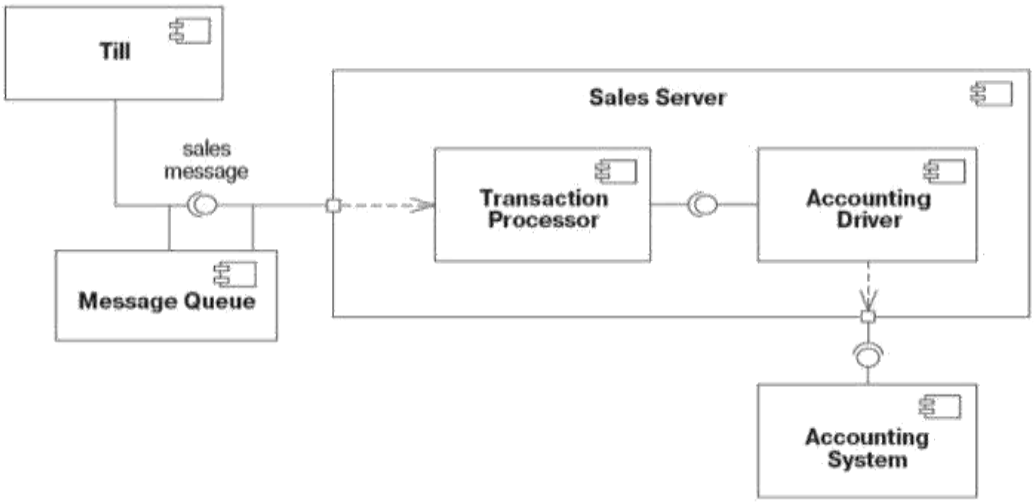


Рисунок 5.1 – Приклад діаграми компонентів

У цьому прикладі компонент **Till (Каса)** взаємодіє з компонентом **Sales Server (Сервер Продажу)** через інтерфейс **sales message (повідомлення про продаж)**. Оскільки мережа може бути ненадійною, для забезпечення стабільної роботи між касою та сервером додано компонент **Message Queue (Черга Повідомлень)**.

Черга повідомлень дозволяє касі продовжувати роботу навіть тоді, коли мережа недоступна. У разі відключення каса надсилає повідомлення до **Message Queue**, яка зберігає їх до відновлення підключення. Після відновлення мережі **Message Queue** пересилає накопичені повідомлення до **Sales Server**.

Таким чином, **Message Queue** забезпечує інтерфейс для обміну даними з касою та потребує такого ж інтерфейсу для зв’язку з сервером продажу.

Sales Server складається з двох основних компонентів:

– **Transaction Processor (Процесор транзакцій)** – реалізує інтерфейс для обробки повідомлень, що надходять від каси через чергу повідомлень.

– **Accounting Driver (Драйвер рахунків)** – відповідає за зв’язок із **Accounting System (Система обліку рахунків)** для подальшої обробки транзакцій та ведення бухгалтерського обліку.

Ця архітектура дозволяє підтримувати безперервну роботу касових апаратів і гарантує, що всі дані про продажі будуть передані до сервера, навіть у разі тимчасової відсутності мережевого з’єднання.

Діаграма розгортання (deployment diagram) у UML призначена для візуалізації фізичних компонентів і елементів програмної системи (ПС), які існують під час її виконання (runtime). На такій діаграмі зображуються лише ті компоненти, що є виконуваними файлами (наприклад, **.exe**) або динамічними бібліотеками (**.dll**). Компоненти, які використовуються виключно на етапі розробки (наприклад, вихідні коди програм), на діаграмі розгортання не відображаються. Вони можуть бути присутні лише на **діаграмі компонентів**.

Діаграма розгортання демонструє графічні зображення процесорів, пристроїв, процесів і зв'язків між ними. На відміну від діаграм логічної структури (наприклад, **діаграми класів**), діаграма розгортання є єдиною для всієї ПС, оскільки повинна охоплювати всі аспекти її фізичної реалізації. Як правило, створення діаграми розгортання є фінальним етапом специфікації моделі ПС.

Цілі створення діаграми розгортання:

– Визначити, як компоненти системи розподіляються між фізичними вузлами мережі.

– Показати фізичні зв’язки між вузлами, які забезпечують виконання системи.

– Виявити потенційні вузькі місця в системі та провести її реконфігурацію для досягнення оптимальної продуктивності.

Розробка діаграми здійснюється в тісній співпраці системних архітекторів, інженерів-системотехніків та програмістів.

Вузол (node) – це фізичний елемент системи з певними обчислювальними ресурсами (процесор, пам’ять тощо). У нових версіях UML вузли можуть представляти не тільки обчислювальні пристрої, але й інші механічні або електронні компоненти, такі як датчики, принтери, модеми, камери, сканери тощо.

Графічне позначення вузла – це тривимірний куб із підписом, що вказує його ім'я.

Приклад реалізації діаграми розгортання:

1. Реалізація на платформі JEE (Java Enterprise Edition): На вузлі **Client** розгорнуто два компоненти (рис. 5.3):

– **Web Browser** – веб-браузер, який забезпечує доступ користувача до системи.

– **JavaScript Framework** – JavaScript-каркас, що використовується для реалізації бізнес-логіки на стороні клієнта.

2. **Реалізація на платформі PHP:**

На вузлі **Server** розгортаються компоненти PHP, що виконують серверну логіку системи.

Ці діаграми дозволяють чітко побачити, як компоненти розподіляються по вузлах у мережі, які пристрої беруть участь у виконанні ПС та як вони взаємодіють під час роботи системи.

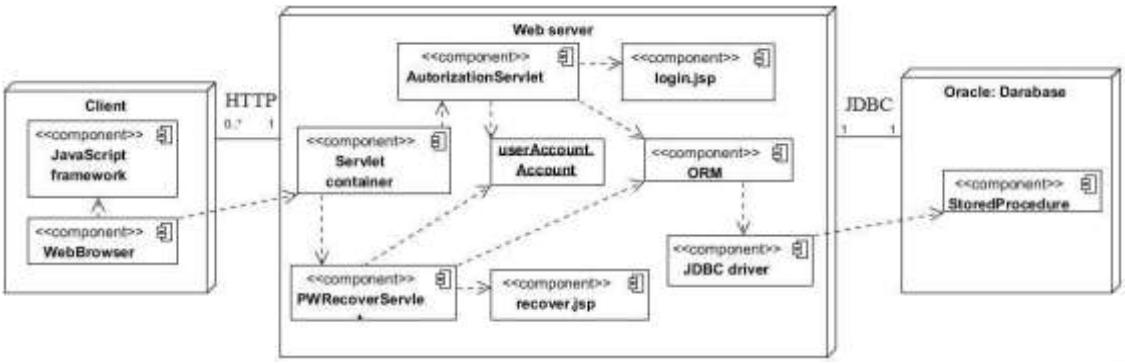


Рисунок 5.2 – Діаграма розгортання для реалізації на Java

Вузол Web server – це центральний елемент системи, який забезпечує обслуговування клієнтів за допомогою протоколу **HTTP**. Клієнтська частина системи взаємодіє з веб-сервером, надсилаючи запити, на які сервер відповідає відповідно до своєї конфігурації та розгорнутих компонентів.

Зв’язок між клієнтом і веб-сервером має кардинальність **«0.. : 1»**, що означає можливість одночасної роботи багатьох клієнтів із одним сервером.

На вузлі **Web server** розгорнуто такі компоненти:

1. **Servlet container** – контейнер java-сервлетів, який забезпечує виконання серверної логіки, приймаючи запити від клієнтів і генеруючи відповіді.

2. **AuthorizationServlet** – сервлет, що відповідає за аутентифікацію користувачів у системі.

3. **PWRecoverServlet** – сервлет, який обробляє процес відновлення пароля користувача.

4. **ORM (Object-Relational Mapping)** – компонент, що забезпечує об’єктно-реляційне відображення даних для зручної роботи з базою даних.

5. **login.jsp** – серверна JSP-сторінка, яка відповідає за відображення інтерфейсу для аутентифікації користувача.

6. **Recover.jsp** – серверна JSP-сторінка, що використовується для відображення інтерфейсу під час відновлення пароля користувача.

7. **userAccount** – програмний об’єкт, що представляє обліковий запис користувача і створюється в процесі роботи системи.

8. **JDBC Driver** – драйвер для підключення до бази даних. Він створюється під час аутентифікації або відновлення пароля користувача.

Вузол Database – це сервер бази даних, який зберігає всю інформацію системи. У даному випадку база даних реалізована за допомогою СКБД **Oracle**.

Зв’язок між веб-сервером і сервером баз даних здійснюється через протокол **JDBC**. Кардинальність цього зв’язку – **«1:1»**, що означає, що на один веб-сервер припадає один сервер баз даних.

