КаМІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до підготовки кваліфікаційної роботи бакалавра
для здобувачів вищої освіти зі спеціальності
121 «Інженерія програмного забезпечення» усіх форм навчання

Черкаси 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до підготовки кваліфікаційної роботи бакалавра
для здобувачів вищої освіти зі спеціальності
121 «Інженерія програмного забезпечення» усіх форм навчання

Затверджено на засіданні
кафедри програмного забезпечення
автоматизованих систем,
протокол № 24 від 22.03.2023 р.,
та Методичною радою ЧДТУ,
протокол № ___ від _____р.

Черкаси 2023

Упорядники: **Голуб С.В.,** д.т.н., професор

Заспа Г.О., к.т.н., доцент

Куницька С.Ю., к.т.н., доцент

Півень О.Б., к.ф.-м.н., доцент

Катаєва Є.Ю. к.т.н., доцент

Салапатов В.І. к.т.н., доцент

Метелап В.В. к.т.н., доцент

Олексюк В.В. PhD., ст.викладач

Рецензент: Рудницький В.М., д.т.н., професор, професор кафедри інформаційної безпеки та комп'ютерної інженерії ЧДТУ.

Методичні рекомендації до підготовки кваліфікаційної роботи бакалавра для здобувачів вищої освіти зі спеціальності 121 «Інженерія програмного забезпечення» усіх форм навчання [Текст] /Упорядники: Голуб С.В., Заспа Г.О., Куницька С.Ю., Півень О.Б. Катаєва Є.Ю., Салапатов В.І., Метелап В.В., Олексюк В.В.; М-во освіти і науки України, Черкас, держ. технол. ун-т. Черкаси: ЧДТУ, 2023. 130 с.

Методичні рекомендації призначені для допомоги студентам при виконанні кваліфікаційної роботи бакалавра та відповідають галузевому стандарту вищої освіти освітнього рівня бакалавра спеціальності 121 «Інженерія програмного забезпечення». Можуть бути корисні для студентів суміжних спеціальностей.

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

Методичні рекомендації створені на базі рекомендацій щодо розроблення стандартів вищої освіти наказ Міністерства освіти і науки України від 01.10.2019 №1254 [1].

Кваліфікаційною роботою бакалавра є закінчена оригінальна робота, яка містить сукупність результатів розробки, визначеній в «Освітньо-професійній програмі» за відповідною спеціальністю, що захищається студентом публічно.

Кваліфікаційна робота бакалавра виконується відповідного до індивідуального завдання, затвердженого на засіданні випускової кафедри. У випадку успішного виконання індивідуального завдання бакалавр подає до екзаменаційної комісії звіт у формі пояснювальної записки, програмних продуктів, технічних пристроїв та інших засобів, що підтверджують та демонструють отримані результати.

Пояснювальна записка до кваліфікаційної роботи бакалавра повинна містити результати теоретичних та експериментальних досліджень, опис нових технологій, методичних прийомів та методик вирішення наукових завдань у предметній області інженерії програмного забезпечення, галузі інформаційних технологій, а також їх теоретичне обґрунтування та впровадження у практику проектування та конструювання програмних продуктів.

В процесі захисту своїх результатів бакалавр повинен довести свою здатність самостійно вести науковий пошук, використовуючи теоретичні знання та практичні навички, бачити професійні проблеми, вміти формулювати завдання розробки, підбирати методи їх вирішення, планувати, організовувати і проводити наукове розробки, інтерпретувати його результати, формулювати висновки, подавати отримані результати у доступній формі та супроводжувати впровадження наукових результатів у процес проектування та конструювання програмного забезпечення інформаційних систем.

Робота підлягає обов'язковому рецензуванню. Для проведення рецензування, робота надається одному або декільком рецензентам із числа осіб, які не є працівниками кафедри, на якій виконано випускну кваліфікаційну роботу.

Рецензентами можуть бути фахівці-практики, науковці, викладачі закладів вищої освіти (які мають науковий ступінь кандидата або доктора наук) тощо. Рецензент проводить аналіз та надає на кафедру письмову рецензію стосовно зазначеної роботи.

У кваліфікаційній роботі бакалавра неприпустимі порушення етики наукової розробки, серед яких: фальсифікація наукових даних, некоректні запозичення, порушення правил наукового цитування, привласнення чужих наукових ідей, спотворення наукових фактів та ідей інших дослідників та результатів власного розробки, використання ненаукових та сумнівних, з академічної точки зору, джерел інформації та ін.

Метою цих методичних рекомендацій є подання обов'язкових вимог до змісту, структури, оформлення звіту та прилюдного захисту результатів, отриманих при виконанні кваліфікаційної роботи бакалавра (КРБ) перед екзаменаційною комісією.

КРБ повинна мати:

− характер інженерної розробки;

− розв’язок конкретних технічних задач в галузі розробки програмного забезпечення.

За всі відомості, викладені в КРБ, порядок використання при підготовці фактичного матеріалу й іншої інформації, обґрунтованість і вірогідність висновків та положень, що захищаються, несе відповідальність безпосередньо бакалавр - студент КРБ.

До захисту КРБ допускаються студенти, які виконали всі вимоги навчального плану.

Згідно з положенням про перевірку академічних і наукових робіт на плагіат наказ ЧДТУ №200/01 від 23.06.2021 вказано, що: «Експертна комісія повертає на доопрацювання та визначає необхідність повторної перевірки засобами автоматичної перевірки академічних текстів кваліфікаційної роботи, які мають некоректно оформлені запозичення». У разі виявлення плагіату, свідомого

зловживання авторськими правами іншого вітчизняного чи зарубіжного науковця, КРБ може знімається з розгляду якщо зазначені зауваження не усунуто.

2 СТРУКТУРА, ОБСЯГ ТА ЗМІСТ КВАЛІФІКАЦІЙНОЇ РОБОТИ

2.1 Структура кваліфікаційної роботи бакалавра

Звіт про кваліфікаційну роботу бакалавра містить пояснювальну записку і засоби підтвердження та демонстрації отриманих результатів – програмний продукт, технічні засоби, протоколи випробувань, акти впроваджень, презентація доповіді тощо.

Пояснювальна записка до кваліфікаційної роботи бакалавра повинна містити:

1 Титульний лист (Додаток А).

2 Завдання на кваліфікаційну роботи бакалавра (Додаток Б).

3 Анотації (українською та англійською мовами).

4 Зміст.

5 Перелік умовних скорочень (за необхідністю).

6 Вступ.

7 Основну частину у формі 3 розділів:

1 Вироблення вимог до розроблюваного програмного забезпечення.

2 Проектування програмного забезпечення.

3 Конструювання та тестування програмного забезпечення.

8 Висновки.

9 Список використаних джерел.

10 Додатки.

До кваліфікаційної роботи додаються:

− друкований відгук керівника кваліфікаційної роботи;

− друкована рецензія (завірена печаткою установи, де працює рецензент, якщо вона є);

− подання голові Екзаменаційної комісії (готується секретарем ЕК);

− електронна копія роботи у форматі .pdf з типовим ім’ям файлу: наприклад, Bakalavr_Diplom_Kramarenko_2023.pdf.

2.2 Обсяг ПЗ кваліфікаційної роботи бакалавра

Обсяг пояснювальної записки визначається необхідністю описати отримані результати. Здатність формувати звіт про проороблену роботу і отримані результати є складовою фахової підготовки бакалавра.

Зазвичай описати результати бакалаврської роботи вдається мінімум на 40-50 сторінках друкованого тексту формату А4 (210×297 мм) через півтора міжрядкових інтервали, до 30 рядків на сторінці. Текст має бути надрукований у текстовому редакторі Word (Times New Roman) розміром 14 пт. До обсягу КРБ не входять додатки, список використаних джерел, таблиці та рисунки, які повністю займають площу сторінки. Всі сторінки зазначених елементів КРБ підлягають суцільній нумерації.

Загальна кількість сторінок пояснювальної записки розподіляється пропорційно між розділами.

Менший обсяг пояснювальної записки свідчить про недостатньо детальне опрацювання результатів роботи та відсутність однієї із частин звіту.

2.3 Зміст розділів пояснювальної записки кваліфікаційної роботи бакалавра

2.3.1 Назва кваліфікаційної роботи бакалавра

Назва КРБ повинна бути по можливості короткою, відповідати спеціальності та суті вирішеної інженерної задачі, вказувати на мету розробки і його завершеність.

2.3.2 Анотації

Анотації розміщують на першій після завдання сторінці українською і англійською мовами. Номер сторінок, на яких не ставлять, в загальну кількість аркушів вона не входить. Складається вона в останню чергу після закінчення пояснювальної записки. Анотації складаються за формою, яка має такий зміст:

- прізвище та ініціали студента:
- назва КРБ;
- спеціальність (шифр і назва);
- установа, де відбудеться захист;

- місто, рік;
- основні ідеї, результати та висновки КРБ.

Анотація українською та англійською мовами призначена для ознайомлення зі змістом. Вона містить стислу інформацію про роботу, яка дозволяє прийняти рішення про доцільність знайомства з усією роботою. В анотації слід використовувати прості речення, стандартизовану термінологію, використовуючи синтаксичні конструкції, притаманні мові ділових документів, уникати складних граматичних зворотів маловідомих термінів і символів. Текст анотації на пункти не поділяють.

Обсяг анотації не повинен перевищувати однієї сторінки формату А4. Після кожної анотації наводять ключові слова відповідною мовою.

Ключові слова, що є визначальними для розкриття сутності роботи, наводять після тексту анотації. Перелік ключових слів повинен включати від 5 до 10 слів (словосполучень) у називному відмінку. Перелік подається в рядок через кому великими буквами. Перший рядок – з абзацного відступу, вирівнювання «за шириною».

2.3.3 Ключові слова

Ключовим словом називається слово або стійке словосполучення із тексту анотації, яке з точки зору інформаційного пошуку несе смислове навантаження.

Сукупність ключових слів повинна відображувати поза контекстом основний зміст КРБ. Загальна кількість ключових слів має бути не меншою трьох і не більшою десяти.

Ключові слова подають у називному відмінку, друкують у рядок, через кому.

Зміст подають на початку КРБ. Він містить найменування та номери початкових сторінок усіх розділів, підрозділів та пунктів (якщо вони мають заголовок), зокрема вступу, висновків до розділів, загальних висновків, додатків, списку використаних джерел.

2.3.4 Перелік умовних позначень

Перелік умовних позначень, символів, одиниць, скорочень і термінів подається, якщо в КРБ вжито специфічну термінологію, а також використано маловідомі скорочення, нові символи, позначення і т. ін. Перелік може бути поданий у вигляді окремого списку, який розміщують перед вступом.

Перелік треба друкувати двома колонками, в яких зліва за абеткою наводять скорочення, справа - їх розшифрування.

Якщо в КРБ спеціальні терміни, скорочення, символи, позначення і т. ін. повторюються менше трьох разів, перелік не складають, а їх розшифрування наводять у тексті при першому згадуванні.

2.3.5 Вступ

Структура розділу «Вступ» містить послідовність пунктів що надають загальну характеристику КРБ в наданій нижче послідовності:

- актуальність теми;
- мета і завдання розробки;
- об’єкт розробки;
- методи проектування та конструювання;
- опис отриманих результатів;
- практичне значення отриманих результатів;
- особистий внесок автора.

У пункті «Актуальність теми» обґрунтовують актуальність і доцільність роботи для розвитку відповідної галузі науки чи виробництва шляхом аналізу та порівняння з відомими розв'язками задачі.

Наприклад. Захист програмного забезпечення від порушення цілісності знаходиться в ряді пріоритетних завдань в області інформаційних технологій. Захист програмного забезпечення від модифікації ми захищаємо його від злому. До програм, які потребують захист, підпадають операційні системи, мережеві й комунікаційні програми, текстові й графічні редактори, ігрові модулі,

компілятори тощо. На даний час не існує повністю надійних і універсальних рішень проблеми захисту. Саме тому обрана тема розробки на сьогодні є актуальною.

У пункті «Мета і завдання розробки формулюють мету роботи і завдання, які необхідно вирішити для досягнення поставленої мети.

Пункт «Об'єкт розробки» характеризує процес або явище, що породжує проблемну ситуацію обрану, для вивчення, характеризує процес або явище яке міститься в межах об'єкта. Саме на предмет спрямовується основна увага, оскільки предмет розробки визначає тему КРБ.

Наприклад. Моделі роботи програмної системи підтримки асинхронної дистанційної взаємодії в мережі Інтернет та підходи до їх розробки Інтернет [2].

Пункт «Методи розробки» подають перелік використаних методів розробки для досягнення поставленої в КРБ мети. Перераховувати їх треба не відірвано від змісту роботи, а коротко та змістовно визначаючи, що саме досліджувалось цим методом. Це дасть змогу пересвідчитися в логічності та прийнятності вибору саме цих методів.

Наприклад. Для досягнення поставленої мети застосовано такі методи: порівняння і спостереження — під час вивчення існуючих підходів до створення ПСПАДВІ та моделювання їх роботи; абстрагування, емпіричний аналіз, ідеалізація, індукція, моделювання, системний підхід та формалізація — при побудові формальної моделі ПСПАДВІ і розробки властивостей цієї моделі; експеримент і синтез — при побудові прототипу реалізації ПСПАДВІ і перевірці того, що вона має властивості побудованої моделі [2].

У пункті «Практичне значення одержаних результатів» подаються відомості про практичне використання результатів досліджень або рекомендації щодо їх використання, а в КРБ, що має прикладне значення, - відомості про практичне застосування одержаних результатів або рекомендації, як їх використати. Відзначаючи практичну цінність здобутих результатів, необхідно подати інформацію про ступінь їх готовності до використання або масштабів використання.

Необхідно дати короткі відомості щодо впровадження результатів досліджень із зазначенням назв організацій, в яких здійснена реалізація, форм реалізації та реквізитів відповідних документів.

Наприклад.

1. Розроблено математичне та програмне забезпечення функціонування окремих модулів системи підтримки прийняття рішень формування територіальних громад на основі ройових та генетичних алгоритмів [3];
2. Модифіковано алгоритм Пріма для вирішення завдань моделювання планування ремонту доріг у межах територіальної громади [3];
3. Розв'язано задачу планування ремонту адміністративних будівель у межах територіальної громади на основі зведення задачі планування до задачі динамічного програмування; - розроблено та впроваджено окремі модулі системи підтримки прийняття рішень формування та розвитку територіальних громад [3].

2.3.6 Розділ 1. Існуючі методи та засоби розв'язання поставлених завдань

У першому розділі проводиться аналіз наукової та технічної інформації щодо теми розробки, при цьому бакалавр досліджує матеріал у відкритому доступі – наукові та популярні статті, дисертації, підручники, монографії, Інтернет-сайти, обов'язково із посиланнями на джерела.

У цій частині бакалаврантом конкретизуються завдання КРБ, конкретизація завдань роботи, актуальні проблеми, що виникають в процесі виконання завдань методи та засоби, які вже використовуються для усунення проблем та виконання завдань. Актуальні проблеми, що виникають в процесі виконання завдань і відбувається пошук методів та засобів, які вже використовуються для усунення проблем та виконання завдань. Відбувається систематизація отриманих даних, описується досягнутий результат.

Чим ширше і різноманітніше коло наукової літератури, які використовував студент для огляду та аналізу, тим вищою є теоретична та практична цінність його розробки. Після конспектування матеріалу необхідно перечитати його ще раз, щоб склалося цілісне уявлення про предмет вивчення. Опрацювання

історіографії питання найкраще розпочати з праць, де проблема відображається в цілому, а потім перейти до більш спеціалізованих досліджень.

Стисло, критично висвітлюючи роботи науковців, студент повинен назвати ті питання, що залишились невирішеними, і визначити своє місце у розв'язанні задачі.

Також в першому розділі обов’язковий аналіз існуючих в світі аналогів інформаційних технологій (програмного забезпечення), що стосуються теми розробки.

Закінчують розділ коротким висновком стосовно необхідності проведення досліджень у даній галузі.

Кожна цитата, має супроводжуватися точним описом джерела з позначенням сторінок, на яких опубліковано цей матеріал. Застосування, цитат, коли думки іншого автора видаються за особисті, розглядається як грубе порушення академічної доброчесності, кваліфікується як плагіат.

Обсяг першого розділу має бути не більше 20% обсягу кваліфікаційної роботи бакалавра (тобто 8-12 сторінок).

2.3.7 Розділ 2. Впровадження результатів досліджень у практику проектування програмного забезпечення інформаційних систем

Структура другого розділу містить послідовність підрозділів, що дозволяють розробити методику впровадження отриманих у 2 розділі результатів у практику проектування програмних комплексів [4].

2.3.7.1 Моделювання предметної області

Необхідність аналізу предметної області до початку написання програми була усвідомлена давно при розробці масштабних проектів. При цьому предметна область включає тільки ті об’єкти і взаємозв’язки між ними, які потрібні для опису вимог і умов розв’язання деякої задачі. Сам процес виділення або ідентифікації компонентів предметної області називається концептуалізацією предметної області.

Тобто, щоб створити якісну ІС, не досить зрозуміти бізнес-процеси і потреби замовника. Важливо розуміти, якою саме інформацією система повинна

управляти. А для цього треба знати, які об’єкти потрапляють у предметну область ІС і які логічні зв’язки між ними існують.

Метою побудови моделі предметної області є отримання графічного представлення логічної структури досліджуваної предметної області. Логічна модель предметної області ілюструє сутності у формі класів, а також їх взаємовідносини між собою.

Побудова моделі предметної області розпочинається з виявлення абстракцій, існуючих у реальному світі, тобто тих основних концептуальних об’єктів, які зустрічаються в системі.

Якщо використати ітеративний процес розробки програмного забезпечення (наприклад, Rational Unified Process – RUP або дуже компактну методологію екстремального програмування – eXtreme, XP-процес), то кожній ітерації відповідатиме своя модель предметної області, що відображає сценарії прецедентів, які реалізуються на цьому етапі. Таким чином, модель предметної області еволюціонує у процесі розробки системи.

2.3.7.1.1 Предметна область моделювання. Модель предметної області. Словник предметної області

Модель предметної області – це найважливіша модель об’єктно-орієнтованого аналізу, яка відображає візуальне подання концептуальних класів або об’єктів реального світу в термінах предметної області, а не програмні компоненти. Такі моделі також називають концептуальними моделями або моделями об’єктів предметної області. Це свого роду словник основних абстракцій, тобто найважливіших іменників у просторі задачі.

Іменники, які описують поняття з предметної області, називають *доменними об’єктами*. На самому початку аналізу і проектування необхідно створити модель предметної області, в якій всі доменні об’єкти будуть зображені на одній великій діаграмі класів.

У термінології UML модель предметної області – це діаграма класів. Зазвичай в цій моделі опускається велика частина деталей, зокрема атрибути та операції (методи) класів. Ось чому можна вважати, що модель предметної області

є зведеною діаграмою класів, як візуальний словник важливих абстракцій або візуальний словник предметної області.

Словник предметної області – часто згадуване поняття об'єктно-орієнтованого аналізу і проектування. Усі теоретики об'єктно-орієнтованого підходу стверджують, що словник термінів предметної області – найважливіший артефакт, що лежить в основі аналізу і проектування, яким автори прецедентів можуть користуватися на більш пізніх етапах.

2.3.7.1.2 Елементи моделювання предметної області

У об'єктно-орієнтованих системах структура коду визначається класами (узагальненими об'єктами). Класифікація означає, що об'єкти з однаковими структурами даних (атрибутами) і поведінкою (операціями) групуються у класи. Тому, перш ніж приступати до кодування, треба з'ясувати, які класи знадобляться. З цією метою слід намалювати одну або декілька діаграм класів. Насправді, це перший крок до отримання справжньої діаграми класів, при якій потрібно представити систему в цілому. Потім у процесі роботи над прецедентами поступово уточняються діаграми і врешті-решт буде отримано детальну статичну модель системи.

Модель предметної області повинна бути побудована до складання варіантів використання, щоб уникнути дублювання понять. Якщо схожі назви потраплять в тексти варіантів використання, то знайти їх там набагато складніше, ніж у словнику проекту. Модель предметної області, яка є діаграмою класів аналітичного рівня, – перше наближення до статичної структури системи.

Вимоги до системи зазвичай виглядають як фрази, побудовані у вигляді тверджень: система повинна робити те і не повинна робити це. Тобто, кращими джерелами класів, мабуть, є високорівневий опис завдання, низькорівневі вимоги, описи прецедентів, і експертна оцінка завдання. У міру уточнення даного переліку необхідно враховувати, що:

- іменники та іменні групи стають об'єктами і атрибутами;
- дієслова і дієслівні групи стають операціями і асоціаціями;
- родовий відмінок показує, що іменник має бути атрибутом, а не об'єктом.

Для моделювання предметної області та при проектуванні системи в цілому будемо використовувати мову UML Для діаграм мови UML існують три типи візуальних позначень: графічні символи, зв’язки (єднальні елементи), текст (рисунок 2.1-2.2).

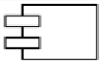
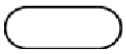
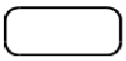
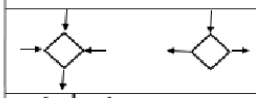
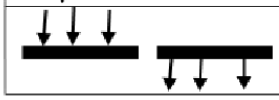
Графічний символ	Назва елемента
	дійова особа (актор)
	варіант використання
	коментар
	пакет
	клас
	об'єкт
	компонент
	вузол
	діяльність
	стан
	лінія життя, фокус управління
	початковий і кінцевий стан
	логічний з'єднувач, розгалужувач
	паралельний з'єднувач, розгалужувач

Рисунок 2.1 – Основні графічні символи UML

Більшість розробників при створенні ПЗ задіють лише невелику підмножину мови UML. Необхідно знайти свою підмножину, яка б підходила для розв’язання вашої задачі.

Графічний символ	Назва елемента
	відношення асоціації
	відношення залежності
	відношення успадкування
	агрегація
	композиція
	приймач події
	джерело події
	заміна
	межа
	простий потік управління
	послати
	виклик
	рекурсія
	повернути

Рисунок 2.2 – Єднальні елементи UML

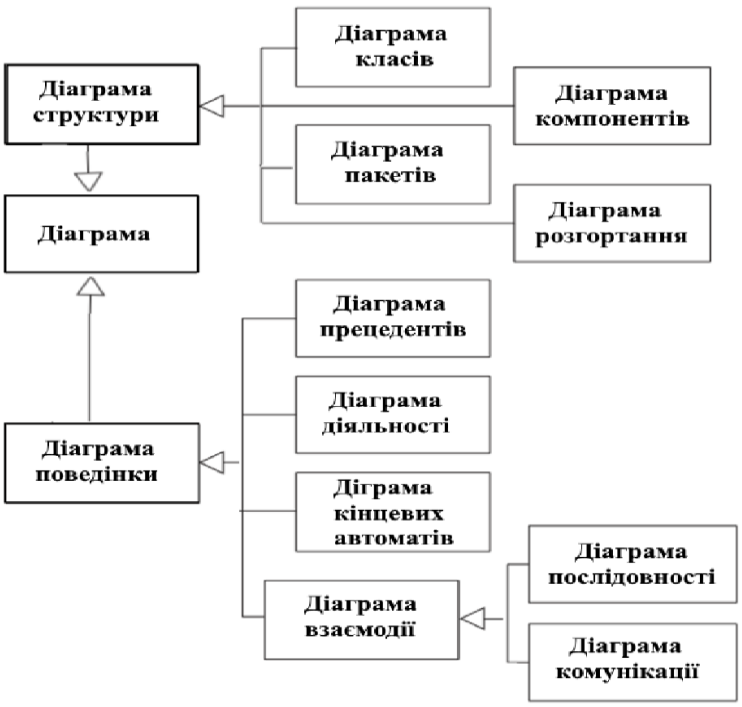


Рисунок 2.3 – Основні діаграми UML

При проектуванні ПЗ кваліфікаційної роботи необхідно використовувати основні діаграми мови UML для моделювання предметної області, формування та аналізу вимог до системи, логічної і фізичної структури та поведінки системи (рисунок 2.3).

У ході виявлення об’єктів з предметної області необхідно встановити, які зв’язки існують між ними. Особливий інтерес представляють два види відношень: узагальнення (відношення між підкласом і суперкласом, відношення виду «є», <|––) й агрегація (відношення між цілим і частиною, відношення виду «має», <>—). Між класами предметної області можуть існувати й інші відношення, у тому числі прості асоціації (——), але ці два відношення дуже важливі. Класи, що відображають модель предметної області, потім увійдуть до основи статичної моделі діаграми класів.

У результаті модель предметної області, доповнена асоціаціями (статичними відношеннями) між парами класів, повинна адекватно описувати аспекти завдання, не залежні від часу (статичні), і в цьому нагадувати діаграми сутність-зв’язок, що використовуються в моделюванні даних.

Необхідно також відвести фіксований час на побудову початкової моделі предметної області. Не потрібно домагатися її досконалості, просто необхідно її закінчити швидше, а потім модифікувати у міру просування вперед.

Класи в UML – це місце для розміщення атрибутів (даних-членів) і операцій (функцій, що виконуються об’єктами). Проте, починаючи моделювати предметну область, не потрібно витрачати багато часу на ідентифікацію атрибутів і операцій; це потрібно буде робити при уточненні і наповненні статичної частини моделі. Зараз же слід сконцентрувати увагу на виявленні власне об’єктів і відношень між ними.

2.3.7.1.3 Робоча область моделювання

Як приклад робочої області моделювання розглянемо internet-магазин, наприклад, з продажу дитячого і дорослого одягу різних брендів. Покупець (клієнт) переглядає каталог і робить замовлення. Припускаємо, що потенційний клієнт заходить на сайт магазину, він може натиснути кнопку перегляду (або

завантаження) каталогу, далі може покласти відібраний товар в кошик, змінити кошик і, прийнявши рішення про купівлю товарів, перейти з кошику до оформлення замовлення [5].

Для того щоб коректно створити систему, що відповідає усім вимогам замовника, необхідно абсолютно чітко уявити собі її основні бізнес-функції і з'ясувати вимоги, що пред'являються до системи. Для цього необхідно провести обстеження компанії і побудувати її повну бізнес-модель. Оскільки наш приклад є придуманим, ми не можемо провести таке обстеження і не маємо можливості спілкуватися із замовником, то ми спиратимемося на придуманий нами словесний опис системи.

Кожен товар в каталозі описується артикулом, розмірним рядом, ціною і фото з коротким описом.

Покупець може завантажити каталог товарів. Каталог складається з набору товарів з фото, ціною і розмірами. Покупець складає вподобані товари в кошик, при цьому вибираючи розмір і кількість необхідного товару цього артикулу.

Кошик можна змінити: проглянути, видалити товар, змінити кількість позицій одного артикулу, повернутися в каталог.

Коли покупець робить замовлення, він вводить свої особисті дані, телефон і оплачує його по банківській картці або готівкою.

Після того як зроблено замовлення, його можна забрати із складу через один робочий день. Дані про замовлення поступають співробітникові магазину (співробітник відділу продаж), він перевіряє наявність товарів і передає його комірникові на комплектацію. Комірник, зібравши замовлення, робить відмітку про готовність.

Замовлення видається зі складу комірником іншою системою, назвемо її Склад. Комірник видає замовлення і відмічає в системі, що замовлення видане.

Оскільки від створення до видачі замовлення воно проходить різні стадії, то введемо поняття статусу замовлення. Співробітники магазину можуть змінювати статус замовлення, а покупець може простежити за складанням замовлення. У такому разі наша система надає ще одну функцію: узнати статус замовлення.

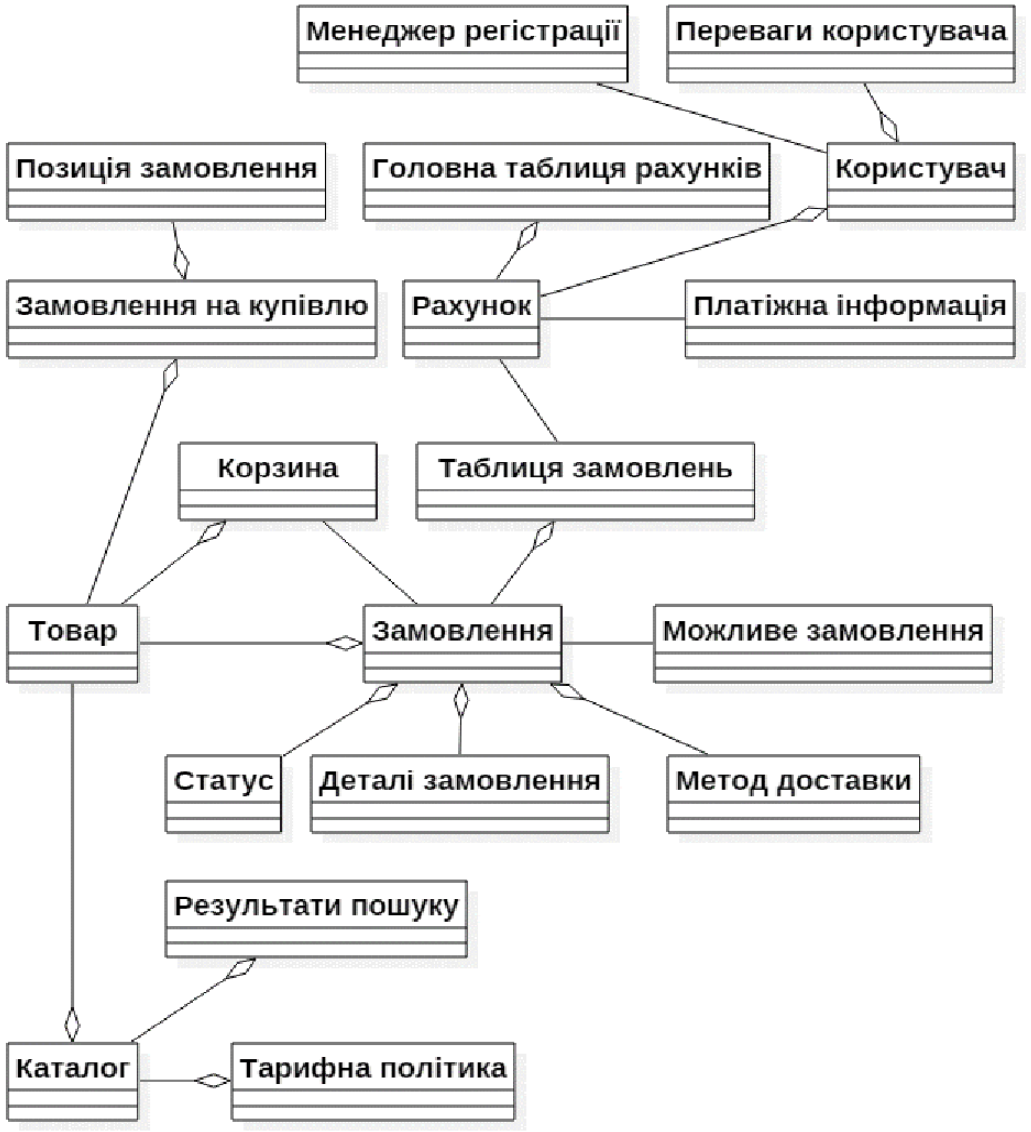


Рисунок 2.4 – Модель предметної області internet-магазину

На рисунку 2.4 наведена повна модель предметної області для internet-магазину. Ті прецеденти і класи, які вибрані із вимог до системи, і які будуть зустрічатися пізніше, покликані задовольнити вимоги замовника (власника магазину). Для відображення класів у діаграмі, окрім стандартного позначення класу (прямокутник, розділений на три частини: назва класу, дані-атрибути і операції представлені пустими клітинами), можна використати й інші варіанти відображення класів аналізу [6].

2.3.7.2 Формування та аналіз вимог

2.3.7.2.1 Формування вимог до програмного забезпечення. Первинні і детальні вимоги. Вимоги замовника і розробника. Функціональні та нефункціональні вимоги

Для побудови чогось, передусім, необхідно зрозуміти, чим це повинно бути і що робить система. Тому процес розуміння та його документування називається аналізом вимог.

Вимога – це умова чи можливість, якій повинна відповідати система. Основна задача етапу визначення вимог – знайти, обговорити і зафіксувати, що дійсно вимагається від системи у зрозумілій формі як клієнтам, так і членам команди розробників. Результатом аналізу вимог є документ, який називають специфікацією вимог, або специфікацією вимог до програмного забезпечення.

Для моделювання і опису усіх процесів, що протікають під час розробки додатку, застосовують уніфіковану мову моделювання (UML). Усі діаграми можна умовно розділити на поведінкові і структурні.

Поведінкові діаграми відображають процеси, що протікають у модельованому середовищі. Структурні діаграми відображають елементи, з яких складається система. При цьому одні й ті ж типи діаграм можуть використовуватися як для моделювання бізнес-процесів, так і для безпосереднього проектування архітектури.