У контейнері сервера баз даних також розгорнуто компонент **StoredProcedure** – це збережені процедури, які виконують додаткову бізнес-логіку системи безпосередньо на рівні бази даних.

Реалізація на платформі PHP

У випадку реалізації на платформі PHP, структура системи має спрощений вигляд.

На вузлі **Client** розгорнуто два компоненти (рис. 5.3):

1. **Web Browser** – веб-браузер, який забезпечує доступ користувачів до системи авторизації.

2. **JavaScript Framework** – JavaScript-каркас, що використовується для виконання додаткової бізнес-логіки на стороні клієнта.

У цій конфігурації веб-сервер обробляє запити клієнтів і взаємодіє з базою даних, забезпечуючи всі основні функції системи, такі як авторизація та відновлення пароля.

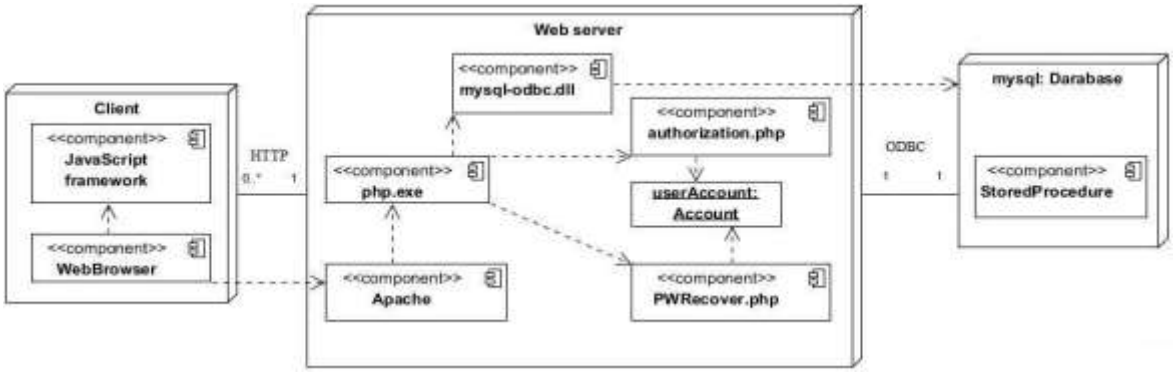


Рисунок 5.3 – Діаграма розгортання у разі реалізації на PHP

Вузол Web server виступає як веб-сервер системи, з яким клієнт взаємодіє через протокол **HTTP**. Розмірність зв'язку *«0.. : 1»** означає, що декілька клієнтів можуть одночасно працювати з одним веб-сервером.

На цьому веб-сервері розгорнуто такі компоненти:

1. **Apache** – компонент веб-сервера, який відповідає за виконання PHP-скриптів та обробку запитів клієнтів.
2. **authorization.php** – PHP-скрипт, який виконує функції аутентифікації користувача в системі.
3. **PWRecover.php** – PHP-скрипт для відновлення пароля користувача.
4. **php.exe** – інтерпретатор PHP, який обробляє серверний код і забезпечує виконання PHP-скриптів.
5. **php-mysql.dll** – бібліотека драйверів, яка забезпечує підключення PHP-сторінок до бази даних за допомогою MySQL.
6. **userAccount** – об'єкт, що представляє обліковий запис користувача. Він створюється під час процесу аутентифікації або відновлення пароля.

Вузол Database містить базу даних системи, яка реалізована на платформі **MySQL**. Зв'язок між веб-сервером і базою даних здійснюється через протокол **ODBC**, із розмірністю **«1:1»**, що означає наявність одного веб-сервера і одного сервера бази даних у системі.

У контейнері сервера баз даних передбачено компонент **StoredProcedure** – це збережені процедури, що виконують додаткову бізнес-логіку на рівні бази даних.

Сервер баз даних також може взаємодіяти через протокол **JDBC**, що дозволяє з'єднувати сервер додатків із базою даних для виконання запитів і збережених процедур.

Таким чином, діаграма розгортання показує, як компоненти системи розподіляються між фізичними вузлами, і демонструє взаємодію клієнтської частини, серверної частини та бази даних під час виконання основних процесів, таких як аутентифікація або відновлення пароля.

6 РОЗРОБКА ER–ДІАГРАМИ ЛОГІЧНОГО РІВНЯ ПРОЕКТУВАННЯ БАЗИ ДАНИХ

ER-діаграма (Entity-Relationship Diagram) є основним інструментом для логічного моделювання структури бази даних. Вона дозволяє графічно відобразити сутності, їх атрибути та взаємозв'язки між ними, які визначають логіку взаємодії даних у системі.

Цей тип діаграм використовується аналітиками, архітекторами баз даних та розробниками для проектування логічної структури бази даних перед її фізичною реалізацією. Завдяки ER-діаграмам забезпечується чітке розуміння структури даних та залежностей між ними.

Сутність (entity) – це об'єкт, інформацію про який потрібно зберігати в базі даних. Графічно сутність зображується у вигляді прямокутника, всередині якого вказується її назва.

Атрибути (attributes) – характеристики сутності, які визначають її властивості. Атрибути зображуються у вигляді овалів, пов'язаних лінією із сутністю.

Зв'язки (relationships) – це взаємозв'язки між сутностями, які відображають логічну залежність між ними. Зв'язки зображуються у вигляді ромбів, пов'язаних лініями із відповідними сутностями. ER-діаграма допомагає визначити кардинальність зв'язків (наприклад, «один до одного», «один до багатьох» або «багато до багатьох») та забезпечує основу для подальшого створення реляційної бази даних.

Діаграма класів і ER-діаграма є взаємопов'язаними, оскільки вони обидві моделюють структуру даних та їх взаємозв'язки. Однак їх використання і фокус дещо різняться:

Діаграма класів зосереджується на моделюванні об'єктно-орієнтованої структури системи. Вона відображає класи, їх атрибути, методи, а також асоціації між класами. Діаграма класів використовується для проектування програмної системи на рівні програмного коду, де кожен клас представляє шаблон для створення об'єктів.

ER-діаграма моделює структуру даних у вигляді сутностей (entities) і зв'язків (relationships). Вона є основою для проектування бази даних і допомагає визначити, які дані повинні зберігатися, їх атрибути і залежності.

Зв'язок між діаграмою класів і ER-діаграмою полягає у відповідності між об'єктно-орієнтованою моделлю та реляційною базою даних. Основні аспекти цього зв'язку:

1. **Класи і таблиці**:
   – Кожен клас діаграми класів може бути представлений таблицею в ER-діаграмі.
   – Атрибути класів відповідають стовпцям таблиць бази даних.
2. **Зв'язки між класами і відношення в ER-діаграмі**:
   – Асоціації між класами діаграми класів трансформуються у зовнішні ключі та зв'язки між таблицями в ER-діаграмі.

– Наприклад, асоціація "один до багатьох" у діаграмі класів відповідає зовнішньому ключу в таблиці бази даних.

3. **Успадкування і підтипи**: успадкування в діаграмі класів може бути представлене як окрема таблиця для кожного підтипу або як спільна таблиця із спеціальним атрибутом, який вказує на тип.
4. **Атрибути і їх типи даних**: атрибути класів у діаграмі класів відповідають стовпцям таблиць у базі даних із зазначенням типу даних.
5. **Методи**: методи класів у діаграмі класів зазвичай не відображаються в ER-діаграмі, оскільки остання моделює лише структуру даних.

Приклад зв'язку:

– У діаграмі класів є клас User з атрибутами email, password, firstName, lastName.

– У ER-діаграмі це трансформується в таблицю users з полями email, password, first_name, last_name.

Таким чином, діаграма класів допомагає визначити програмну структуру системи, тоді як ER-діаграма деталізує, як дані будуть організовані і зберігатися в базі даних. У процесі проектування обидві ці діаграми працюють разом, забезпечуючи узгодженість між програмною логікою та структурою даних.

На рис. 6.1 представлений приклад ER-діаграми, що демонструє основні принципи побудови та проектування бази даних, включаючи сутності, атрибути та взаємозв'язки між ними.