Кількість діаграм для моделі конкретного додатку не є строго фіксованою. Це означає, що для простих додатків немає необхідності будувати всі без виключення типи канонічних діаграм. Деякі з них можуть бути просто відсутніми у проекті системи, і цей факт не вважатиметься помилкою розробника. Наприклад, модель системи може не містити діаграму розгортання для додатка, який виконуватиметься локально на комп’ютері користувача. Тобто, склад діаграм залежить від специфіки конкретного проекту системи.

Кожна діаграма в нотації мови UML має область змісту для зображення графічних вузлів і шляхів між ними, які є власне елементами моделі в нотації UML. Додатково діаграма може бути розміщена у фрейм (прямокутну рамку) і мати заголовок, як це зображено на рисунку 2.5.

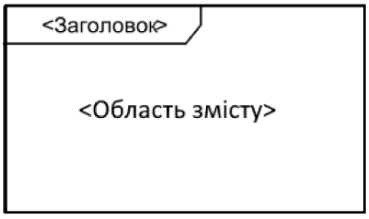


Рисунок 2.5 – Зображення діаграми мови UML 2.0 у вигляді фрейма

Вимогами називають опис функціональних можливостей і обмежень, що накладаються на створювану програмну систему. Вимоги відображають те, що система повинна робити. Тут не намагаються сформулювати, як добитися виконання цих функцій.

Наприклад, можлива така вимога до банківської системи:

Система повинна надати клієнтові можливості виконання таких операцій над його рахунком: перегляд, зняття грошей, додавання грошей.

А ось такий запис, наприклад, вимогою не є:

Інформація банківського рахунку повинна зберігатися у вигляді трьох таблиць СКБД MySQL.

Тут вказується як має бути побудована система, або що вона повинна робити. Втім, можуть бути і виключення з цього правила. Наприклад, у замовника можуть бути особливі причини для такої реалізації.

Розрізняють дві категорії представлення вимог: вимоги замовника (первинні вимоги) і вимоги розробника (детальні вимоги). Відрізняються вони одна від одної мірою опрацювання описів.

Первинні вимоги документують бажання і потреби замовника і пишуться мовою, зрозумілою замовникові. Роботу із створення первинних вимог називають збором або формуванням вимог. На стадії проектування вимоги відображаються діаграмою прецедентів (варіантів використання, Use Case).

Детальні вимоги документуються у спеціальній, структурованій формі, вони деталізовані стосовно до первинних вимог. Роботу зі створення детальних вимог називають аналізом вимог. На стадії проектування детальні вимоги

документуються спеціальною структурованою мовою за допомогою діаграм діяльності та діаграм взаємодії.

Деякі проблеми можуть бути породжені відсутністю чіткого розуміння відмінності між цими категоріями вимог. Пояснимо відмінність між ними. Наприклад, вимога замовника має вигляд:

ВИМОГИ ЗАМОВНИКА

1 ПЗ повинно забезпечити засоби для введення і збереження різноманітних даних абонента-користувача.

А вимоги розробника, що відповідає їй, може записуватися у структурованій формі:

ВИМОГИ РОЗРОБНИКА

1.1 Користувач повинен мати можливість визначати типи даних, що вводяться.

1.2 Для кожного типу даних має бути відповідний засіб, що забезпечує введення і збереження елементу даних цього типу.

1.3 Кожен тип даних повинен представлятися відповідною піктограмою на дисплеї користувача.

1.4 Користувачеві повинна пропонуватися піктограма для кожного типу даних та можливість самостійного вибору піктограми для кожного типу даних.

1.5 При виборі користувачем піктограми типу даних до елементу даних має бути застосований засіб, що асоціюється з вказаним типом.

Розрізняють два види вимог:

1 Функціональні вимоги описують поведінку системи і сервіси (функції), які вона повинна виконувати. При цьому виходять з усебічного аналізу проблемної (предметної) області. Розглядаються різноманітні варіанти поведінки, що визначаються різними даними і станами зовнішнього середовища.

2 Нефункціональні вимоги належать до характеристик системи та її зовнішнього оточення, критеріїв ефективності і безпеки. Додатково можуть перераховуватися обмеження, що накладаються на дії і функції системи, а також на умови розробки.

Формування вимог є лише початковою фазою роботи з вимогами.

Аналіз вимог є мостом між підготовкою, плануванням і проектуванням програмного забезпечення, який розглядає вимоги замовника як вихідні дані, на виході аналізу – вимоги розробника, які справедливо називають детальними вимогами. Тут відбувається перехід зі світу замовника у світ розробника. Змінюється мова запису вимог. Тепер це не природна мова людини, а мова формалізованих моделей. Вона незрозуміла замовнику, але зрозуміла і близька розробнику.

Реалізація процедур цього етапу проектування відбувається у процесі побудови відповідних діаграм мовою UML або іншими мовами, що використовуються системами автоматизованого проектування.

Результати процедур із формування та аналізу вимог оформляються у вигляді діаграми прецедентів (варіантів використання, Use Case), діаграми діяльності, діаграми взаємодій (діаграми послідовностей і комунікації) та їх описів.

Процес побудови кожної з цих діаграм є відображенням результатів процесу одержання певної частки інформації про вимоги до системи.

Детальніше з особливостями використання процесів побудови діаграм прецедентів, діаграм діяльності та діаграм взаємодій (у формі діаграм послідовностей і комунікації) для реалізації етапу формування та аналізу вимог об’єктно-орієнтованої технології проектування інформаційних систем необхідно ознайомитись з посібником з проектування ІС [6].

2.3.7.2.2 Формування вимог за допомогою діаграми прецедентів

Досвід показує, що моделі формування та аналізу вимог повинні доповнювати, а не замінювати специфікації вимог природною мовою (тобто вимоги в текстовому представленні).

У цьому підрозділі необхідно розглянути основне питання, яке необхідно задати на початку розробки будь-якої системи: «Що хочуть її користувачі?». Тут необхідно якомога докладніше уявити дії користувачів і реакцію системи,

оскільки її поведінка зумовлена їхніми вимогами. Іншими словами, робота системи залежить від того, як до неї звертаються і чого хочуть досягти.

Діаграми прецедентів (діаграми варіантів використання, Use Case) дозволяють отримати відмінне візуальне уявлення про вимоги користувачів та поведінку системи з погляду користувача. Діаграма прецедентів розглядається як головний засіб для первинного моделювання динаміки системи, використовується для з'ясування вимог замовника до розроблюваної системи, фіксації цих вимог у формі, яка дозволить проводити подальшу розробку.

Основна ідея в дослідженні і формуванні функціональних вимог полягає в написання історій «з життя системи». Ці історії допомагають сформулювати різні задачі і являють собою сценарії використання системи – це спеціальна послідовність дій або взаємодій між виконавцями і системою.

У процесі опису сценаріїв необхідно задавати питання: «Хто є користувачем системи? Які типові сценарії використання системи? Які цілі й задачі користувачів?». Такий погляд на систему набагато більше орієнтований на користувача, ніж на звичайний список системних вимог, і дозволяє отримати відмінне візуальне уявлення про вимоги користувачів.

Діаграма варіантів використання є відправною точкою у процесі моделювання. Вона призначена для опису взаємодії проектованої системи з будь-якими зовнішніми або внутрішніми об'єктами – користувачами, іншими системами тощо.

До складу діаграм варіантів використання входять елементи Use Case (варіанти використання), актори, а також відносини залежності, узагальнення та асоціації. Як й інші діаграми UML, діаграми Use Case можуть включати примітки і обмеження.

Актор – це роль, яку виконує користувач або інша система, при взаємодії з проектованою системою. Кожен актор має унікальне ім'я. Проектування діаграми варіантів використання розпочинається з визначення списку акторів.

Загальноприйнятим графічним позначенням актора на діаграмах у нотації мови UML являється фігурка «дротяного чоловічка», під якою записується

обов’язкове ім’я актора (рисунок 2.6, а). Актор також може зображуватися символом прямокутника з ключовим словом <<actor>>, записаним у формі стереотипу вище за його ім’ям (рисунок 2.6, б). Додатково в мові UML 2.0 допускається зображувати актора у формі запропонованої розробником піктограми, зовнішній вигляд якої, на думку самого розробника, наочніше ілюструє характер виконуваної ним ролі (рисунок 2.6, в) [7].

Наступним етапом після визначення списку акторів є визначення списку варіантів використання.

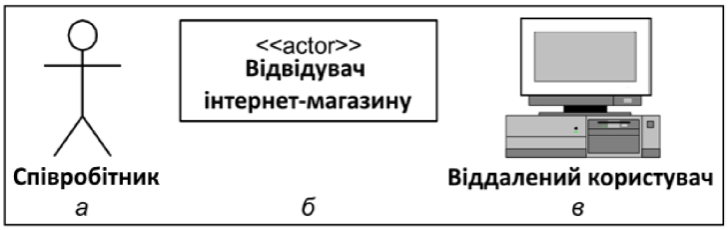


Рисунок 2.6 – Варіанти символів графічного позначення актора:
а – у вигляді «дротяного чоловічка»; б — у вигляді прямокутника;
в – у формі піктограми

Варіант використання – це кінцева одиниця взаємодії актора і системи. Сукупність усіх варіантів використання повністю визначає поведінку системи. Кожен варіант використання відноситься до деякого актора. Таке відношення означає, що цей актор ініціює цей варіант використання.

Залежно від мети виконання процедури розрізняють такі варіанти використання [6, розділ 10.3]:

− основні (базові) – забезпечують необхідну функціональність програмного забезпечення, що розробляється;

− допоміжні – забезпечують виконання необхідних налаштувань системи і її обслуговування (наприклад, архівація інформації);

− додаткові – забезпечують додаткові зручності для користувача (як правило, реалізуються, якщо не вимагають серйозних витрат яких-небудь ресурсів ні при розробці, ні при експлуатації).

Загальноприйнятим графічним зображенням варіанту використання в нотації мови UML є еліпс, усередині якого або нижче його записується ім’я у формі рядка тексту. Ім’я використовується для ідентифікації окремого варіанту використання, яке рекомендується формулювати у формі закінченої пропозиції, що починається, як правило, з іменника або дієслова.

Коментар. Коментар у мові UML призначений для включення в модель довільної текстової інформації у формі примітки, яка може бути приєднана до одного або декількох елементів моделі. Графічно коментар на усіх типах діаграм зображується у формі прямокутника з «загнутим» правим верхнім кутом (рисунок 2.7).

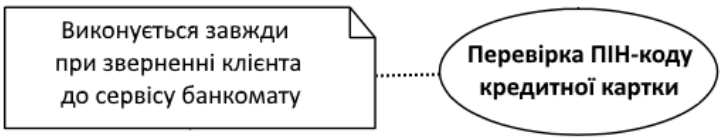


Рисунок 2.7 – Приклад коментарів в мові UML

Відношення асоціації. Асоціація є одним із фундаментальних понять в мові UML і може використовуватися на різних канонічних діаграмах при побудові візуальних моделей. Стосовно діаграм варіантів використання відношення асоціації може служити тільки для позначення взаємодії актора з варіантом використання у вигляді суцільної лінії. Розрізняють асоціації спрямовані (вказується простою стрілкою у формі букви «V») та не спрямовані (напрям взаємодії актора з варіантом використання для розробника не має принципового значення). Приклад графічного представлення відношення спрямованої і неспрямованої асоціації між актором і варіантом використання наведено на рисунок 2.8.

Рисунок 2.8 – Приклад відношення спрямованої і неспрямованої асоціації між актором і варіантом використання

Відношення узагальнення. Відношення узагальнення призначене для специфікації того факту, що один елемент моделі є спеціальним або окремим випадком іншого елементу моделі. Головною особливістю відношення узагальнення є те, що воно може зв'язувати між собою тільки елементи одного типу. Графічно відношення узагальнення позначається суцільною лінією із стрілкою у формі не зафарбованого трикутника, яка вказує на загальний елемент моделі (рисунок 2.9).

Рисунок 2.9 – Приклад графічного зображення відношення узагальнення

Відношення включення. Відношення включення між двома варіантами використання в мові UML є окремим випадком загального відношення залежності (dependency), яке визначається як форма взаємозв'язку між двома елементами моделі. Зміна одного елементу моделі у відношенні залежності призводить до зміни деякого іншого елементу.

У загальному випадку залежність є спрямованим бінарним відношенням, яке зв'язує між собою тільки два елементи моделі – незалежний і залежний.

Відношення включення (include) специфікує той факт, що деякий варіант використання містить поведінку, визначену в іншому варіанті використання.

Графічно це відношення позначається як відношення залежності у формі пунктирної лінії з «V»-образною стрілкою, спрямованою від залежного варіанту використання до незалежного варіанту використання. Залежний варіант використання часто називають також базовим або включаючим, а незалежний – включаємим варіантом використання. При цьому лінія зі стрілкою обов'язково позначається ключовим словом (стереотипом) <<include>> (рисунок 2.10).

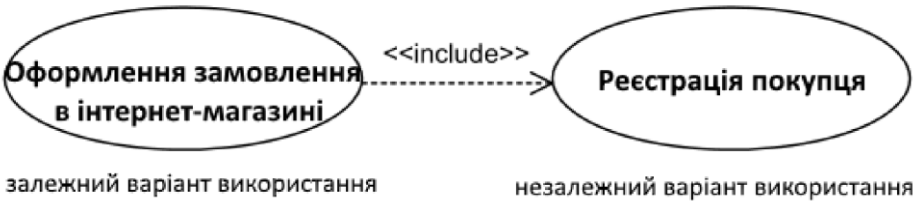


Рисунок 2.10 – Приклад графічного зображення відношення включення

Відношення розширення. Відношення розширення (extend) в мові UML також є окремим випадком загального відношення залежності між двома варіантами використання. Відношення розширення це спрямоване бінарне відношення, що визначає взаємозв'язок одного варіанту використання з деяким іншим варіантом використання, функціональність або поведінка якого задіються першим не завжди, а тільки при виконанні деяких додаткових умов.

Графічно це відношення позначається як відношення залежності у формі пунктирної лінії з «V»-образною стрілкою, спрямованою від залежного варіанту використання до незалежного варіанту використання. При цьому залежний варіант використання відносно розширення називається таким, що розширює, а незалежний – базовим або розширюваним варіантом використання. Лінія зі стрілкою обов'язково позначається ключовим словом <<extend>>, яке записується у формі стереотипу (рисунок 2.11) [8].

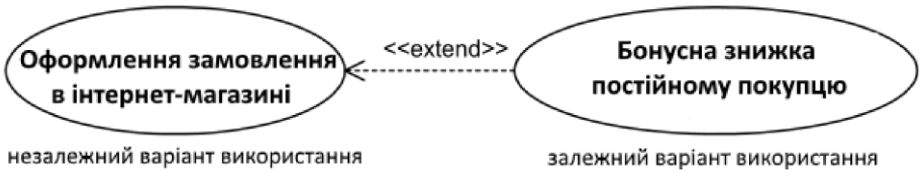


Рисунок 2.11 –Відношення розширення між варіантами використання

Для ілюстрації застосування розглянутих елементів графічної нотації мови UML побудуємо діаграму варіантів використання на прикладі системи продажу товарів в інтернет-магазині (рисунок 2.12) [8].

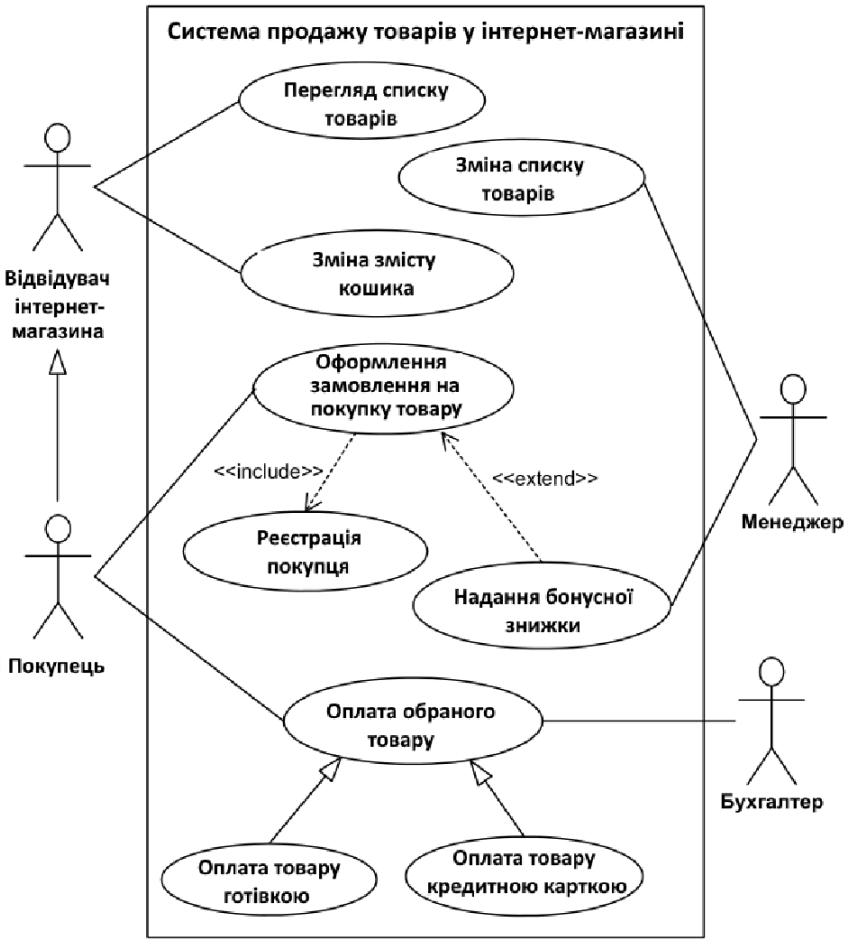


Рисунок 2.12 – Нотація діаграми використання (прецедентів)

В якості основного актора описуваної системи можна розглядати відвідувача інтернет-магазину, який може переглядати список товарів, поміщати товар у віртуальний кошик і змінювати вміст цього кошику. Відвідувач може стати покупцем, якщо він приймає рішення про оформлення замовлення на купівлю вибраних ним товарів.

Іншими акторами даної системи можуть виступати менеджер інтернет-магазину і бухгалтер. При цьому менеджер може змінювати список товарів і специфікувати умови для надання бонусної знижки, а бухгалтер – приймати плату за вибраний покупцем товар.

Найбільш значимим для цієї системи і її акторів прецедентом являється прецедент Замовлення товарів. Для нього побудуємо додаткову діаграму прецедентів, що пояснює цей варіант використання (рисунок 2.13).

Рисунок 2.13 – Діаграма варіантів використання основного прецеденту Замовлення товарів

Детальний опис усіх варіацій діаграм використання наведений в розділі 10.3 роботи [6].

2.3.7.3 Проектування логічної структури програмного комплексу

Якщо формування та аналіз вимог проектування – це розробка відповідної концептуальної моделі, яка описує усі можливі та прийнятні варіанти розв'язання задач, які вказують, що повинна робити система, то ***проектування логічної структури системи*** – це розроблення логічної моделі системи у вигляді моделі класів і пакетів, які вказують з чого складається система. Логічне проектування ще називають ***детальним проектуванням класів***.

2.3.7.3.1 Діаграми класів

Попередні етапи проектування забезпечили **аналіз об'єкта проектування** (формування та аналіз вимог, ідентифікація об'єктів). Але вже на етапі ідентифікації об'єктів вимальовуються архітектурні риси майбутньої системи. Паралельно з аналізом починається етап структурного проектування.

Структурне та архітектурне проектування формують фазу деталізації проектування. Архітектурний скелет доповнюється деталями – класами, що здатні реалізовувати сценарії, які описані діаграмами послідовності.

У процесі реалізації етапу детального проектування орієнтуються на максимальну підготовку до кодування програмної системи. Програмісти повинні

отримати детальні проектні рішення, що забезпечать їх повною інформацією для створення програмного коду.

Фундаментальними поняттями ООАП є поняття класу і об’єкту, а також його основні принципи: абстракція, інкапсуляція, поліморфізм, наслідування. Як відомо, класи можуть застосовуватись на етапі аналізу для складання переліку об’єктів (глосарію системи) або для аналізу предметної області. Але ці класи повинні бути деталізовані до одного або кількох проектних класів на етапі детального проектування. Ця потреба викликана відсутністю у класів на етапі аналізу повного набору атрибутів та операцій.

Таким чином, основними будівельними блоками детального проектування є ***класи***. Результат етапу детального проектування повинен бути відображений діаграмами класів (що містять опис всіх операцій та атрибутів класів), які призначені для статичного моделювання об’єктів.

Діаграма класів описує типи об’єктів системи і різного роду статичні відношення, які існують між ними. На діаграмах класів відображаються також властивості класів, операції класів та обмеження, які накладаються на зв’язок між об’єктами.

Діаграма класів не містить інформації про часові аспекти функціонування системи. Вона призначена для представлення тільки статичної структури моделі системи в термінах базових будівельних блоків і відносин між ними. До решти інших структурних діаграм належать діаграми компонентів і розгортання.

Діаграма класів може містити наступні сутності: класи, відношення (залежності, узагальнення, асоціації, кооперації), інтерфейси, коментарі, обмеження, пакети, підсистеми.

Основні елементи діаграм класів

Об'єкт – це деяка сутність реального світу або концептуальна (абстрактна) сутність. Прикладами об’єктів можуть служити будинок №5 по вулиці Гоголя, співробітник фірми Степан Хоменко, ваш комп'ютер або щось абстрактне: математична формула, торгове замовлення номер 123654, банківський рахунок

клієнта Степана Хоменка. Об’єкт має чіткі певні межі і значення для системи і характеризується станом, поведінкою і індивідуальністю.

Стан об'єкту – це одна з умов, в якій він може знаходитися. Стан зазвичай змінюється з часом і характеризується набором властивостей, які називаються атрибутами. Наприклад, покупець визначається його ім'ям, адресою, телефоном, датою народження.

Поведінка визначає, як об'єкт реагує на запити інших об'єктів і що може робити сам об'єкт. Поведінка характеризується операціями об'єкту. Наприклад, покупець може додати товар в кошик, переглядати каталог, видаляти товар з кошику.

Як правило, в системі існує безліч об'єктів, які мають однакову поведінку, приймають однакові стани. Наприклад, співробітники фірми, яких може бути декілька десятків, і дані про яких містяться у базі даних, мають однакові атрибути (з різними значеннями цих атрибутів) – прізвище, ім’я, по батькові, дату народження, посаду тощо, а також можуть мати схожу поведінку. Для угрупування об'єктів використовуються класи.

Клас – це іменований опис сукупності (групи) об’єктів із загальними властивостями (атрибутами), однаковою поведінкою (операціями) та типами відношень, способів доступу до даних и методів, зв’язками і семантикою.

Кожен клас є шаблоном для створення об'єкту. А кожен об'єкт – це екземпляр класу. Важливо пам'ятати, що кожен об'єкт може бути екземпляром тільки одного класу. Кожен об’єкт має свою індивідуальність, яка передбачає наявність у нього унікального імені, що відрізняється від імен інших можливих об’єктів того ж класу.

Стосовно нашого Internet-магазину ми можемо згрупувати співробітників магазину, описавши загальний для них клас Співробітник. Об'єкт цього класу, наприклад, Степан Хоменко, може включати таку інформацію: ім'я, адреса, посада, розмір заробітної плати, крім того цей об'єкт може вийти у відпустку.

У нотації UML графічно клас зображується у вигляді прямокутника, який додатково може бути розділений горизонтальними лініями на три секції, які

містять ім’я класу, атрибути (змінні) і операції (методи). Ім’я класу унікальне в межах пакету, вказується по центру в першій верхній секції прямокутника напівжирним шрифтом із заголовної букви (рисунок 2.14).

Рисунок 2.14 – Графічне зображення класу і об'єктів

Назви класів вибираються відповідно до понять предметної області. Це повинен бути іменник або словосполучення в однині, що найточніше характеризує предмет. Клас повинен описувати тільки одну сутність. Ім’я класу може бути простим, як це показано на рисунку 2.14, або складеним (рисунок 2.15). Складене ім’я класу складається з самого імені класу і з імені пакету, якому належить клас, розділених двокрапкою. Ім’я класу має бути унікальним усередині пакету.

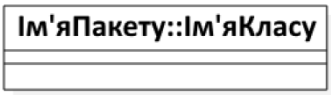


Рисунок 2.15 – Складене ім'я класу

Складене ім’я об’єкту також складається з імені об’єкту і імені класу, розділених двокрапкою. Об’єкт може бути анонімним, якщо невідоме його справжнє ім’я. Тоді на діаграмі об’єкт зображується з ім’ям, яке складається з двокрапки і імені класу, якому належить об’єкт. Якщо доки невідомий клас, екземпляром якого є об’єкт, то зображується ім’я об’єкту після якого йде двокрапка. Такий об’єкт називається «сиротою» (рисунок 2.16).

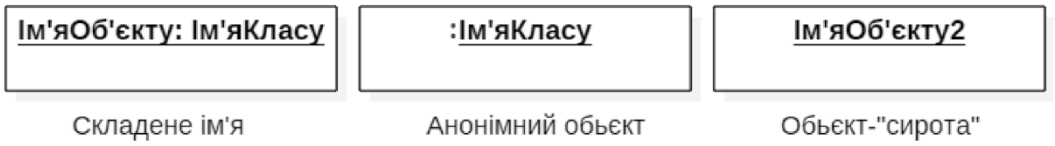


Рисунок 2.16 – Іменування об'єктів

Виявлення класів можна розпочати з вивчення потоку подій. Іменники в описі цього потоку можуть виявитися дійовою особою, класом, атрибутом класу або виразом, що не є ні дійовою особою, ні класом, ні атрибутом класу.

Якщо в ході проектування системи вже були побудовані діаграми взаємодії, то, перед тим, як приступати до побудови діаграм класів, пошукайте на цих діаграмах схожі об'єкти. Наприклад, якщо на діаграмі послідовності, що описує оформлення замовлення об'єктами Івановим і Петровим з однаковими властивостями (ім'я, рахунок у банку тощо), то в системі повинен з'явитися клас з ім'ям **Покупець**, який буде шаблоном об'єктів Іванова і Петрова.

Деякі класи будуть виявлені при розгляді трьох стереотипів: сутність (entity), межа (boundary) і управління (control). Ми вже зустрічалися із стереотипами відношень на діаграмах прецедентів. Той же принцип створення нового типу на основі вже існуючого застосовується і для класів.

Стереотип – це механізм, що дозволяє категоризувати класи. Він використовується для створення нового типу елементу, в даному випадку нового типу класу. Наприклад, якщо необхідно виділити усі екранні форми в моделі, для цього треба створити стереотип Form (Форма). Стереотипи допомагають краще зрозуміти відповідальність кожного класу в моделі, категоризувати виконувані ними функції. У UML для цього застосовують три основні стандартні види стереотипів класів: класи-сутності, граничні класи і управляючі класи.

Клас-сутність містить інформацію, яка зберігається постійно. Використовуються такі класи для моделювання даних і поведінки з довгим життєвим циклом. Вони можуть представляти інформацію про предметну область, а можуть представляти елементи самої системи. Часто будучи абстракціями предметної області, вони мають найбільше значення для

користувача, тому в їх назвах застосовуються терміни предметної області. Позначаються класи-сутності стереотипом <<entity>> або спеціальною піктограмою (рисунок 2.17, *а*).

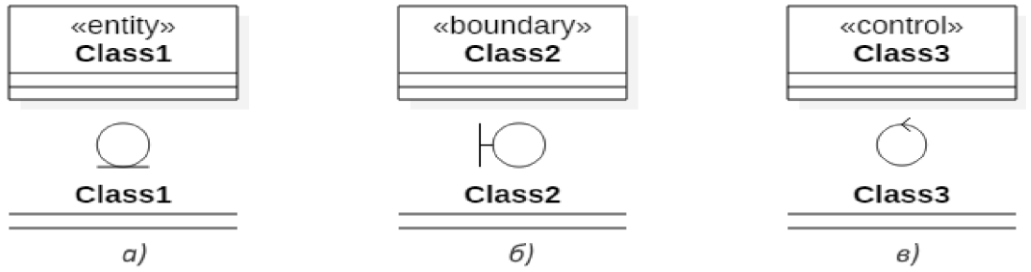


Рисунок 2.17 – Позначення класів-стереотипів

Граничними класами називаються класи, розташовані на межі системи зі всім іншим світом, вони забезпечують взаємодію між довкіллям і внутрішніми елементами системи.

Для виявлення пограничних класів необхідно досліджувати діаграми варіантів використання. Для кожної взаємодії між актором і прецедентом треба створити хоч би один граничний клас. Якщо дві дійові особи ініціюють один прецедент, то вони можуть застосовувати один загальний пограничний клас для взаємодії з системою. Позначаються граничні класи ім'ям стереотипу <<boundary>> або спеціальною піктограмою (див. рисунок 2.17, *б*).

Управляючий клас відповідає за координацію дій інших класів. Вони служать для моделювання послідовної поведінки одного або декількох прецедентів і координації подій, що реалізовують закладену в них поведінку. Позначаються управляючі класи ім'ям стереотипу <<control>> або спеціальною піктограмою (див. рисунок 2.17, *в*).

Управляючий клас делегує відповідальності іншим класам. Сам він може отримувати мало повідомлень, але посилати множину. Його називають класом-менеджером. Він запускає альтернативні потоки і знає, як поступити в разі помилки. На початковому етапі проектування управляючі класи створюються для

кожної пари актор-прецедент, надалі вони можуть об'єднуватися, розділятися або виключатися.

Розглянемо сценарій Оформлення замовлення, яке є внутрішнім потоком для прецеденту Замовлення товару. Опишемо його класи.

Цей сценарій дозволяє покупцеві оформити замовлення з кошику і провести оплату з використанням банківської карти. Покупець, знаходячись у своєму купівельному кошику, прийнявши рішення про те, що він готовий зробити замовлення в Internet- магазині, вибирає опцію «Оформити замовлення». Запускається сценарій **Оформлення замовлення**. Користувач повинен на спеціальній формі внести свої особисті дані, підтвердити замовлення або ні, і зробити оплату, потім отримати підтвердження замовлення. У системі з'являється новий об'єкт – замовлення покупця.

Необхідно вибрати хоч би один граничний клас, який здійснює зв'язок між дійовою особою **Покупець** і додатковим прецедентом **Оформити замовлення**. Назвемо його **ОформленняЗамовлення(PlaceOrder)**. Цей клас знає, які товари і в якій кількості були в кошику покупця, їх треба перенести в замовлення. Також цей клас може знати, порожній кошик покупця чи ні, і, якщо кошик порожній, то вивести про це відповідне повідомлення.

Для того щоб зробити замовлення, покупець повинен ввести свої особисті дані, електронну адресу, телефон і дані кредитної картки. Для цих цілей введемо ще один клас **ВведенняОсобистихДаних (EnterPersonalInformation)**. Після того як вибрані товари і введена особиста інформація покупця, залишається тільки перевірити деталі замовлення і погодитися з ними чи ні. Для цієї дії введемо клас **ПеревіркаДеталейЗамовлення (ConfirmOrder)**. Нарешті, коли покупець завершить оформлення замовлення, то на екрані він побачить номер замовлення і підтвердження, що замовлення послане покупцеві на електронну адресу. Для виконання цих прецедентів створимо ще один граничний клас **ПідтвердженняЗамовлення (OrderConfirmation)**.

Створимо один управляючий клас, який розподілятиме обов'язки інших класів і викликатиме їх операції при виконанні цього сценарію. Назвемо цей клас

МенеджерОформленняЗамовлення (PlaceOrderManager).

Вибір класів-сутностей. У сценарії **Оформлення замовлення** йдеться про покупця, замовлення і товари. Створимо класи-сутності **Покупець (Customer)**, **Замовлення (Order)**, **Товар (Item)**. Можливо, що в ході подальшого проектування якісь нові класи будуть додані для цього сценарію, а якісь, навпаки, з цієї діаграми будуть видалені.

Створимо усі ці класи на діаграмі класів (рисунок 2.18). Краще виконувати діаграми класів і взаємодії відразу з написами англійською мовою, оскільки ці діаграми надалі будуть брати участь в генерації програмного коду системи. Якщо їх виконувати українською мовою, то згенерований код утримуватиме імена на кирилиці, і для подальшого використання програмного коду такі імена необхідно всі замінити.