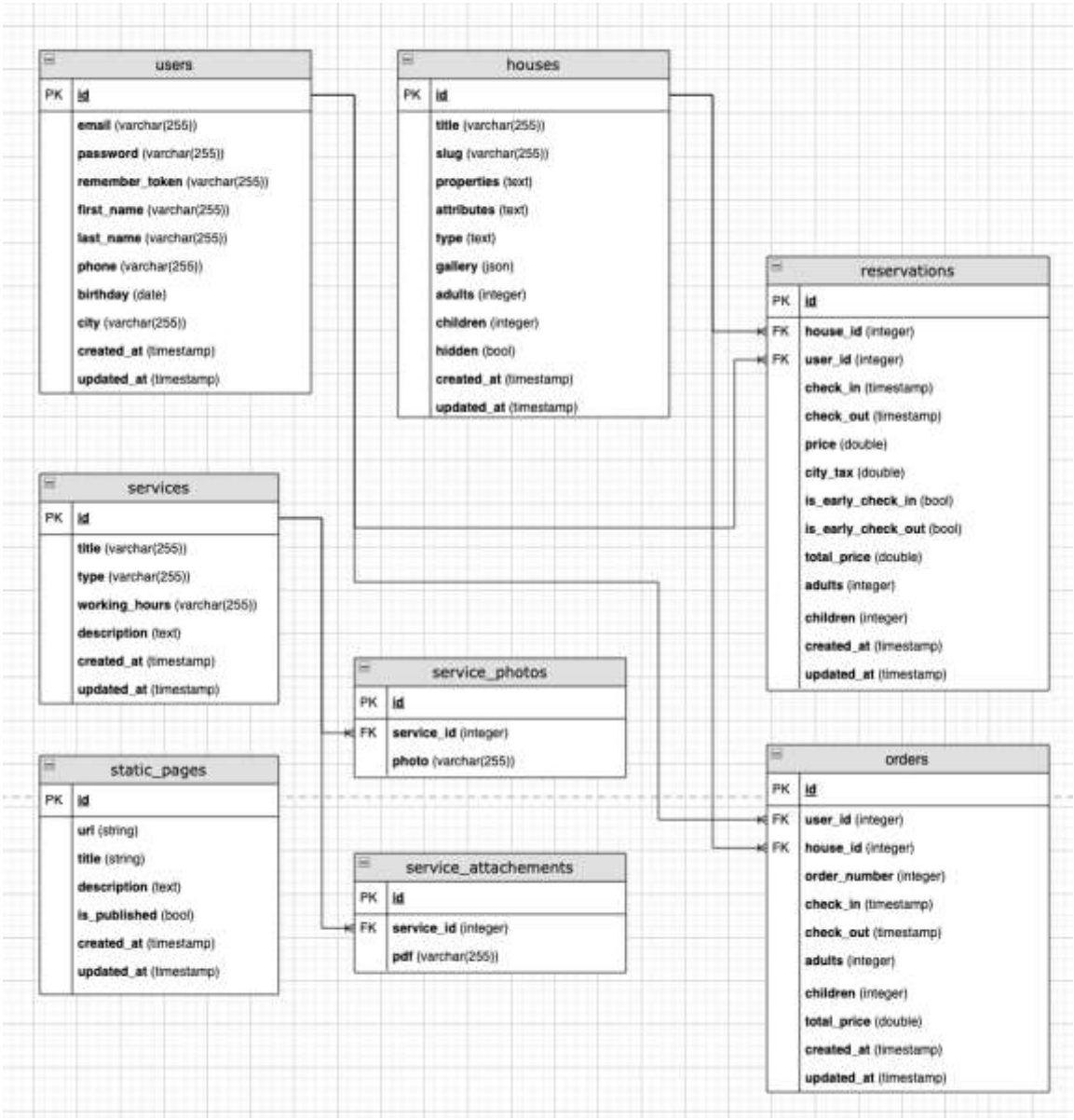


Рисунок 6.1 – ER-діаграма системи бронювання

Ця діаграма є ER-діаграмою, яка моделює структуру бази даних системи, що включає управління користувачами, будинками, сервісами, резервуваннями, замовленнями та статичними сторінками.

Далі подано детальний опис таблиць та їх взаємозв'язків:

1. **Users (Користувачі)**:

– Таблиця містить інформацію про користувачів системи.

– Основні атрибути: email, password, first_name, last_name, phone, city, birthday.

– Поля created_at і updated_at використовуються для відстеження часу створення та оновлення записів.

– Ключовий зв'язок: пов'язана з таблицею reservations та orders, де визначається користувач, що здійснив резервування або замовлення.

2. **Houses (Будинки)**:

– Містить інформацію про доступні будинки.

– Основні атрибути: title, slug, properties, type, adults, children, hidden.
– Поле gallery зберігає медіафайли у форматі JSON.
– Зв’язок: пов’язана з таблицею reservations через поле house_id, яке вказує, для якого будинку здійснено резервування.

3. **Services (Послуги)**:
– Зберігає інформацію про послуги, які надаються.
– Основні атрибути: title, type, working_hours, description.
– Зв’язок: має залежність від таблиць service_photos (для зображень послуг) і service_attachments (для прикріплених PDF-файлів).

4. **Static Pages (Статичні сторінки)**:
– Таблиця зберігає інформацію про статичні сторінки, які є частиною системи (наприклад, сторінки політики конфіденційності).
– Основні атрибути: url, title, description, is_published.
– Поля created_at і updated_at визначають час створення та оновлення сторінки.

5. **Reservations (Резервування)**:
– Таблиця, яка містить інформацію про резервування будинків.
– Основні атрибути: house_id, user_id, check_in, check_out, price, city_tax, total_price, adults, children.
– Поля is_early_check_in та is_early_check_out визначають спеціальні умови бронювання.
– Зв’язок: пов’язана з таблицями users і houses.

6. **Orders (Замовлення)**:
– Таблиця, що зберігає дані про замовлення користувачів.
– Основні атрибути: user_id, house_id, order_number, check_in, check_out, total_price, adults, children.
– Зв’язок: пов’язана з таблицями users і houses.

7. **Service Photos (Фотографії послуг)**:
– Таблиця зберігає зображення, що належать до послуг.
– Основні атрибути: service_id (посилання на таблицю services), photo.

8. **Service Attachments (Прикріплені файли до послуг)**:
– Таблиця для зберігання додаткових файлів до послуг.
– Основні атрибути: service_id (посилання на таблицю services), pdf.

Загальний опис зв’язків:

Таблиці пов’язані між собою через первинні (PK) та зовнішні ключі (FK), що забезпечує інтеграцію даних і підтримку референційної цілісності.

Наприклад:

– users пов’язана з reservations через поле user_id, що дозволяє відслідковувати, які користувачі створювали резервування.
– services має відношення з додатковими таблицями для зберігання фотографій та файлів.

Ця структура бази даних є гнучкою і добре підходить для системи управління орендами, послугами та користувачами.

7 ПОРЯДОК ВИКОНАННЯ ТА ЗАХИСТУ КУРСОВОЇ РОБОТИ

7.1 Етапи виконання та захисту

До основних етапів виконання курсової роботи належать.

1. Вибір напряму дослідження.

Визначення актуальності обраної предметної області дослідження, визначення структури роботи й об'єкта дослідження.

Здобувач має право самостійно здійснювати вибір теми курсової роботи, виходячи із власного професійного інтересу, а також з огляду на актуальність питання (проблеми) для підприємства.

Здобувачу слід враховувати специфіку підприємства, його вимоги щодо збереження корпоративної таємниці, доступність технологічної та техніко-економічної інформації та інші специфічні умови та обмеження в наслідок бойових дій в країні.

2 Здійснення огляду джерел

Аналіз предметної області тематики курсової рооти виконується на основі огляду інформації, опублікованої в навчальній і науково-технічній літературі, в науково-технічних статтях, вебресурсах, інших доступних джерелах інформації.

3. Формулювання теми курсової роботи.

Неприпустимим є вибір однієї теми декількома здобувачами вищої освіти в межах однієї академічної групи. Можливим є виконання комплексних робіт в межах одного технологічного об'єкту. Формулювання теми курсової роботи має бути стислим та вказувати на головний результат дослідження. Теми робіт розглядаються і погоджуються на засіданні кафедри автоматизації виробничих процесів.

4. Виконання курсової роботи.

Після погодження та затвердження плану роботи здобувач вищої освіти починає написання роботи. Вимоги до структури й оформлення окремих розділів наведені в цих методичних рекомендаціях. У процесі написання окремих розділів здобувач вищої освіти подає їх керівнику на перевірку, виправляє та вносить доповнення у разі потреби, звітує керівнику про готовність роботи. Обговорення проблемних питань з викладачем – керівником здійснюється під час індивідуально-консультативних зустрічей з підготовки та захисту курсової роботи або на консультаціях викладача відповідно до затвердженого розкладу.

5 Подання роботи на перевірку.

Контроль виконання, подання на перевірку і представлення закінченої курсової роботи здійснюється керівником роботи.

Керівник курсової роботи надає здобувачу освіти свої зауваження, коментарі, рекомендації, на підставі яких він виправляє роботу.

Завершену, остаточно оформлену роботу, підписану здобувачем, керівником здобувач надає для проведення перевірки робіти на наявність плагіату у роботи.