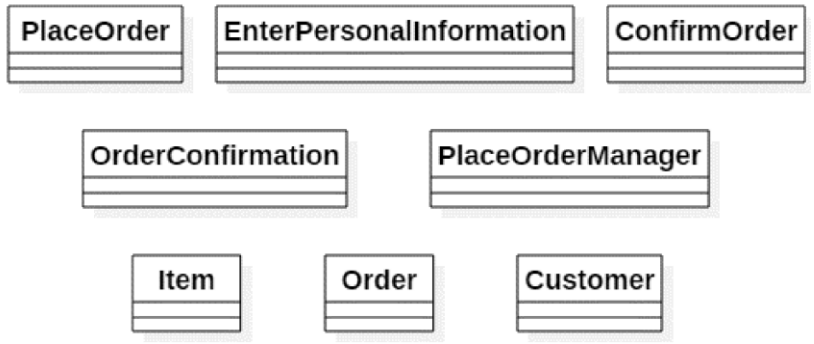


Рисунок 2.18 – Діаграма класів сценарію Оформлення замовлення

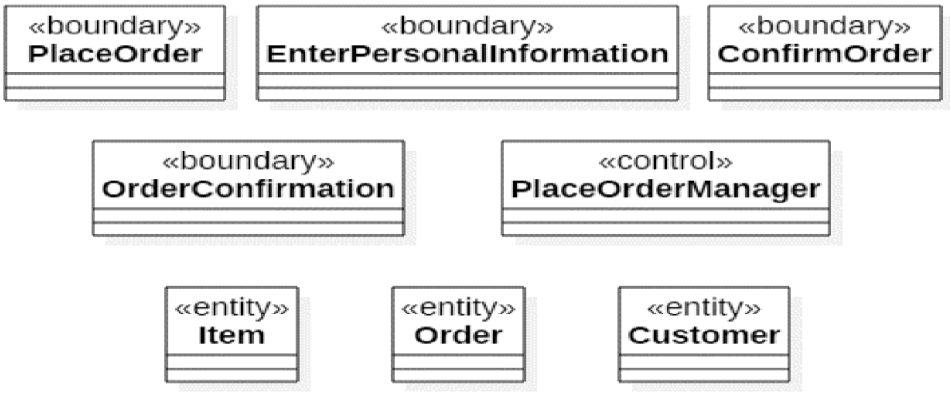


Рисунок 2.19 – Діаграма класів із стереотипами

Присвоїмо класам один із стереотипів UML (текстові або графічні). Після привласнення класам стереотипів їх зовнішній вигляд зміниться. Поряд з ім'ям класу з'явиться ім'я стереотипу, поміщене в кутові дужки (рисунок 2.19).

Атрибути і операції класів. Механізм інкапсуляції в UML реалізується за рахунок об'єднання властивостей і поведінки в одному об'єкті. Властивості об'єкту описуються за допомогою атрибутів класу, до якого відноситься об'єкт, а поведінка – операціями класу.

На прямокутнику класу атрибути описуються в другій секції під ім'ям, а операції – в третій, під атрибутами.

Атрибут класу служить для представлення окремої властивості або ознаки, яка є загальною для усіх об'єктів цього класу. Атрибути, таким чином, визначають структуру класу.

Атрибут – змістовна характеристика класу, що описує безліч значень, які можуть приймати окремі об'єкти цього класу. Усім атрибутам має бути присвоєне деяке значення.

Кожному атрибуту класу відповідає окремий рядок тексту, який може складатися з квантора видимості, імені, кратності, типу значень атрибуту і його початкового значення:

<квантор видимості> <ім'я> [кратність]: <тип>=<початкове значення> {властивість}

Квантор видимості може приймати одне з таких значень:

– символ «+» – загальнодоступний (Public), атрибут доступний або видний з будь-якого іншого класу пакету, в якому визначена діаграма;

– символ «#» – захищений (Protected), атрибут недоступний або невидний для усіх класів, за винятком підкласів цього класу;

– символ «–» – закритий (Private), атрибут недоступний або невидний для усіх класів без виключення;

– символ «~» – пакетний (Package), атрибут недоступний або невидний для усіх класів за межами пакету, в якому визначений клас-власник цього атрибуту.

Ім’я атрибуту є текстовим рядком, який використовується як ідентифікатор відповідного атрибуту і тому має бути унікальним в межах цього класу.

Кратність атрибуту характеризує загальну кількість конкретних атрибутів цього типу, що входять до складу окремого класу. Наприклад: 0..1 – нуль або один; 2..* – два або більше; 2..5 – 2,3,4 або 5; * – будь-яке позитивне число або нуль.

Тип атрибуту є виразом, семантика якого обумовлена деяким типом даних, визначеним у пакеті «Типи даних» мови UML або самим розробником. Тип атрибуту може визначається залежно від мови програмування, яку передбачається використати для реалізації цієї моделі. Якщо атрибутом класу виступає інший клас, то типом атрибуту буде цей клас. У простому випадку тип атрибуту вказується рядком тексту, що має осмислене значення в межах пакету або моделі, до яких відноситься даний клас.

Наприклад, атрибут item класу Замовлення (Order) має бути екземпляром класу Товар (Item). Для інших атрибутів задамо стандартні типи (рисунок 2.20).

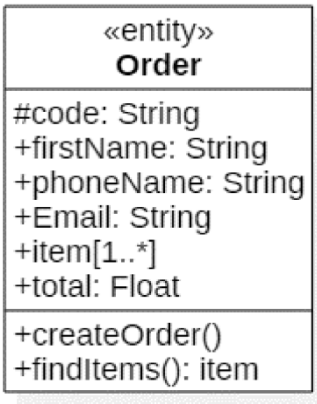


Рисунок 2.20 – Типи атрибутів класу Order

Початкове значення служить для завдання початкового значення відповідного атрибуту в момент створення окремого екземпляра класу.

Операцією або ***методом класу*** називається іменована послуга, яку можна запросити у будь-якого об’єкту цього класу. Сукупність операцій характеризує функціональний аспект поведінки класу.

Операції класу визначаються в розділі, розташованому нижче розділу з

атрибутами. Кожній операції класу відповідає окремий рядок тексту, який може складатися з квантора видимості, імені, списку параметрів операції, типу, поверненого операцією значення:

<квантор видимості> <ім'я> (список параметрів): <тип повернення значення> ={властивість}

Правила завдання квантора видимості і імені операції аналогічні їх визначенню для атрибутів.

Список параметрів є переліком розділених комою формальних параметрів, кожен з яких може бути представлений в такому виді:

<вид параметра> <ім'я параметра>: <вираз типу>=< значення за умовчанням>,

де *вид параметра – одне з ключових слів: in, out* або *inout*;

ім'я параметра – ідентифікатор відповідного формального параметра;

вираз типу є залежною від конкретної мови програмування специфікацією типу поверненого значення для відповідного формального параметра;

значення за умовчанням у загальному випадку є виразом для значення формального параметра, синтаксис якого залежить від конкретної мови програмування.

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

Властивість служить для вказівки значень властивостей, які можуть бути застосовані до цього елементу.

Діаграма класів з деякими операціями атрибутами і стереотипами виглядатиме так, як показано на рисунок 2.21. Наприклад, на діаграмі класу **Оформлення замовлення** перше повідомлення посилається об'єкту **PlaceOrder**. Цей об'єкт повинний уміти запускати оформлення замовлення, якщо кошик не по рожній, тому для нього створили відповідну операцію **placeOrder()**.

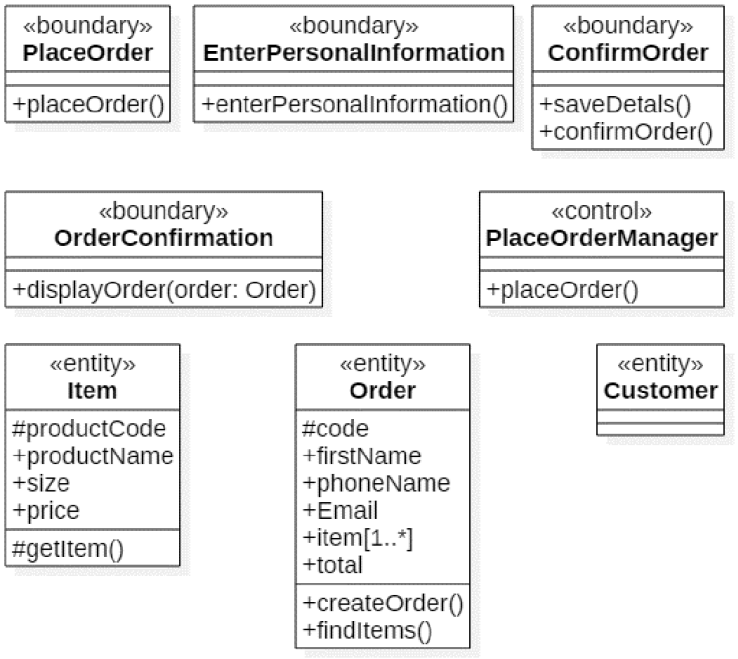


Рисунок 2.21 – Діаграма класів з операціями і атрибутами

Для класу EnterPersonalInformation створили відповідну операцію enterPersonalInformation() з повідомлення, що посилається об'єктом PlaceOrder, а не з такого повідомлення, що посилається Покупцем. Для покупця це повідомлення означає заповнення полів форми. У клас OrderConfirmation додали операцію displayOrder(), створивши її з повідомлення «display order».

Відношення між класами. Для того щоб виявити зв'язки класів, досліджуються сценарії і діаграми послідовності: якщо об'єкт посилає повідомлення іншого об'єкту, то між ними існує відношення. Відношення виникають також, якщо один клас використовує інший клас в якості параметра операції. У нотації UML визначені п'ять видів (типів) відношень між класами: асоціації, агрегації, композиції, залежності, і узагальнення (рисунок 2.22).

Перераховані типи відношень вже були визначені при побудові діаграм варіантів використання. Тому тут наведені тільки приклади їх графічного зображення на діаграмі класів.

Асоціація – довільне відношення або семантичний взаємозв’язок між класами, що мають загальною структуру і семантику. ***Відношення асоціації*** позначається суцільною лінією з додатковими спеціальними символами (ім’я

асоціації, імена і кратність класів-ролей асоціації), які характеризують окремі властивості конкретної асоціації (рисунок 2.23).

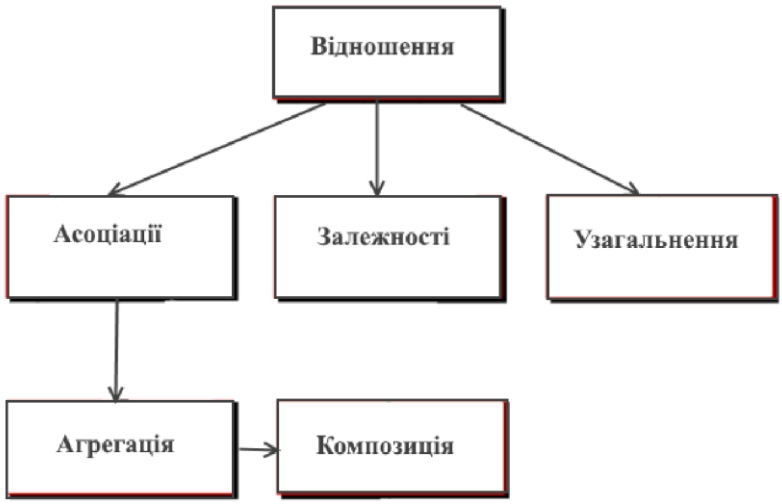


Рисунок 2.22 – Відношення між класами

Асоціація може бути однонаправленою або двонаправленою. У першому випадку її зображують у вигляді стрілки, що показує напрям зв'язку (рис. 2.23, *в)*. У другому випадку – у вигляді подвійної стрілки або просто лінія без стрілок (рисунок 2.23, *а*).

Якщо між класами створений двонаправлений зв’язок, то кожен з них бачить відкриті атрибути і операції інших класів.

Якщо між класами встановлена однонаправлена асоціація, то в цьому випадку клас, від якого спрямована стрілка, знає відкриті атрибути і операції іншого класу, а другий клас, до якого йде стрілка, не бачить атрибути і операції першого класу. Наприклад, клас Покупець знає назву і розмірний ряд Товару, але клас Товар не знає ім’я і адресі Покупця. При такому відношенні між класами Покупець повинен знати про клас Товар і тому не може використовуватися без нього, але клас Товар не залежить від Покупця і може бути повторно використаний самостійно, незалежно від класу Покупець.

Асоціація може бути рефлексивною, це означає, що один екземпляр класу звертається до іншого екземпляра цього класу. Зображується відношення рефлексії як на рисунку. 2.23, *г*.

Асоціації можна присвоїти ім’я. Зазвичай в якості імені використовується дієслово або речення, що пояснює причину виникнення зв’язку. За допомогою чорного трикутника, розташованого над лінією зв’язку справа або зліва від імені асоціації, уточнюється сенс імені й вказується напрям імені зв'язку(рисунок 2.23, *б)*.

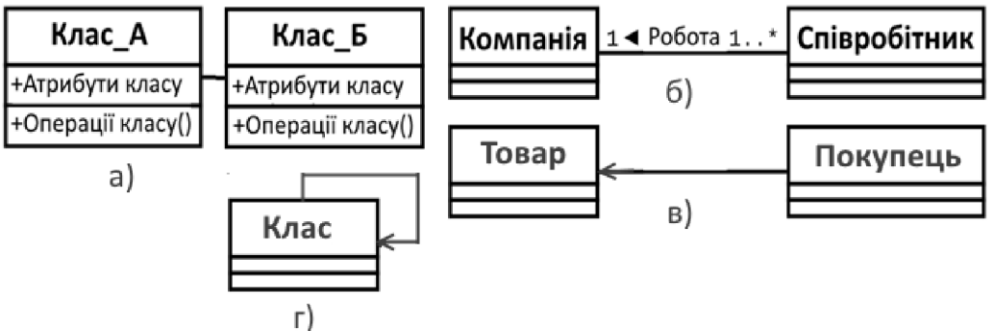


Рисунок 2.23 – Графічне зображення відношення асоціації на діаграмі класів

Закінчення лінії асоціації в місці, де вона з’єднується з класом, називається роллю асоціації. Назва ролі асоціації – це іменник, який уточнює, описує роль, в якій один клас бере участь у зв’язку з іншим класом. Роль може бути вказана для обох класів, що беруть участь у зв'язку, або тільки для одного (рисунок 2.24).

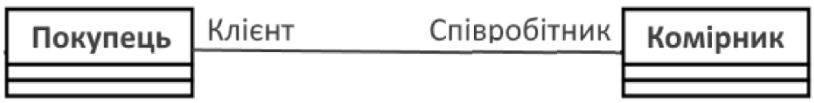


Рисунок 2.24 – Асоціація с ролями

Кратність (потужність) визначається для класів і вказує допустиму кількість об'єктів (екземплярів класу), що беруть участь у відношенні асоціації.

Кратність вказує, скільки екземплярів одного класу взаємодіють за допомогою відношення з одним екземпляром цього класу в даний момент.

Приклади індикаторів потужності:

- 0..1 нуль або один;
- 1 рівно один;
- 1..* один або багато;

– 2..5 2,3,4 чи 5.

Наприклад, покупець може оформити багато замовлень або не оформити жодного. Замовлення має бути зроблене тільки 1 покупцем (рисунок 2.25).

Рисунок 2.25 – Кратність класів в асоціації

Як видно з прикладу, читати кратність класу треба на протилежному кінці зв'язку. Вказувати ім'я, роль або кратність зв'язку необов'язково. Це треба робити, коли це може допомогти точнішому представленню моделі системи і кращому її розумінню.

Якщо в діаграмі класів асоціація між двома класами має спеціальний вигляд «частина-ціле», то використовується асоціація спеціального виду – агрегатна асоціація. У цьому випадку клас «ціле» має більш високий концептуальний рівень, ніж клас «частина». Графічно агрегатні асоціації зображуються у вигляді простої асоціації з незафарбованим ромбом на стороні класу-«цілого» (рисунок 2.26).

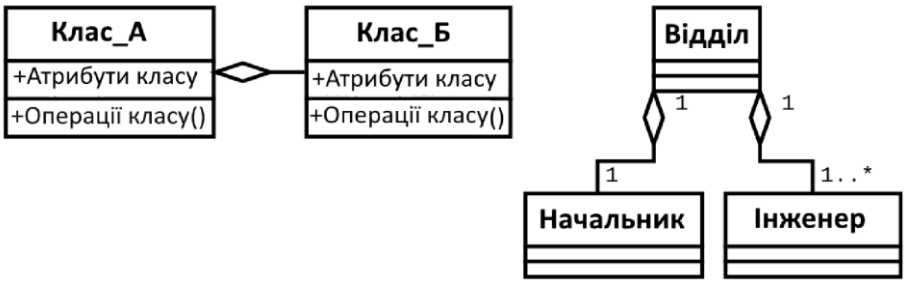


Рисунок 2.26 – Графічне зображення відношення агрегації на діаграмі класів

У нашому прикладі з Internet-магазином будь-яке замовлення складається з товарів (агрегатна асоціація, рисунок 2.27).