7.2 Права та обов’язки керівника курсової роботи, здобувача вищої освіти

Керівництво та консультування курсовою роботою здійснюється з метою надання здобувачу вищої освіти необхідних консультацій, контролю термінів виконання та якості роботи.

Обов’язки керівника роботи є:

– пояснення основних вимог до курсової роботи;

– узгодження розроблення разом зі здобувачем вищої освіти предметного напряму дослідження;

– надання консультацій з питань, що виникають у здобувача під час проведення дослідження;

– надання рекомендацій щодо завершального етапу підготовки та захисту курсової роботи.

Обов’язки здобувача освіти:

– ознайомитись із цими методичними рекомендаціями;

– проявляти ініціативність та сумлінність при виконанні курсової роботи;

– своєчасно відвідувати консультації керівника;

– дотримуватися термінів виконання роботи;

– дотримуватися вимог академічної доброчесності при виконанні та захисті курсової роботи.

Права здобувача освіти:

– отримувати консультації та організаційно-методичні консультації з приводу виконання курсової роботи;

– отримувати роз’яснення від керівника щодо вирішення задач курсової роботи, підготовки тексту, підготовки захисту роботи;

– отримувати поради від керівника щодо літературних джерел та інших інформаційних ресурсів, які можна використати при виконанні курсової роботи;

– вимагати дотримання умов об’єктивності та дотримання процедури оцінювання курсової роботи;

– оскаржувати оцінку керівника та комісії з захисту роботи в установленому порядку.

7.3 Застереження щодо академічної доброчесності

Як член студентської спільноти ДДМА здобувач має дотримуватися певних стандартів та академічної політики відповідно «Положення про академічну доброчесність науково-педагогічних, наукових, педагогічних працівників та здобувачів вищої освіти Донбаської державної машинобудівної академії» (http://surl.li/laufq):

– шахрайство та плагіат заборонені.

– методичні та інші матеріали, які отримані здобувачами в рамках процедур організації виконання роботи, захищені авторським правом, можуть бути використані лише тільки здобувачами освіти, яким призначено даний курс. зарахованих на курс для цілей, пов'язаних з цим курсом і не можуть поширюватися.

– спілкування з однокурсниками та викладачем має бути професійним та ввічливим.

– очікується, що здобувач освіти перевірятиме всі власні письмові повідомлення, включаючи поштові повідомлення, на коректність змісту та мови.

– Академія прагне підтримувати середовище, вільне від дискримінації або дискримінаційних домагань, спрямованих на будь-яку людину або групу в межах своєї спільноти – здобувачів вищої освіти, співробітників або відвідувачів.

Виконання роботи має здійснюватися з урахуванням ***вимог щодо академічної доброчесності.*** Відповідно до статті 42 Закону України «Про освіту»: «***Академічна доброчесність*** – це сукупність етичних принципів та визначених законом правил, якими мають керуватися учасники освітнього процесу під час навчання, викладання та провадження наукової (творчої) діяльності з метою забезпечення довіри до результатів навчання та/або наукових (творчих) досягнень». Головним проявом академічної недоброчесності вважається академічний плагіат. ***Академічний плагіат*** – оприлюднення (частково або повністю) наукових (творчих) результатів, отриманих іншими особами, як результатів власного дослідження (творчості) та/або відтворення опублікованих текстів (оприлюднених творів мистецтва) інших авторів без зазначення авторства, а саме:

а) відтворення в тексті роботи (повний текст роботи, з коментарями, примітками, бібліографією, переліком джерел та всіма додатками до основного тексту) без змін, з незначними змінами, або в перекладі тексту іншого автора (інших авторів), обсягом від речення і більше, без посилання на автора (авторів) відтвореного тексту;

б) відтворення в тексті роботи, повністю або частково, тексту іншого автора (інших авторів) через його перефразування чи довільний переказ без посилання на автора (авторів) відтвореного тексту;

в) відтворення в тексті роботи наведених в іншому джерелі цитат з третіх джерел без вказування, за яким саме безпосереднім джерелом наведена цитата.

г) відтворення в тексті роботи наведеної в іншому джерелі науково-технічної інформації (крім загальновідомої) без вказування на те, з якого джерела взята ця інформація.

д) перефразування тексту джерела у формі, що є близькою до оригінального тексту, або наведення узагальнення ідей, інтерпретацій чи висновків з певного джерела без посилання на це джерело;

е) подання як власних робіт, виконаних на замовлення іншими особами, у тому числі робіт, стосовно яких справжні автори надали згоду на таке використання.

Запобігання академічному плагіату у роботах, наукових та науково-методичних працях, публікаціях Учасників освітнього процесу полягає у здійсненні технічної перевірки за допомогою спеціалізованих програмних засобів, що використовуються в Академії, та експертної оцінки щодо відсутності/наявності академічного плагіату. Процедура та особливості здійснення автоматизованої перевірки робіт, наукових та науково-методичних праць, публікацій Учасників освітнього процесу наведено в Тимчасовому положенні «Про запобігання та виявлення академічного плагіату у навчальній та науково-дослідній роботі учасників освітнього процесу у ДДМА» та в «Порядку проведення перевірки робіт студентів на наявність запозичень з інших документів»

Рекомендації щодо запобігання академічному плагіату в роботі:

а) робота має виконуватися самостійно, без видання за власний результат чужих робіт і результатів;

б) будь-який текстовий фрагмент обсягом від речення і більше, відтворений в тексті роботи без змін, з незначними змінами, або в перекладі з іншого джерела, обов'язково має супроводжуватися посиланням на це джерело (у формі підрядкового посилання, наприклад як це зроблено щодо Закону «Про освіту» на попередній сторінці); винятки допускаються лише для стандартних текстових кліше, які не мають авторства та/чи є загальновживаними;

в) якщо перефразування чи довільний переказ в тексті роботи тексту іншого автора (інших авторів) займає більше одного абзацу, посилання (бібліографічне та/або текстуальне) на відповідний текст та/або його автора (авторів) має міститися щонайменше один раз у кожному абзаці роботи, крім абзаців, що повністю складаються з формул, а також нумерованих та маркованих списків (в останньому разі допускається подати одне посилання наприкінці списку);

г) якщо цитата з певного джерела наводиться за першоджерелом, в тексті роботи має бути наведено посилання на першоджерело; якщо цитата наводиться не за першоджерелом, в тексті роботи має бути наведено посилання на безпосереднє джерело цитування («цитується за ХХХХХХХ») і посилання на відповідний пункт списку використаних джерел;

д) будь-яка наведена в тексті роботи науково-технічна інформація має супроводжуватися чітким вказуванням на джерело, з якого взята ця інформація із посиланням на відповідний пункт списку використаних джерел; винятки припускаються лише для загальновідомої інформації, визнаної всією спільнотою фахівців відповідного профілю; у разі використання у роботі тексту нормативно-правового акту достатньо зазначити його назву, дату ухвалення та, за наявності, дату ухвалення останніх змін до нього або нової редакції, а також посилання на відповідний пункт списку використаних джерел.

е) для підтвердження власних аргументів посиланням на авторитетне джерело або для критичного аналізу того чи іншого друкованого твору слід наводити цитати; науковий етикет потребує точно відтворювати цитований текст, бо найменше скорочення наведеного витягу може спотворити зміст, закладений автором.

Правила цитування та посилання на використані джерела є такими:

1. При написанні здобувач повинен давати посилання на джерела, матеріали з яких наводяться у роботі. Такі посилання дають змогу відшукати документи та перевірити достовірність відомостей про цитування документа, дають необхідну інформацію щодо нього, допомагають з'ясувати його зміст, мову тексту, обсяг. Посилатися бажано на останні видання публікацій. На більш ранні видання можна посилатися лише в тих випадках, коли в них є матеріал, який не включено до останнього видання.

2. Якщо використовують відомості, матеріали з монографій, оглядових статей, інших джерел з великою кількістю сторінок, тоді в посиланні необхідно точно вказати номери сторінок, ілюстрацій, таблиць, формул з джерела, на яке дано посилання в роботі.

3. Посилання додаються одразу після закінчення цитати у квадратних дужках, де вказується порядковий номер джерела у списку літератури та відповідна сторінка джерела (наприклад: [12, с. 172]), або під текстом цієї сторінки у вигляді зноски, в якій вказують прізвище та ініціали автора, назву джерела, видавництво, рік видання та сторінку. При цьому враховувати наступне:

– текст цитати починається і закінчується лапками і наводиться в тій граматичній формі, в якій він поданий у джерелі, із збереженням особливостей авторського написання; наукові терміни, запропоновані іншими авторами, не виділяються лапками, за винятком тих, що викликали загальну полеміку – у цих випадках використовується вираз «так званий»;

– цитування повинно бути повним, без довільного скорочення авторського тексту та без перекручень думок автора;

– пропуск слів, речень, абзаців при цитуванні допускається без перекручення авторського тексту і позначається трьома крапками, вони ставляться у будь-якому місці цитати (на початку, всередині, наприкінці); якщо перед випущеним текстом або за ним стояв розділовий знак, то він не зберігається;

– кожна цитата обов'язково супроводжується посиланням на джерело;

– при непрямому цитуванні (переказі, викладі думок інших авторів своїми словами), що дає значну економію тексту, слід бути гранично точним у викладенні думок автора, коректним щодо оцінювання його результатів і давати відповідні посилання на джерело;

– якщо необхідно виявити ставлення автора роботи до окремих слів або думок з цитованого тексту, то після них у круглих дужках ставлять знак оклику або знак питання;

– коли автор роботи, наводячи цитату, виділяє в ній деякі слова, то робиться спеціальне застереження, тобто після тексту, який пояснює виділення, ставиться крапка, потім дефіс і вказуються ініціали автора дисертації, а весь текст застереження вміщується у круглі дужки. Варіантами таких застережень є: (курсив наш. – М.Х.), (підкреслено мною. – М.Х.), (розбивка моя. – М.Х.).

До числа ***порушень академічної доброчесності***, класифікованих законодавством України, що можуть трапитися при виконанні роботи, належать:

– Академічний плагіат – оприлюднення (частково або повністю) наукових результатів, отриманих іншими особами, як результатів власного дослідження, та/або відтворення опублікованих текстів інших авторів без зазначення авторства;

– Самоплагіат – оприлюднення (частково або повністю) власних раніше опублікованих наукових результатів як нових наукових результатів;

– Академічне шахрайство – будь-які дії учасників освітньо-наукового процесу, сутність яких полягає у наданні або отриманні будь-якої несанкціонованої допомоги або нечесної переваги у будь-якій формі академічної роботи;

– Фабрикація – вигадування даних чи фактів, що використовуються в освітньому процесі або наукових дослідженнях;

– Фальсифікація результатів досліджень, посилань у власних публікаціях, будь-яких інших даних, у тому числі статистичних, що стосуються освітнього процесу та наукових досліджень;

– Списування – виконання письмових робіт із залученням зовнішніх джерел інформації, крім дозволених для використання, зокрема під час оцінювання результатів навчання;

– Обман – надання завідомо неправдивої інформації щодо власної освітньої (наукової, творчої) діяльності чи організації освітнього процесу; формами обману є, зокрема, академічний плагіат, самоплагіат, фабрикація, фальсифікація та списування;

– Хабарництво – надання (отримання) учасником освітнього процесу чи пропозиція щодо надання (отримання) коштів, майна чи послуг матеріального або нематеріального характеру з метою отримання неправомірної вигоди в освітньому процесі;

– Неправомірна вигода – грошові кошти або інше майно, переваги, пільги, послуги, нематеріальні активи, будь-які інші вигоди нематеріального чи не грошового характеру, що обіцяють, пропонують, надають або одержують без законних на те підстав;

– Необ’єктивне оцінювання – свідоме завищення або заниження оцінки результатів навчання здобувачів освіти.

Відповідальність за порушення академічної доброчесності під час здійснення освітньої та наукової діяльності покладається на Працівників та Здобувачів Академії.

Працівників та Здобувачів Академії несуть відповідальність за порушення академічної доброчесності відповідно до вимог законодавства України.

– За порушення норм академічної доброчесності Працівники Академії можуть бути притягнуті до відповідальності відповідно до нормативних і розпорядчих документів ДДМА та норм законодавства України;

– До Здобувачів Академії, у випадку порушення правил академічної доброчесності, в т.ч. встановлення факту плагіату, може бути застосовано такі види заходів впливу: – академічні (попередження; незарахування роботи; повторне проходження оцінювання; повторне проходження відповідного освітнього компонента освітньої (освітньо-наукової) програми); – дисциплінарні (догана, письмове попередження, відрахування з ДДМА) та ін.

7.4 Регламенти і процедури виявлення порушень вимог академічної доброчесності та наслідки такого виявлення

Основні норми і процедури дотримання академічної доброчесності викладено у «Положення про запобігання та виявлення академічного плагіату у ДДМА (тимчасове)» та «Порядок перевірки на плагіат». З положеннями відповідно можливо ознайомиться за посиланням (http://surl.li/stzpv) та (http://surl.li/tpgzh).

На першому етапі особа, яка відповідальна на кафедрі за перевірку роботи на наявність плагіату, проводить перевірку електронної версії документу на наявність ознак академічного плагіату за допомогою систем StrikePlagiarism.com (http://strikeplagiarism.com) (далі – Системи), використання яких регламентується відповідними угодами Академії. Система формує Звіт подібності, що містить інформацію, яка вказує на наявність текстових та інших запозичень зі знайдених джерел.

Така відповідальна особа не дає оцінку змісту наукової роботи, а виконує виключно технічну перевірку. Подальший аналіз Звіту подібності здійснює науковий керівник.

Інтерпретація показників Звіту подібності системи StrikePlagiarism.com та формування Звіту подібності:

У разі відповідності друкованої та електронної версій роботи Системний оператор проводить перевірку її електронної версії на можливу наявність у тексті робіт чужих опублікованих результатів (текстів) без належного посилання на авторів. За результатами перевірки роботи Антиплагіатною інтернет-системою Strikeplagiarism.com Системний Оператор отримує Звіт подібності, що містить інформацію, яка вказує на ймовірність неправомірних запозичень з інших джерел та інші характеристики, а саме:

- Коефіцієнт Подібності 1. Коефіцієнт Подібності № 1 – це значення (у відсотках), що визначає рівень запозичень, знайдених у певних джерелах (базах даних та Інтернеті), які складаються з фрагментів тексту, що містять щонайменше п ять слів (довідково: у більшості мов загальновикористаними є фрази з п'яти і більше слів; перевищення встановленого значення Коефіцієнту Подібності № 1 не є ознакою наявності неправомірних запозичень, але вказує на необхідність додаткової перевірки тексту роботи).

- Коефіцієнт Подібності 2. Коефіцієнт Подібності № 2 – це значення (у відсотках), що визначає рівень запозичень, знайдених у певних джерелах (базах даних та Інтернеті), які складаються з фрагментів тексту, що містять щонайменше 25 слів. Інша довжина фрагменту, відмінна від 25 слів, може

індивідуально визначатися вченою радою факультету (навчально-наукового-інституту) на основі керівних принципів, прийнятих університетом, та рекомендацій, представників компанії StrikePlagiarism.com. (довідково: перевищення встановленого значення Коефіцієнту Подібності № 2 не є ознакою наявності неправомірних запозичень, але вказує на необхідність додаткової перевірки тексту роботи).

- Сигнал „Тривога!”. Сигнал „Тривога!” з являється, якщо є вірогідність прихованого запозичення. Сигнал „Тривога!” – це повідомлення, що вказує на наявність у тексті знаків одного алфавіту, замінених схожими знаками іншого алфавіту. Сигнал „Тривога!” привертає увагу Системного Оператора до можливої необґрунтованості використання зазначених символів, тобто на можливу спробу фальсифікувати результат перевірки з метою збільшення показників оригінальності роботи.

- При перевищенні допустимих значень Коефіцієнтів Подібності, а також при сигналі „Тривога!” робота підлягають перевірці фахівцем із тематики роботи.

Рекомендовані показники оригінальності за Коефіцієнтом Подібності № 1 становлять: – не більше 20% – оригінальна робота (позитивний висновок), – від 21% до 50% – задовільна оригінальність, – більше 50% – умовно оригінальна робота.

Якщо в Звіті Подібності коефіцієнт подібності № 2 перевищує 5% або наявний сигнал «Тривога!», то робота вважається умовно оригінальною (при будь-якому значенні Коефіцієнту № 1) і підлягає перевірці згідно з цим Порядком.

На підставі Звіту Подібності Системний Оператор готує Протокол контролю оригінальності роботи (Додаток Е).

При отриманні позитивного висновку про відсутність неправомірних запозичень у роботі (робота визнана оригінальною), Протокол надається завідувачу кафедри для проведення попереднього захисту роботи. Якщо за результатами перевірки курсової роботи робота оцінюється як задовільно оригінальна або умовно оригінальна, Системний Оператор протягом двох наступних робочих днів готує повний Звіт Подібності, який направляється науковому керівнику та завідувачу кафедри. Якщо робота класифікована як задовільно оригінальна, то в її тексті перевіряється наявність та правильне оформлення цитувань та посилань на першоджерела.

Висновок щодо можливості попереднього захисту задовільно оригінальної роботи приймають завідувач кафедри разом із науковим керівником роботи протягом трьох робочих днів із моменту отримання Звіту Подібності (Додаток Д). Висновок оформлюється у двох екземплярах. Один екземпляр Висновку видається студенту, другий залишається в документах кафедри.