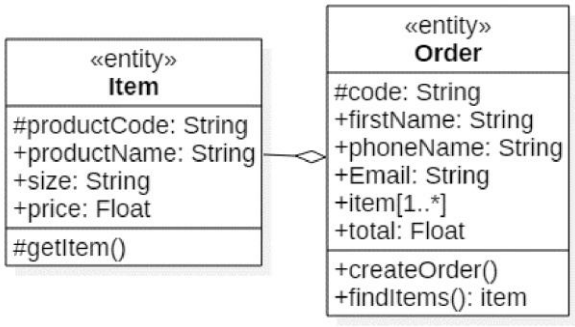


Рисунок 2.27 – Агрегація між класами Товар і Замовлення

Якщо зв'язок «частини» і «цілого» настільки сильний, що знищення «цілого» приводити до знищення усіх його «частин», то такі агрегатні асоціації називаються композитними або просто композиціями. За наявності композиції об'єкт - частина може бути частиною тільки одного об'єкту - цілого(композиту), причому година життя частин і цілого співпадають. Як тільки буде знищений об'єкт Ціле, так разом з ним буде знищений об'єкт Частина. При звичайній агрегатній асоціації «частина» може одночасно належати декільком «цілим». Графічно композиція зображується у вигляді простої асоціації, доповненої зафарбованим ромбом з боку «цілого» (рисунок 2.28).

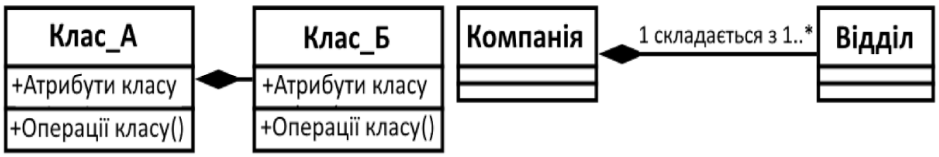


Рисунок 2.28 – Графічне зображення відношення композиції на діаграмі класів

Для відношень композиції і агрегації можуть використовуватися додаткові позначення, які застосовуються для відношення асоціації. А саме можуть вказуватися кратності окремих класів, які в загальному випадку не є обов'язковими.

Проте кратність агрегованого кінця композитної агрегації не повинна мати верхньої межі більше ніж 1. Розглянемо приклад - вікно графічного інтерфейсу програми, яку складається із заголовка, півосі прокрутки, головного меню і

робочої області (рисунок 2.29). Як тільки ми закриємо вікно програми, усі інші елементи вікна перестануть існувати. Подібне вікно є класом, а його елементи також є класами.

Оскільки агрегація і композиція є відношеннями асоціації, то для них допустимо, але не обов'язково, вказувати ім’я, роль і кратність.

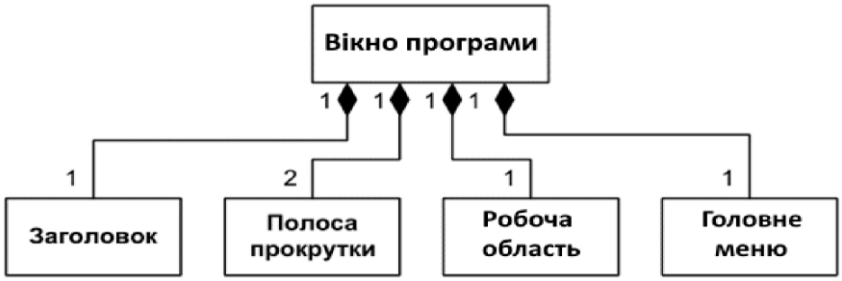


Рисунок 2.29 – Діаграма класів для ілюстрації відношення композиції

Залежністю називається відношення використання, яке визначає, що зміна в специфікації однієї сутності може вплинути на іншу сутність, яка її використовує, причому зворотне в загальному випадку не вірно.

Відношення залежності графічно зображується пунктирною лінією між відповідними елементами зі стрілкою, спрямованою від покладу класу до непокладу класу (рисунок 2.30). Залежності застосовуються, щоб показати, що один клас використовує інший. Тобто, один клас є клієнтом іншого класу-постачальника і використовує цей клас-постачальник як параметр своєї операції.

Рисунок 2.30 – Графічне зображення відношення залежності

Наприклад, якщо використати клас Order (Замовлення) як вхідний параметр операції displayOrder (відобразитиЗамовлення) класу OrderConfirmation (ПідтвердженняЗамовлення), то для виконання цієї операції клас

OrderConfirmation (ПідтвердженняЗамовлення) використовує клас Order (Замовлення), тобто вони пов' язані відношенням залежності (рисунок 2.31).

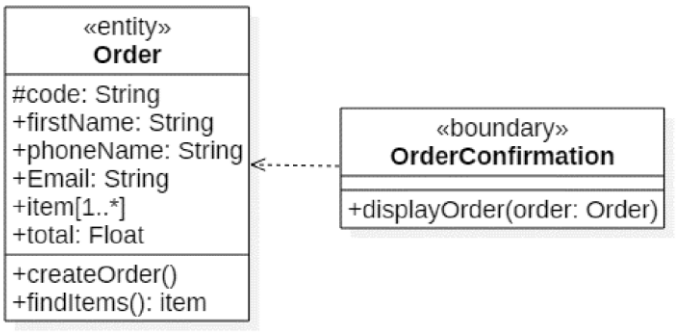


Рисунок 2.31 – Залежність між класами Order і OrderConfirmation

Створення екземпляра класу ПідтвердженняЗамовлення не спричинить за собою створення екземпляра класу Замовлення. Проте ці два класи зможуть обмінюватися повідомленнями на діаграмах взаємодії.

Узагальнення – це відношення успадкування між двома елементами моделі. Воно дає класу можливість успадковувати відкриті або захищені атрибути і операції суперкласу (класу від якого успадковуються атрибути і операції). Крім успадкованих кожен клас може мати свої атрибути і операції.

На діаграмах узагальнення зображується у вигляді стрілки з не зафарбованим трикутником у суперкласу, що йде від нащадка.

Рисунок 2.32 – Відношення узагальнення між класами

Наприклад. у магазині можуть працювати різні співробітники: співробітник відділу продажів, комірник, директор. Усі вони мають загальні властивості: ім’я, адреса, телефон, дата народження, посада, тому можна розглядати

узагальнювальну сутність Співробітник, атрибути і операції якої сутності Директор і Комірник будуть наслідувати (рисунок 2.32).

Якщо Співробітник має в якості атрибутів ім’я, адресу, телефон, дату народження, посаду, то сутності Директор і Комірник, звичайно, наслідують ці атрибути зі своїми значеннями. Крім того вони можуть мати і власні атрибути або операції. Наприклад, у Директора може бути операція звільнити Співробітника, якої не може бути у Комірника, а у останнього операція – видати Товар.

Закриті атрибути і операції не можуть унаслідуватися нащадками.

Приклади діаграм класів. На рисунку 2.33 представлена діаграма класів сценарію Оформлення замовлення в Internet-магазині представлена. Клас PlaceOrder пов'язаний з EnterPersonalInformation, а об'єкт ConfirmOrder посилає повідомлення об'єкту класу PlaceOrderManager. PlaceOrderManager пов'язаний з об'єктами класів Order і OrderConfirmation.

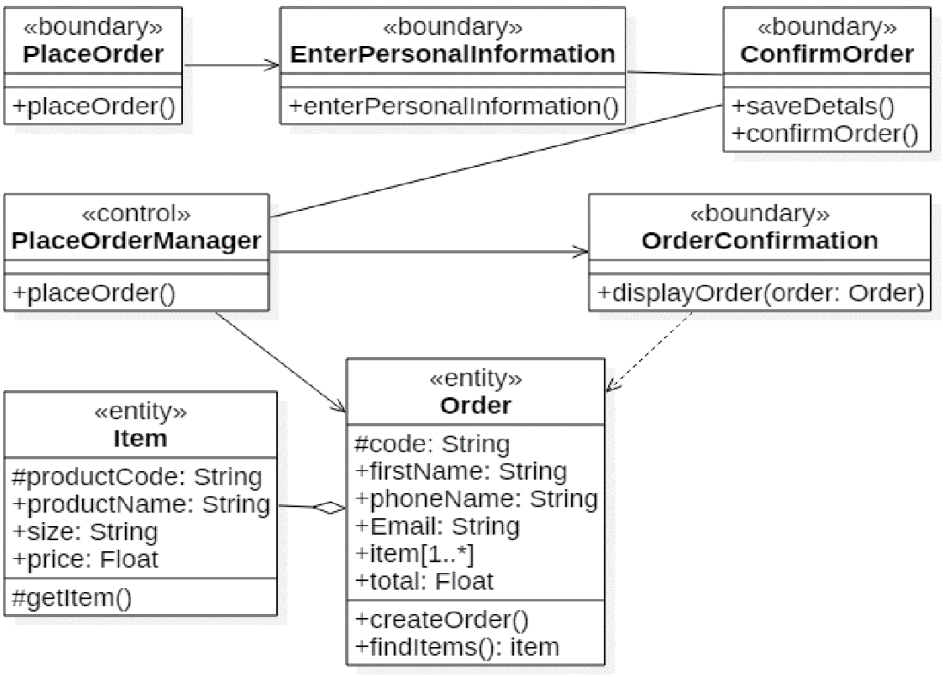


Рисунок 2.33 – Діаграма класів сценарію Оформлення замовлення

Усі перераховані зв'язки визначимо відношенням асоціації. Клас OrderConfirmation використовує клас Order як параметр своєї операції: між ними

визначимо відношення залежності. Екземпляри класу Order складаються з екземплярів класу Item. Між ними створено відношення агрегації.

Для того щоб клас ConfirmOrder міг виконувати операцію підтвердження замовлення, він має бути пов'язаний з класом EnterPersonalInformation, тому між ними створено відношення асоціації.

На рисунку 2.34 показана сукупність класів, узятих з ІС закладу вищої освіти (ЗВО). У нижній частині діаграми розташовані класи Студент, Курс і Викладач. Між класами Студент і Курс є асоціація показує, що студенти можуть відвідувати курси. Більше того, кожен студент може відвідувати будь-яку кількість курсів і на кожен курс може записатися будь-яке число студентів.

Два класи (ЗВО і Факультет) містять операції, що дозволяють маніпулювати їх частинами. Ці операції включені із-за їх важливості для підтримки цілісності даних (наприклад, додавання або видалення класу Факультет зачіпає цілий ряд інших об’єктів).

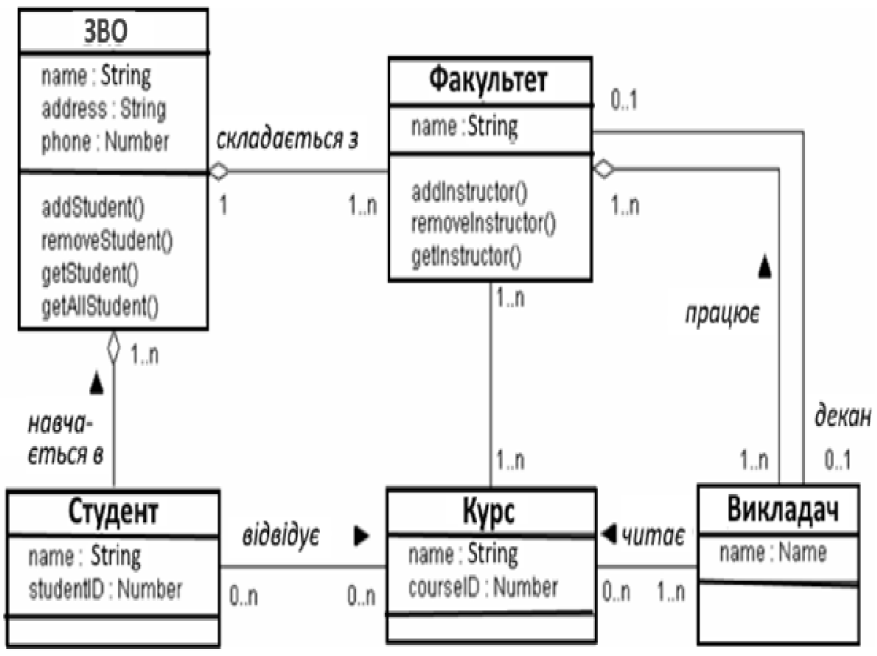


Рисунок 2.34 – Діаграма класів для інформаційної системи ЗВО

Оскільки діаграма класів є однією з головних діаграм моделі проектованої системи, то побудуємо ще одну UML-діаграму класів. В якості прикладної області

візьмемо відділ кадрів деякого підприємства (наприклад, internet-магазину) і почнемо будувати його модель. Для прикладів атрибутів і методів класів використовуватимемо мову Java.

Узагальнення. Розглянемо два класи: абстрактний клас «Man» (людина) і більш спеціалізований клас «Employee» (співробітник). Оскільки клас «Employee» наслідує властивості і методи класу «Man», то між ними встановимо зв’язок у вигляді відношення узагальнення (рисунок 2.35). Відношення узагальнення – це спадкоємність. Це відношення у будь-якій ООП мові програмування має явну реалізацію через розширення (extends) одного класу іншим.

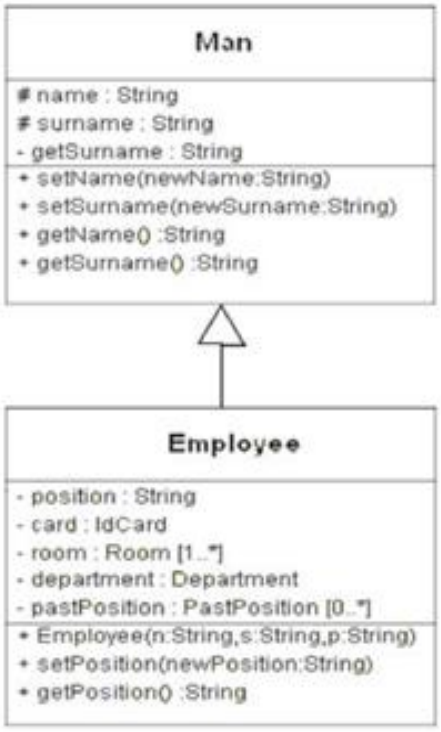


Рисунок 2.35 – Відношення узагальнення

Асоціація. Асоціація показує відношення між об'єктами-екземплярами класу. У модель додамо клас «IdCard», що представляє ідентифікаційну картку (пропуск) співробітника. Кожному співробітнику може відповідати тільки одна ідентифікаційна картка, потужність зв'язку 1 до 1 (бінарна асоціація).

Клас Employee має поле card, в якого тип IdCard, клас також має методи для надання значення (setIdCard) цьому полю і для отримання значення (getIdCard). З

екземпляра об'єкту Employee можна дізнатися про пов’язаний з ним об’єкт типу IdCard, а це означає, що навігація (стрілка на лінії) спрямована від класу Employee до класу IdCard (рисунок 2.36).

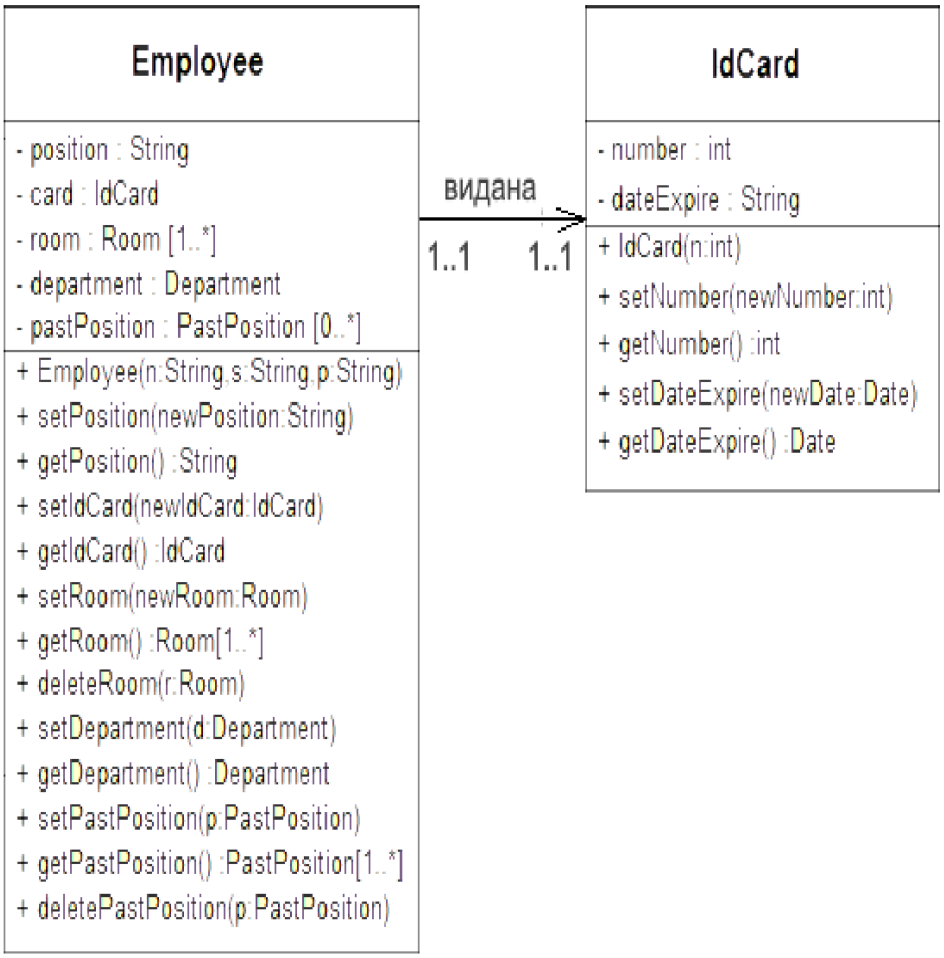


Рисунок 2.36 – Бінарна асоціація

Якщо в організації закріплюють за працівниками приміщення, то додаємо новий клас Room. Кожному об’єкту працівник (Employee) може відповідати декілька робочих приміщень. Потужність зв’язку один-до-багатьох. Додамо в клас Employee поле і методи для роботи з класом Room. Навігація від Employee до Room (рисунок 2.37).