Якщо прийнято рішення про неможливість допустити задовільно оригінальну роботу до попереднього захисту, студент має право протягом трьох робочих днів подати на кафедру доопрацьований текст роботи. Після належного оформлення проводиться попередній захист роботи.

Якщо робота визнана умовно оригінальною, то члени кафедри разом із науковим керівником роботи мають проаналізувати: – наявність у роботі великих фрагментів тексту, що ідентифіковані системою як подібні, – наявність детальної подібності роботи та джерела (джерел), розташованого в мережі Інтернет та/або базах даних, – можливість кваліфікувати особливості викладу тексту роботи як механічне переписування вже існуючого документа.

Висновок кафедри щодо можливості захисту умовно оригінальної роботи має бути зроблений протягом двох робочих днів із моменту отримання Звіту Подібності. Висновок кафедри оформлюється у двох екземплярах відповідно до форми, наведеної у Додатку Д. Один екземпляр Висновку видається студенту, другий залишається у документах кафедри.

Якщо прийнято рішення про неможливість допустити умовно оригінальну роботу до попереднього захисту, студенту дається п ять робочих днів на корегування тексту роботи. Після отримання Системним Оператором оновленого варіанту роботи проводиться повторна перевірка відповідно до цього Порядку. Якщо за результатами повторної перевірки робота оцінюється як умовно оригінальна або неоригінальна, то ця робота не допускається до захисту в Екзаменаційній комісії та не завантажується до бази даних системи. Висновок кафедри про це повинен бути наданий студенту протягом двох робочих днів. Декану факультету подається витяг із протоколу засідання кафедри про недопуск курсової роботи до захисту. Студент підлягає відраховуванню з Академії як такий, що не виконав навчальний план.

Порядок подання апеляції та її розгляд проводиться відповідно «Положення про запобігання та виявлення академічного плагіату у ДДМА (тимчасове)» та «Порядок перевірки на плагіат». З положеннями відповідно можливо ознайомиться за посиланням (http://surl.li/stzpv) та (http://surl.li/tpgzh)

Порядок подання апеляції та її розгляд проводиться:

- У випадку незгоди з висновком щодо виявлення факту плагіату (компіляцій) у роботі, автор має право у триденний термін з моменту виявлення подати письмову апеляційну заяву на ім’я завідувача кафедрою.

- Для розгляду апеляційної заяви студента створюється апеляційна комісія, персональний склад якої формується розпорядженням завідувача кафедрою з найбільш досвідчених та авторитетних викладачів кафедри (загальний склад від 3 до 5 осіб). Секретарем комісії призначається працівник кафедри.

- Голова апеляційної комісії проводить засідання у тижневий термін з моменту виходу розпорядження завідувача кафедрою про створення апеляційної комісії. Про дату та час проведення засідання заявник попереджається щонайменше за два дні. Якщо заявник не з’являється на засідання апеляційної комісії, питання розглядається за його відсутності.

- У випадку необхідності отримання додаткової уточнюючої інформації, засідання апеляційної комісії може проводитись у кілька етапів з розривом не більше трьох робочих днів.

- Сумніви, що виникають у членів апеляційної комісії, трактуються на користь особи, робота якої розглядається апеляційною комісією.

- За результатами засідання апеляційна комісія формує висновки, які підписує голова апеляційної комісії, її члени та заявник, зазначаючи «З висновками апеляційної комісії погоджуюсь». Висновки апеляційної комісії щодо академічного плагіату (компіляцій) у творах студентів зберігаються на кафедрі.

7.5 Критерії оцінювання курсової роботи

Курсова робота є самостійним теоретико-прикладним дослідженням здобувача вищої освіти, що виконується ним на етапі здобуття повної вищої освіти, що засвідчує професійну зрілість випускника, виявити його загальнотеоретичну та спеціальну підготовку, уміння застосовувати здобуті знання для розв’язання конкретних практичних завдань і, відповідно, готовність до самостійної професійної діяльності.

При виставлянні підсумкової оцінки враховується попередні оцінки, виставлені керівником. Підсумкова оцінка може не збігатися з попередніми оцінками роботи.

Підсумкову оцінку курсової роботи комісія, з спеціальності 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка», яка враховує такі чинники:

1. Достатній теоретичний рівень розробки проблеми.
2. Актуальність проведеного дослідження та впровадження системи автоматизації.
3. Зв'язок теоретичних положень, розглянутих у роботі, із практикою.
4. Наявність елементів самостійної творчості:

– формулювання й обґрунтування власного підходу;

– самостійність аналізу матеріалу;

– повнота й системність пропозицій по розглянутій проблемній ситуації;

– самостійний вибір і обґрунтування теоретичної бази досліджень;

– самостійне формулювання висновків за результатами проведеного дослідження та практичної реалізації системи автоматизації.

5. Використання оригінальних джерел аналітичного й статистичного характеру.
6. Збалансована комбінація кількісних і якісних методів аналізу.
7. Повнота розв'язку поставлених у роботі завдань.
8. Грамотність і логічність у викладі матеріалу.
9. Якість оформлення роботи.

Комісія оцінює всі етапи захисту:

– презентацію результатів роботи;

– розуміння питань, що задаються членами комісії, і якість відповідей на запитання;

– уміння вести професійну дискусію;

– кваліфікацію й загальний рівень розуміння дослідженої проблеми, продемонстровані студентом у процесі захисту;

– загальний рівень культури спілкування з аудиторією.

Критерії оцінювання курсової роботи та її захисту наведені у табл. 7.1. Роботи, за якими визначено, що вони виконані без дотримання вимог академічної доброчесності, не оцінюються і до захисту не допускаються.

Таблиця 7.1 – Критерії оцінювання курсової роботи

Критерії оцінювання курсової роботи	Оцінка
Текст роботи свідчить про оволодіння навичками самостійного (під керівництвом викладача) проведення роботи: відбір і аналіз літератури, узагальнення і творче осмислення теоретичних основ вирішення проблеми, формулювання висновків. Всі завдання виконані, мета роботи досягнута. Методи дослідження відібрані і застосовані коректно. Сформульовані в роботі пропозиції обґрунтовані і достатні. Текст роботи викладений логічно, послідовно, науково-професійною державною мовою, з коректним використанням професійної термінології. Оформлення роботи цілком відповідає вимогам. Під час захисту роботи доповідь відображала усі її основні положення, висновки і рекомендації. Презентація під час доповіді повністю відповідала її змісту. Під час відповідей на питання здобувач показав повне володіння матеріалом роботи, аргументовано відстоював свої ідеї.	90-100 балів A «Відмінно»
Курсову роботу виконано на високому професійному рівні, вона містить усі необхідні елементи, має практичне значення. Всі завдання роботи виконані, мета досягнута. Висновки та пропозиції у роботі в цілому достатньо обґрунтовані й логічні. Технічні та програмні складові роботи відповідають сучасним вимогам. Доповідь на захисті стисла, логічна, проголошена вільно. Презентація доповіді повністю відповідає її змісту. При відповіді на запитання здобувач вищої освіти в цілому продемонстрував високий рівень володіння матеріалом, однак окремі відповіді не зміг чітко аргументувати.	81-89 балів B «Добре»
Тема роботи в цілому розкрита, але мають місце окремі недоліки непринципового характеру (неповнота теоретичного огляду літературних джерел, запропоновані програмні рішення є неоптимальними, допущені незначні помилки у формулюванні висновків). Текст роботи свідчить про оволодіння навичками самостійного (під керівництвом викладача) виконання курсової роботи: проаналізована предметна область, коректно реалізована сформульована задача автоматизації, причинно-наслідковий зв’язок між результатами аналізу, висновками і пропозиціями не порушений. На захисті доповідь логічна, проголошена вільно, але затягнута і містить несуттєві проміжні результати і	75-80 балів C «Добре»

Критерії оцінювання курсової роботи	Оцінка
подробиці. Презентація доповіді в цілому відповідає її змісту, але має недоліки оформлення. Відповіді здобувача на запитання членів комісії загалом вірні, але недостатньо конкретні та/або неповні.	
В цілому завдання роботи виконані й мета досягнута. Текст роботи викладений логічно, послідовно, науково-професійною державною мовою з коректним використанням професійної термінології. В оформленні роботи допущені незначні помилки. Згідно із змістом теми курсової роботи загалом розкрита, але є зауваження змістовного характеру (проаналізовано недостатньо джерел інформації, не в повному обсязі реалізовані поставлені задачі, недостатньо обґрунтовані запропоновані рішення; висновки і пропозиції не повністю відповідають завданням тощо). Здобувач освіти під час доповіді недостатньо розкрив усі суттєві положення роботи, презентація доповіді не повністю відповідає її змісту та/або має вади оформлення. Здобувач під час захисту не завжди міг відповісти на запитання по суті роботи, аргументувати свої відповіді.	61-74 бали D «Задовільно»
В основному завдання роботи виконані й мета досягнута. Текст роботи свідчить про помилки в оволодінні навичками самостійного (під керівництвом викладача) проведення роботи: аналіз предметної області недостатньо повний для вирішення поставлених завдань; при реалізації задач автоматизації проблеми допущені помилки. Запропоновані рішення щодо технічного та програмного забезпечення системи автоматизації недостатньо обґрунтовані, при їх реалізації допущені помилки. Доповідь під час захисту не була достатньо чіткою, побудована недостатньо логічно і послідовно та/або не повністю відображала всі суттєві результати, висновки і пропозиції. Презентація до доповіді оформлена зі значними недоліками, неповна або містить матеріал, який не ілюструє тези доповіді. Здобувач демонструє суттєві труднощі з аргументацією власних ідей, недостатньо володіє професійною термінологією, на значну кількість запитань не може дати відповідь.	55-60 балів E «Задовільно»
Текст роботи свідчить про значні прогалини в оволодінні навичками самостійного (під керівництвом викладача) виконання роботи: аналіз предметної області недостатньо повний для вирішення поставлених завдань; при постановці задач автоматизації проблеми та їхній реалізації допущені помилки. Сформульовані в роботі пропозиції щодо технічного та програмного забезпечення системи автоматизації є недостатньо обґрунтованими і неповними. Окремі завдання	35-55 балів FX «Незадовільно»

Критерії оцінювання курсової роботи	Оцінка
роботи виконані, але мета досягнута не повністю. Текст роботи викладений недостатньо логічно і послідовно, містить стилістичні помилки, використання професійної термінології не завжди коректне. В оформленні роботи є суттєві невідповідності вимогам. Оцінка рецензента негативна. Під час захисту здобувач освіти у доповіді не зміг розкрити результати аналізу, аргументувати висновки і пропозиції, погано знає матеріал роботи і погано володіє професійною термінологією. Презентація до доповіді оформлена зі значними недоліками, неповна або містить матеріал, який не ілюструє тези доповіді. Здобувач не зміг відповісти на переважну кількість запитань комісії.	
Текст роботи свідчить про значні прогалини в оволодінні навичками самостійного (під керівництвом викладача) проведення дослідницької роботи: відбір і аналіз літератури недостатньо повний для вирішення поставлених завдань, аналіз предметної області недостатньо повний для вирішення поставлених завдань; при постановці задач автоматизації проблеми та їхній реалізації допущені помилки. Сформульовані в роботі пропозиції щодо технічного та програмного забезпечення системи автоматизації є недостатньо обґрунтованими і неповними. Окремі завдання роботи виконані, але мета досягнута не повністю. Текст роботи викладений недостатньо логічно і послідовно, містить стилістичні помилки, використання професійної термінології не завжди коректне. Відсутні логічна побудова роботи, її системність та глибина дослідження. Оформлення роботи не відповідає вимогам Оцінки наукового керівника і рецензента негативні. Робота до захисту не допускається.	0-34 бали F «Незадовільно»

7.6 Порядок оскарження результатів оцінювання курсової роботи

У разі незгоди з оцінкою здобувач вищої освіти має право подати апеляцію відповідно п.5. «ПОЛОЖЕННЯ про екзаменаційні комісії Донбаської державної машинобудівної академії з атестації здобувачів вищої освіти» (http://surl.li/tppze). Апеляція подається особисто здобувачем вищої освіти на ім’я ректора в день проведення захисту випускної роботи або оголошення результатів письмового екзамену, але не пізніше ніж на наступний робочий день після оголошення результатів.

За дорученням ректора у разі надходження апеляції, розпорядженням першого проректора за його головуванням створюється комісія для розгляду апеляції. Апеляція розглядається апеляційною комісією не пізніше ніж на наступний робочий день після її подання. Висновки апеляційної комісії оформлюються відповідним протоколом. У разі встановлення апеляційною

комісією порушень під час проведенні атестації, які вплинули на результати оцінювання, рішенням екзаменаційної комісії на підставі висновків комісії з розгляду апеляції здійснюється зміна оцінки.

У випадку незгоди з висновком щодо виявлення факту плагіату (компіляцій) у роботі відповідно п.8 «Положення Про запобігання та виявлення академічного плагіату у навчальній та науково-дослідній роботі учасників освітнього процесу у ДДМА» (http://surl.li/stzpv), автор має право у триденний термін з моменту виявлення подати письмову апеляційну заяву на ім'я завідувача кафедрою.

Для розгляду апеляційної заяви студента створюється апеляційна комісія, персональний склад якої формується розпорядженням завідувача кафедрою з найбільш досвідчених та авторитетних викладачів кафедри (загальний склад від 3 до 5 осіб). Секретарем комісії призначається працівник кафедри.

Голова апеляційної комісії проводить засідання у тижневий термін з моменту виходу розпорядження завідувача кафедрою про створення апеляційної комісії. Про дату та час проведення засідання заявник попереджається щонайменше за два дні. Якщо заявник не з'являється на засідання апеляційної комісії, питання розглядається за його відсутності.

У випадку необхідності отримання додаткової уточнюючої інформації, засідання апеляційної комісії може проводитись у кілька етапів з розривом не більше трьох робочих днів.

Сумніви, що виникають у членів апеляційної комісії, трактуються на користь особи, робота якої розглядається апеляційною комісією.

За результатами засідання апеляційна комісія формує висновки, які підписує голова апеляційної комісії, її члени та заявник, зазначаючи «З висновками апеляційної комісії погоджуюсь». Висновки апеляційної комісії щодо академічного плагіату (компіляцій) у творах студентів зберігаються на кафедрі.

Процедура розгляду апеляції проводиться відповідно «Положення про апеляційну комісію Донбаської державної машинобудівної академії» (http://surl.li/tpqdk)

ПЕРЕЛІК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Буч Г. Об'єктно-орієнтоване проєктування з прикладами застосування / Пер. з англ. – К.: Конкорд, 1992. – 519 с.
2. Кватрані Т. Візуальне моделювання за допомогою Rational Rose 2002 та UML / Пер. з англ. – К.: Видавничий дім “Вільямс”, 2003. – 192 с.
3. Розенберг Д., Скотт К. Застосування об'єктного моделювання з використанням UML та аналіз прецедентів / Пер. з англ. – К.: ДМК Прес, 2002. – 160 с.
4. Якобсон А., Буч Г., Рамбо Дж. Мова UML. Керівництво користувача / Пер. з англ. – К.: ДМК, 2000.
5. Новиков Л. Вступ до Rational Unified Process [Електронний ресурс] / novikov@interface.ru.
6. Якобсон А., Буч Г., Рамбо Дж. Уніфікований процес розробки програмного забезпечення. – К.: Пітер, 2002. – 496 с.
7. Booch G. Object-Oriented Analysis and Design with Applications. 3rd ed. – Addison-Wesley, 2007. – 720 p.
8. Quatrani T. Visual Modeling with Rational Rose 2002 and UML. – Addison-Wesley Professional, 2002. – 208 p.
9. Rosenberg D., Scott K. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. – Pearson, 2005. – 560 p.
10. Jacobson I., Booch G., Rumbaugh J. The Unified Software Development Process. – Addison-Wesley, 1999. – 463 p.
11. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. – Addison-Wesley, 2003. – 208 p.
12. Rumbaugh J., Jacobson I., Booch G. The Unified Modeling Language Reference Manual. 2nd ed. – Addison-Wesley, 2004. – 768 p.
13. Arlow J., Neustadt I. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design. 2nd ed. – Addison-Wesley Professional, 2005. – 624 p.
14. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. 3rd ed. – Prentice Hall, 2004. – 736 p.
15. Oestereich B. Developing Software with UML: Object-Oriented Analysis and Design in Practice. 2nd ed. – Addison-Wesley Professional, 2002. – 336 p.
16. Rational Software Corporation. Rational Unified Process: Best Practices for Software Development Teams. – Rational Software White Paper, 2003.