Агрегація. Якщо підприємство структуроване по відділах, то введемо в модель клас Department (відділ). У кожному відділі може працювати один або більше людей. Відділ включає одного або більше співробітників і, таким чином, їх

агрегує. На підприємстві можуть бути співробітники, які не належать жодному відділу, наприклад, директор підприємства (рисунок 2.38).

Клас Department, окрім конструктора і методу зміни імені відділу, має методи для занесення у відділ нового співробітника, для видалення співробітника і для отримання всіх співробітників, що входять у цей відділ.

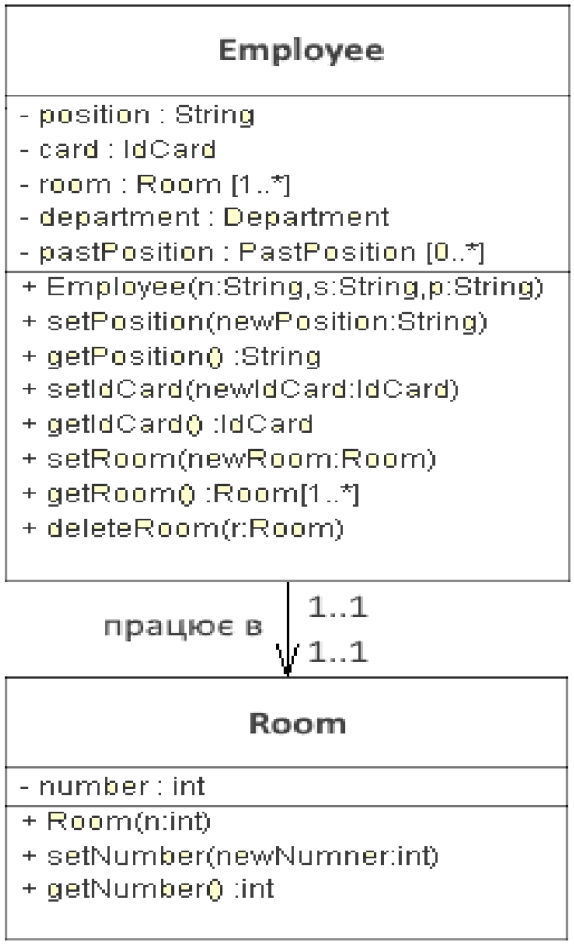


Рисунок 2.37 – N-арна асоціація

Навігація на діаграмі не показана, це означає, що вона є двонаправленою: від об’єкту типу «Department» можна дізнатися про співробітника і від об’єкту типу «Employee» можна дізнатися до якого відділу він відноситься.

Для того щоб дізнатися до якого відділу відноситься який-небудь співробітник, то додамо у клас Employee поле і методи для призначення і отримання відділу.

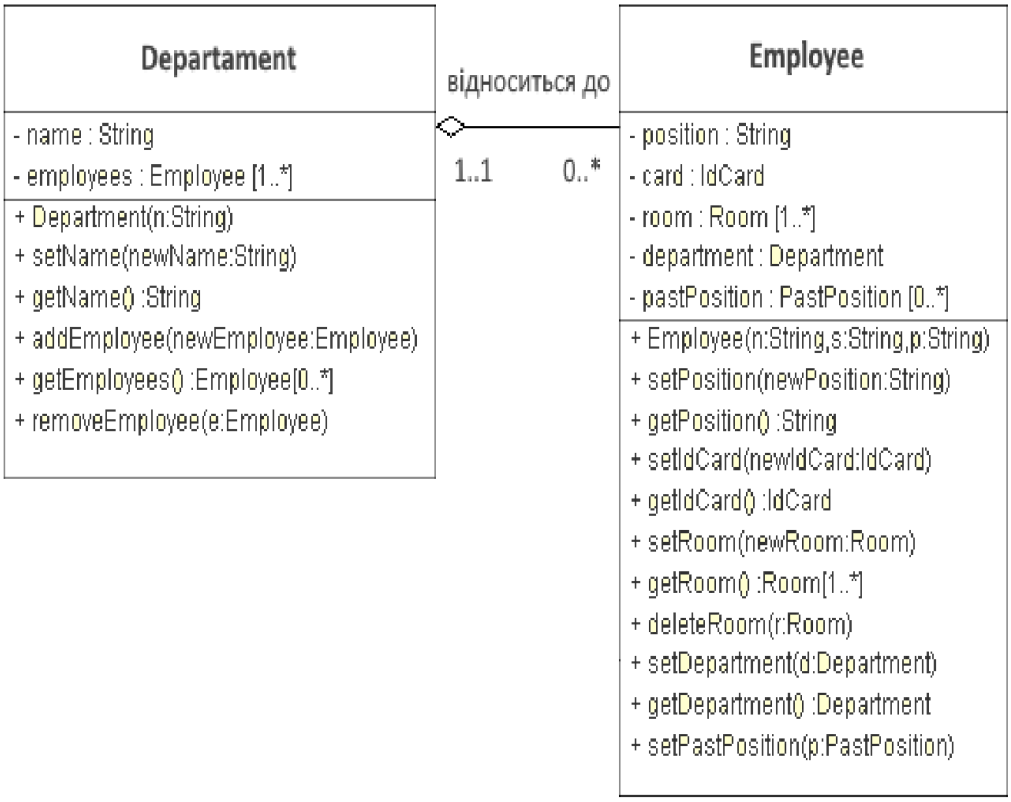


Рисунок 2.38 – Відношення агрегації

Композиція. Припустимо, що однією з вимог до нашої системи є вимога про те, щоб зберігати дані про колишню займану посаду на підприємстві. Введемо новий клас «pastPosition». У нього, окрім властивості «ім’я» (name), введемо і властивість «department», яка зв’яже його з класом «Department».

Дані про минулі займані посади є частиною даних про співробітника, таким чином, між ними зв’язок ціле-частина. В той же час, дані про минулі посади не можуть існувати без об'єкту типу «Employee».

Знищення об'єкту «Employee» повинне привести до знищення об'єктів «pastPosition» (відношення композиції, рисунок 2.39).

У клас Employee додамо властивості і методи для роботи з даними про минулу посаду (setPastPosition(), getPastPosition(), deletePastPosition()).

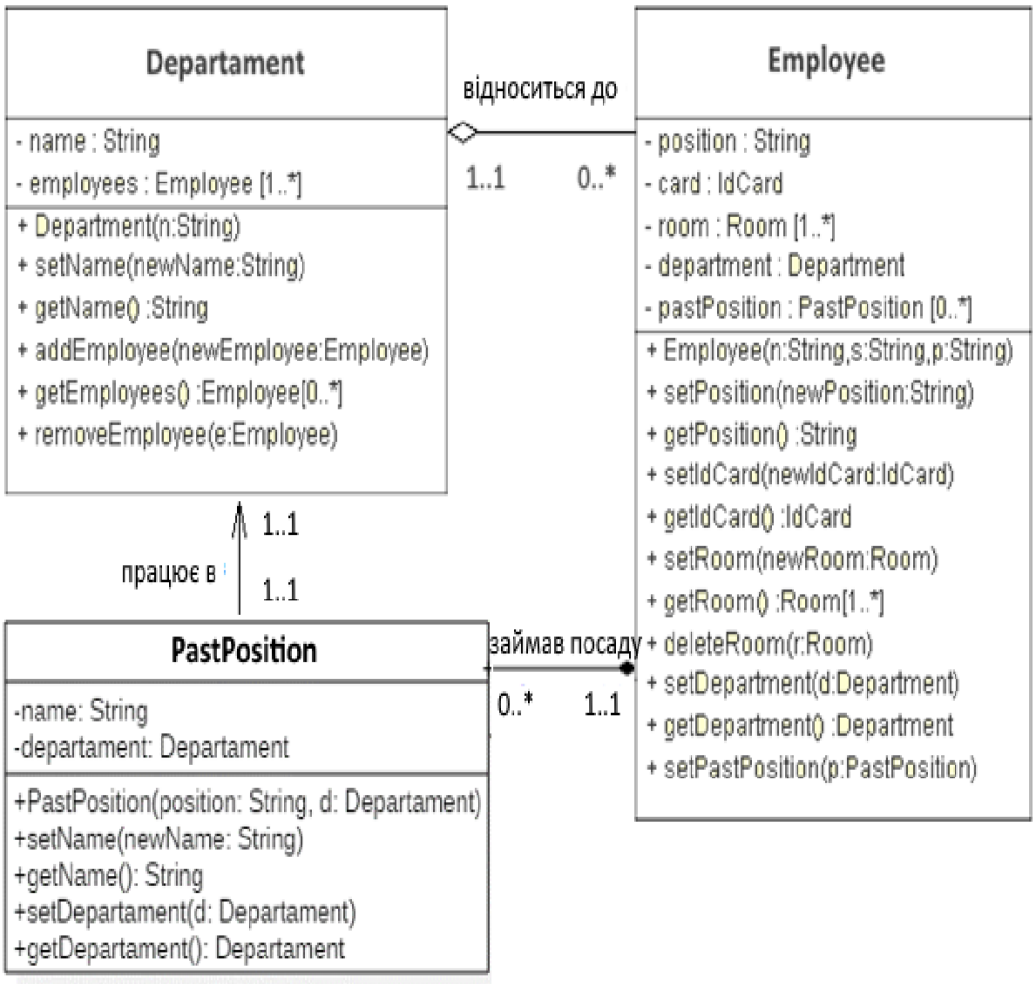


Рисунок 2.39 – Відношення композиції

Залежність. Для організації діалогу з користувачем введемо в систему клас «Menu» з методом «showEmployees», який показує список співробітників та їх посади. Параметром для методу є масив об'єктів «Employee». Таким чином, зміни внесені у клас «Employee» можуть спричинити і зміни класу «Menu» (відношення залежності, рисунок 2.40).

На даній діаграмі клас «Menu» не відноситься до прикладної області, а є «системним» класом уявного застосування.

Таким чином, в результаті моделювання отримаємо таку загальну діаграму класів відділу кадрів (рисунок 2.41).

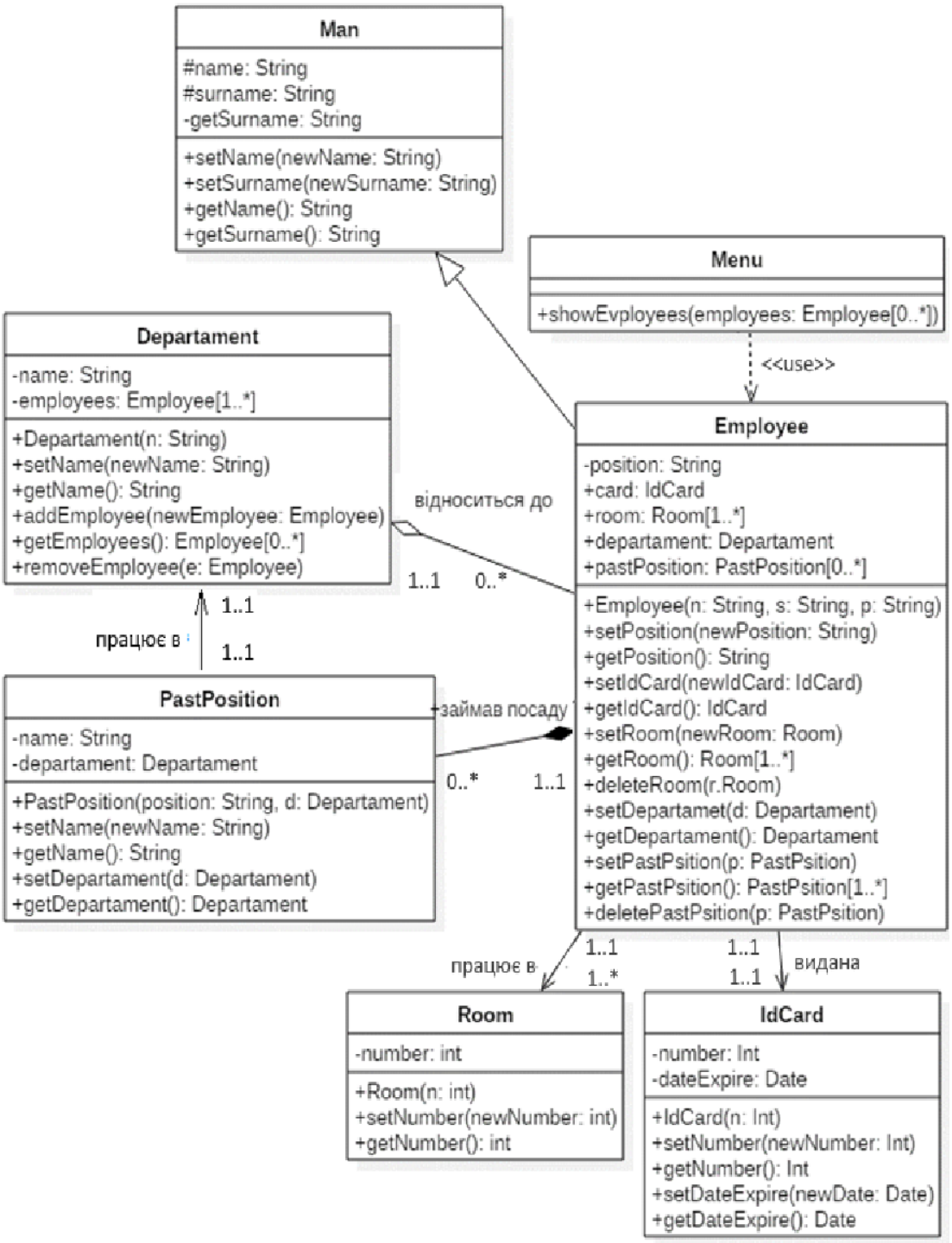


Рисунок 2.40 – Відношення залежності

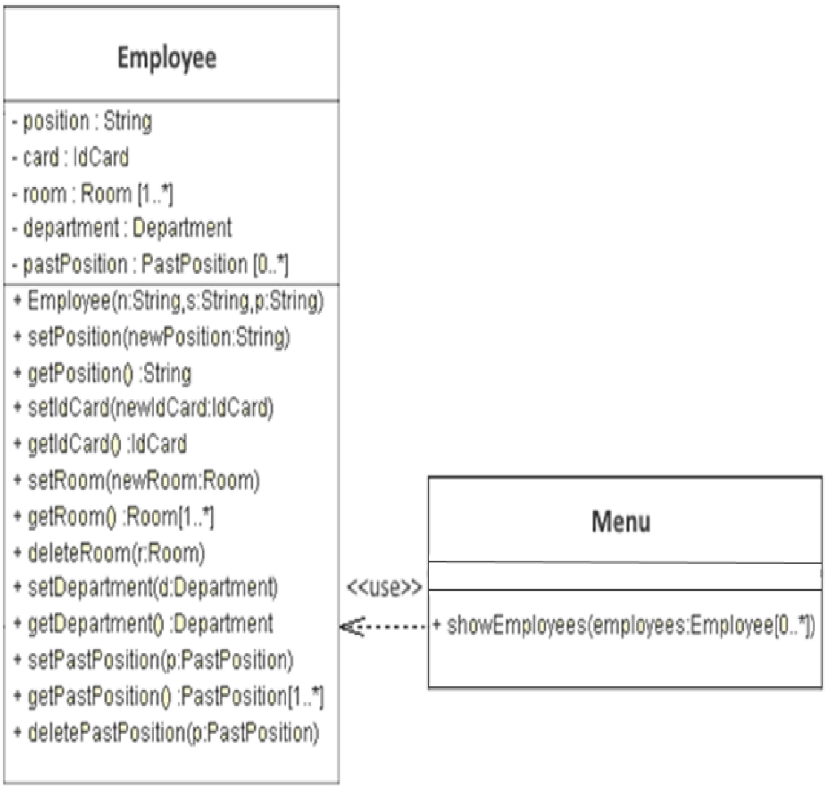


Рисунок 2.41 – Діаграма класів відділу кадрів

2.3.7.3.2 Діаграми пакетів

Якщо в моделі небагато класів, то ними легко управляти, проте більшість систем містять велику кількість класів, тому потрібний механізм, що дозволяє групувати класи і полегшати їх повторне використання. Таким механізмом в UML є пакети.