ДОДАТОК А. ПРИБЛИЗНА ТЕМАТИКА КУРСОВИХ ПРОЄКТІВ

1. Розробити систему автоматизації роботи ветеринарної клініки, а також інтегровану реляційну базу даних для зберігання і обробки даних.
2. Розробити систему складського обліку, що забезпечує управління товарами, прихід і витрати, а також інтегровану реляційну базу даних для зберігання інформації про товари, постачальників і клієнтів.
3. Розробити онлайнову систему залізничної каси для продажу квитків на пасажирські потяги з обліком клієнтів, поїздів, розкладу, тарифів, купівлі та повернення квитків, а також інтегровану реляційну базу даних для зберігання і обробки відповідної інформації.
4. Розробити систему залізничного терміналу для відображення розкладу, продажу, друку та повернення квитків із взаємодією з сервером залізниці, який містить реляційну базу даних для зберігання інформації про розклад, тарифи, місця та квитки.
5. Розробити систему авіаційного терміналу для відображення розкладу, продажу, друку та повернення квитків із взаємодією з сервером авіакомпанії, який містить реляційну базу даних для зберігання інформації про розклад, тарифи, місця та квитки.
6. Розробити систему управління турнікетами для контролю входу, виходу та оплати проїзду пасажирів у міському транспорті з взаємодією із сервером міського транспорту, який містить реляційну базу даних для зберігання інформації про станції, лінії, турнікети, ціни поїздок, проходи пасажирів та використані карти.
7. Розробити систему терміналу оплати міського транспорту для поповнення транспортних карт пасажирів, запису квитків, обробки оплат та передачі даних на сервер міського транспорту, який містить реляційну базу даних для зберігання інформації про типи квитків, ціни, транспортні карти, стоп-аркуш, термінали та сеанси обслуговування клієнтів.
8. Розробити систему терміналу видачі замовлень для автоматизованого обслуговування клієнтів інтернет-магазину, включаючи видачу, відмову від замовлень, оплату при отриманні та передачу інформації на сервер інтернет-магазину, який містить реляційну базу даних для зберігання відомостей про клієнтів, товари, замовлення та термінали видачі.
9. Розробити систему інтернет-магазину для управління клієнтами, товарами, замовленнями, оплатами, доставкою та відгуками, включаючи інтегровану реляційну базу даних для зберігання інформації про клієнтів, замовлення, товари та їх статуси.
10. Розробити систему піццамата для автоматичного продажу та приготування піц із можливістю налаштування складу, та передачі звітів на сервер мережі піццаматів, який містить реляційну базу даних для зберігання інформації про піци, добавки, ціни, запаси, технічне обслуговування та сеанси продажу.

ДОДАТОК Б. ПРИКЛАД ОФОРМЛЕННЯ ТИТУЛЬНОГО ЛИСТА

Міністерство освіти та науки України

Донбаська державна машинобудівна академія

Кафедра Автоматизації виробничих процесів

КУРСОВА РОБОТА

з дисципліни «Технологія програмування складних систем»

*тема «*Назва теми*»*

Виконав: ст. гр. АВПХХ-Х

Хххххххх Х.Х.

Перевірив:

к.т.н., доцент Хххххххх Х.Х.

Краматорськ, 202Х

ДОДАТОК В. ПРИКЛАД ОФОРМЛЕННЯ ЗМІСТУ

ЗМІСТ

ДОДАТОК Г. ПРИКЛАД ОФОРМЛЕННЯ ЗАВДАННЯ ДО КУРСОВОЇ РОБОТИ

Міністерство освіти та науки України
Донбаська державна машинобудівна академія
Кафедра автоматизації виробничих процесів

ТЕХНІЧНЕ ЗАВДАННЯ
до курсової роботи
“Технологія програмування складних систем”
студенту(ке) групи АВП ХХ-Х спеціальності 174

Хххххххх Х.Х.

Тема курсової роботи: Назва теми

Розробити систему автоматизації роботи ветеринарної клініки, а також інтегровану реляційну базу даних для зберігання і обробки даних.

Система автоматизації роботи ветеринарної клініки доступна для всіх її співробітників, які з її допомогою працюють з даними про ветеринарів, про пацієнтів клініки (домашніх тварин) і про клієнтів клініки (власників домашніх тварин). Співробітник реєстратури за допомогою системи підтримує в актуальному стані дані про ветеринарних лікарів (ф., І., О., Дата народження, стаж, спеціальність або спеціальності, графік роботи по днях тижня, телефон, e-mail). Також система дає можливість співробітникові вводити дані про новий пацієнта (вид, кличка, стать, рік народження, ветеринарна картка з історією хвороби) і його власника (ф., І., О., Адреса, телефон, e-mail, розмір персональної знижки). Передбачається, що в один і той же час клієнт клініки не може бути власником різних домашніх тварин з однаковими кличками. У різних клієнтів клички тварин можуть збігатися.
Лікар за допомогою системи зазначає надані пацієнтам медичні послуги (обстеження, щеплення, процедура, операція) із зазначенням дати і вартості, а також виписані клієнтам рецепти на лікарські засоби. Також за допомогою системи лікар веде історії хвороб своїх пацієнтів. Клієнти звертаються в реєстратуру очно або по телефону для запису на прийом до фахівця. Відомості про призначених прийомах заносяться в систему.

Архів системи містить відомості про минулі прийомах (давністю від півроку і більше), про надані в минулому послуги (півроку і древнє), про клієнтів і пацієнтів, які не зверталися в клініку більше двох років. Переклад даних в архів відбувається автоматично в кінці кожного місяця. Працівник реєстратури може видаляти дані з архіву вказуючи або період часу, або клієнта / пацієнта, до якого вони належать.

Система ветклініки на вимогу користувача формує і видає на друк наступну довідкову інформацію:
тижневий графік роботи ветеринарів;
перелік послуг і рецептів, що відносяться до окремого пацієнта, наданих / виписаних в заданий період часу;
рахунок клієнта клініки на оплату медичних послуг;
рецепт клієнту на виписане для пацієнта ліки.

В ході виконання завдання повинна бути розроблена схема реляційної бази даних системи. Ця БД є частиною системи, а не якимось зовнішнім сховищем даних.

Дата видачі завдання	xx.xx.20xx
Дата закінчення роботи	xx.xx.20xx
Керівник роботи	Xxxxxxx X.X.

ДОДАТОК Д. ЗАЯВА ТА ПРОТОКОЛ ПЕРЕВІРКИ РОБОТИ НА ПЛАГІАТ

ЗАЯВА

щодо самостійності виконання курсової роботи (проекту),

Я, _____, студента __ курсу, групи АВП**-*, ФМ, Донбаської державної машинобудівної академії, заявляю:

моя курсова робота на тему:

«_____

_____

_____.»

представлена для захисту у комісію, виконана самостійно і в ній не міститься елементів плагіату. Всі запозичення з друкованих та електронних джерел, а також із захищених раніше дослідницьких робіт, кандидатських і докторських дисертацій мають відповідні посилання. Я ознайомлений(а) з діючим положенням ДДМА, згідно з яким виявлення плагіату є підставою для відмові в допуску письмової роботи до захисту та застосування дисциплінарних заходів.

Підписуючи цей документ, надаю згоду Власнику бази даних (кафедра АВП) на збір, обробку та використання моїх матеріалів кваліфікаційної роботи з метою перевірки матеріалу в системі та зберіганні роботи в репозитарії кафедри (академії).

Підтверджую, що з Положенням «Протидія плагіату» кафедри ознайомлений.

Матеріали автора також можуть бути використані Власником або третіми особами з метою та на умовах, що визначені відповідно до ст. 32 Закону України «Про вищу освіту» та до Закону України «Про захист персональних даних».

Власник зобов'язується використовувати матеріали справедливо і законно відповідно до законодавства. У разі зміни мети обробки матеріалів Власник має повідомити про це суб'єкта і отримати згоду на обробку його матеріалів у відповідності зі зміненою метою.

Дата: Підпис:

ЕКСПЕРТНИЙ ВИСНОВОК
про перевірку на наявність академічного плагіату у випускній курсовій роботі студента

Комісія з виявлення та запобігання академічного плагіату, перевіривши курсову роботу студента _____,
гр. АВПХХ-Х на тему:

«_____
_____
_____»

прийшла до висновку:

Рівень оригінальності твору станом на «____» _____ 202Х р. є *задовільним (_____)*

Зав. кафедри _____ ПІБ.
Відповідальна особа по кафедрі _____ ПІБ.
Керівник роботи _____ ПІБ

Навчальне видання

КАРТАМИШЕВ Дмитро Олександрович

ТЕХНОЛОГІЯ ПРОГРАМУВАННЯ СКЛАДНИХ СИСТЕМ

Методичні вказівки до виконання курсової роботи з дисципліни «Технологія програмування складних систем»
для здобувачів 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» першого бакалаврського рівня за ОПП «Автоматизація та комп'ютерно-інтегровані технології»

Редактор І. І. Дьякова

Комп'ютерна верстка О. П. Ордіна

125/2025. Підп. до друку . Формат 60 х 84/8.
Папірофсетний. Ум. друк. арк.. Обл.-вид. арк. .
Тираж прим. Зам. №.

Видавець і виготівник
Донбаська державна машинобудівна академія
84313, м. Краматорськ, вул. Академічна, 72.
Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру
серія ДК № 1633 від 24.12.2003