Пакети – це спосіб угрупування будь-яких елементів мови UML: діаграм, варіантів використання, класів тощо. Пакети застосовують, щоб розбивати великі моделі на частини, групуючи класи, що мають деяку спільність. Це значною мірою спрощує розуміння і підтримку проектованої моделі.

Пакети є життєво необхідним засобом для великих проектів. Їх слід використати в тих випадках, коли діаграма класів, що охоплює усю систему в цілому і розміщена на єдиному аркуші паперу формату А4, стає нечитабельною.

Існує декілька найпоширеніших підходів до угрупування.

Угрупування по стереотипу. Стереотипи – це механізм, що дозволяє розділяти класи на категорії. В UML визначені три основні стереотипи: межа, об’єкт та управління.

Пограничні класи розташовані на межі проектованої системи. Вони включають усі форми взаємодії з довкіллям: звіти, форми документів, інтерфейси з апаратурою, користувачами й іншими системами.

Класи-сутності містять інформацію про класи внутрішнього устрою системи, тобто є суттю цієї системи.

Управляючі класи відповідають за координацію дій інших класів, їх взаємодію між собою.

Угрупування по функціональності. Окрім згаданих вище стереотипів можна створювати і свої власні (теми пакетів), тобто тема пакету може бути досить довільною. Нею може бути основний клас, основні відношення, найважливіші аспекти функціональності. Наприклад, модель класів компілятора можна було б розділити на пакети лексичного аналізу, розбору, семантичного аналізу, генерації коду та оптимізації.

Для графічного зображення пакетів на діаграмах застосовується спеціальний графічний символ – великий прямокутник з невеликим прямокутником, приєднаним до лівої верхньої частини першого. Візуально символ пакету нагадує піктограму теки на екрані монітора (рисунок 2.42).

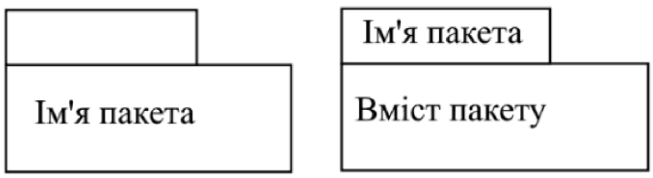


Рисунок 2.42 – Графічне зображення пакетів

Пакет може бути зображений без своїх членів. У цьому випадку ім’я пакету можна вказати у великому прямокутнику, воно має бути унікальним у межах даної моделі. Якщо зображуються класи-члени пакету, то тип пакету пишеться на

закладці, а усередині великого прямокутника перераховуються класи-члени пакету за допомогою символу класу.

У UML для встановлення взаємодій між пакетами в основному використовується відношення залежності.

Залежність – це однонапрямлене відношення, що показує, як клас залежить від визначень, зроблених в іншому класі. Залежність означає відношення типу «постачальник-клієнт», коли модифікація постачальника може вплинути на одного або декілька клієнтів. Клієнт – елемент моделі, залежний в деякому контексті від елементу або елементів постачальника.

Залежність між двома пакетами існує у тому випадку, якщо між двома будь-якими класами з цих пакетів існує будь-який взаємозв'язок.

Відношення залежності зображують у вигляді пунктирної лінії зі стрілкою, спрямованою від залежного класу до незалежного класу (рисунок 2.43).

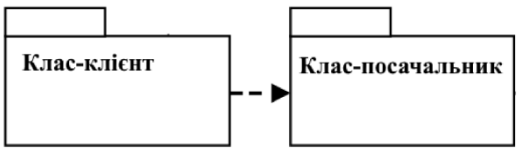


Рисунок 2.43 – Графічне зображення відношення залежності

Іншим типом відношень між пакетами (без використання ліній залежності) є відношення вкладеності або включення пакетів один в одного. Це відношення може бути зображене простим розміщенням одного пакету усередині іншого пакету (рисунок 2.44, а). Так, у даному випадку Пакет А містить в собі два підпакети: Пакет В і Пакет С. З іншого боку, це ж відношення може бути зображене за допомогою відрізків ліній аналогічно графічному представленню дерева (рисунок 2.44, б). Пакет-контейнер з'єднується з підпакетами

суцільною лінією, на кінці якої зображується спеціальний символ – ⊕.

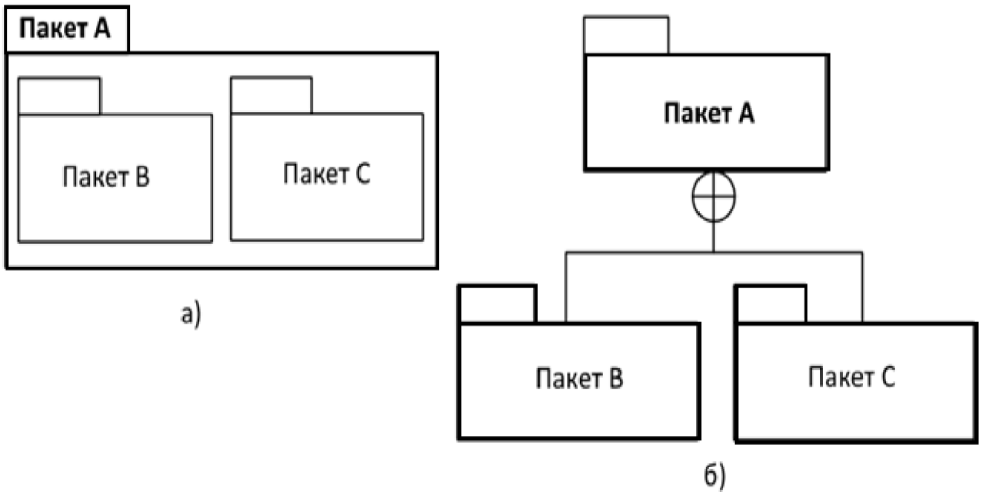


Рисунок 2.44 – Приклади графічного зображення пакетів

Таким чином, діаграма пакетів – це структурна діаграма, призначена для представлення розміщення елементів моделі в пакетах і специфікації залежностей між пакетами і їх елементами (рисунок 2.45).

У контексті архітектурного проектування пакет є зручним засобом упаковки підсистеми, яка є частиною архітектурної організації, а діаграма пакетів – моделлю структури архітектурного рівня.

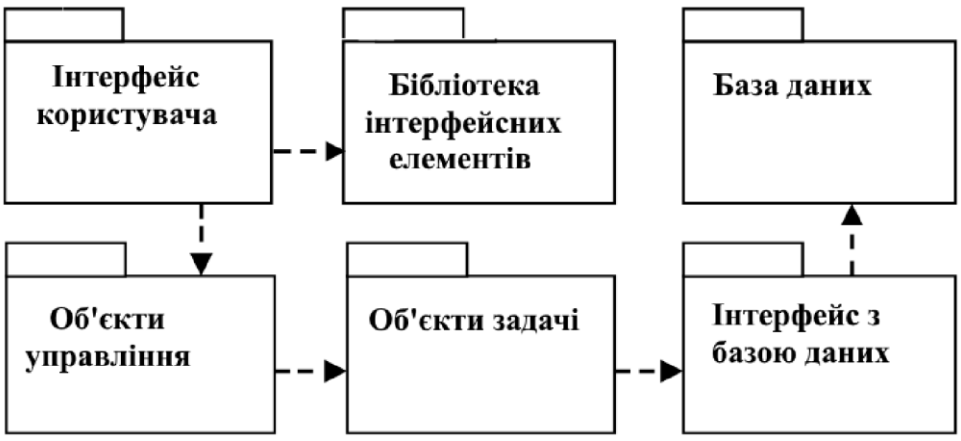


Рисунок 2.45 – Проста діаграма пакетів

2.3.7.4 Архітектурне проектування

Основне призначення логічного представлення полягає в аналізі структурних і функціональних відношень між елементами моделі системи. Проте для створення конкретної фізичної системи необхідно деяким чином реалізувати усі елементи логічного представлення в конкретні матеріальні сутності. Для опису таких реальних сутностей призначений інший аспект модельного представлення, а саме – фізичне представлення моделі.

Створення архітектури – це проектування на найвищому рівні. Логічна архітектура описує систему в термінах її принципової організації у вигляді пакетів, програмних класів і підсистем. Вона називається логічною, оскільки не визначає способи розгортання цих елементів у різних операційних системах або на фізичних комп'ютерах в мережі (це відноситься до архітектури розгортання).

Етап архітектурного проектування ІС повинен бути відображений шляхом опису процесу об'єктно-орієнтованої декомпозиції системи до рівня переліку підсистем та їх зв'язків, опису її логічної структури у вигляді пакетів компонентів (**діаграма компонентів**) [6].

2.3.7.4.1 Діаграма компонентів

Ще одним будівельним блоком для створення архітектури об'єктно-орієнтованої системи на фізичному рівні вважається компонент програмного забезпечення. Діаграма компонентів показує визначення, внутрішню структуру і залежності набору компонентів. В якості фізичних компонент можуть виступати файли, бази даних, бібліотеки, модулі, пакети, інтерфейси ІС тощо.

Компонент – елемент моделі, що представляє деяку модульну частину системи з інкапсульованим вмістом. У мові UML в якості компонента може розглядатися будь-який автономний елемент системи або підсистеми.

Внутрішні частини компонентів приховані і недоступні ззовні, окрім тих, які надаються за допомогою його інтерфейсів. Компоненти можуть бути пов'язані між собою за допомогою певних відношень. Ці відношення визначені так, щоб розглядати компонент як можна незалежнішими від свого оточення.

Для графічного представлення компонента в UML 1 використовується спеціальний символ – прямокутник зі вставленими ліворуч двома

прямокутниками меншого розміру (рисунок 2.46, а, б). Усередині великого прямокутника записується ім’я компонента і, можливо, деяка додаткова інформація.

У мові UML 2 компонент зображується у формі прямокутника з необов’язковим ключовим словом «component», записаним у формі стереотипу. Додатково у правому верхньому кутку символу класифікатора може бути зображена піктограма компонента у формі невеликого прямокутника з двома меншими прямокутниками (рисунок 2.46, в). Усередині символу прямокутника вказується ім’я компонента. Зображення компонента на діаграмі може містити також символи інтерфейсів, які приєднуються до прямокутника компонента за допомогою з’єднувачів (рисунок 2.46, *г*).

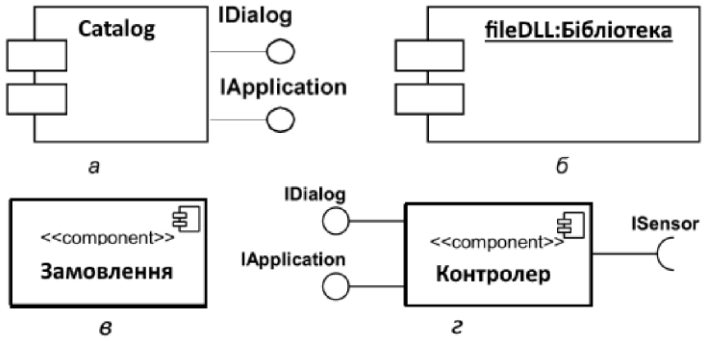


Рисунок 2.46 – Приклади графічного зображення компонент

Компонент може мати два різновиди інтерфейсів: інтерфейси, що надаються, і необхідні інтерфейси, які утворюють основу для взаємозв’язків компонентів між собою з використанням залежностей і з’єднувачів.

Інтерфейс, що надається, – це інтерфейс, який компонент пропонує для свого оточення.

Необхідний інтерфейс – це інтерфейс, який потрібний компоненту від свого оточення для виконання заявленої функціональності або поведінки.

Інтерфейси, що надаються, і необхідні інтерфейси, у мові UML 2 зображуються з використанням спеціальної нотації, яка дістала назву «Кулі і шарніра» або «льодяника на паличці».

Разом з інтерфейсами, які відносяться до зовнішнього представлення, на діаграмі може бути зображене внутрішнє представлення або представлення компонента у формі білого ящика. У цьому випадку усередині символу компонента зображуються класи або інші класифікатори, які реалізують поведінку цього компонента. Так, наприклад, компонент Замовлення має два класи, які реалізують його функціональність, а саме: ЗаголовокЗамовлення і РядокТовару (рисунок 2.47) [8]. Причому ці класи пов'язані відношенням композиції.

Діаграма компонентів, на відміну від раніше розглянутих діаграм, описує особливості фізичного представлення системи та дозволяє визначити архітектуру розроблюваної системи, встановивши залежності між програмними компонентами. У багатьох середовищах розробки модуль або компонент відповідає файлу.

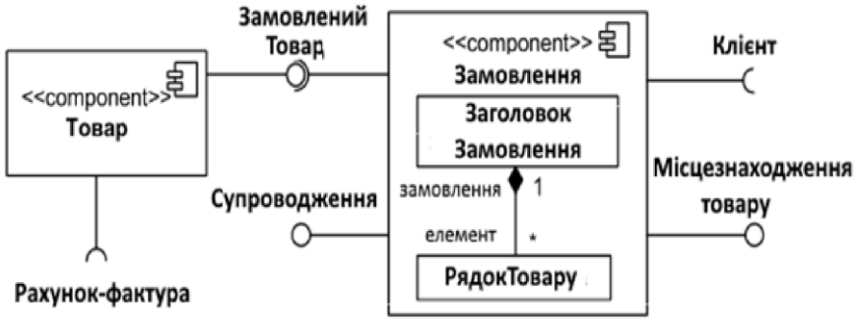


Рисунок 2.47 – Зображення компонент с інтерфейсами в нотації «кулі і шарніра»

Пунктирні стрілки, що з'єднують модулі, показують відношення взаємозалежності аналогічні тим, які мають місце при компіляції вихідних текстів програм. Основними графічними елементами діаграми компонентів є компоненти, інтерфейси і залежності між ними.

Інтерфейси також можуть бути зображені у формі звичайних прямокутників класу із стереотипом <<interface>> і секцією операцій. У цьому випадку необхідний інтерфейс позначається за допомогою відношення залежності, спрямований від одного з класів, що входять до його складу. Додатково це відношення може мати стереотип <<use>> (рисунок 2.48, ліворуч). Інтерфейс, що

надається, позначається за допомогою відношення реалізації, спрямованого від компонента або від одного з класів, що входять до його складу, до цього інтерфейсу (рисунок 2.48, справа). Неважко помітити, що ця нотація практично ідентична нотації, яка використовується на діаграмі класів [8].

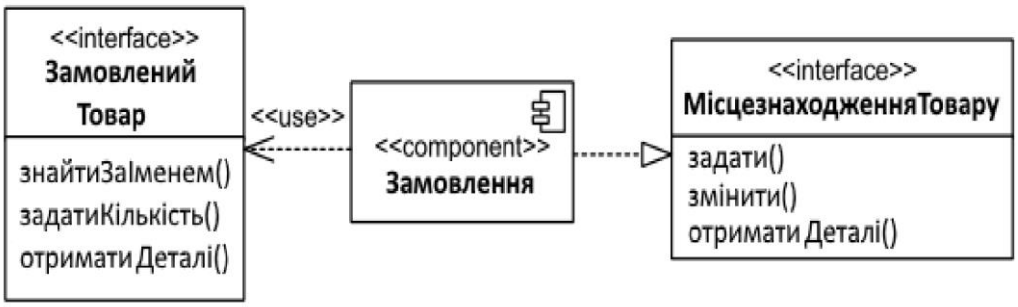


Рисунок 2.48 – Представлення інтерфейсів у формі символу класифікатора з відношеннями залежності і реалізації

На рисунку 2.49 показаний приклад простої діаграми компонентів. У даному прикладі компонент Till (Каса) може взаємодіяти з компонентом Sales Server (Сервер продаж) за допомогою інтерфейсу sales message (повідомлення про продажі).

Оскільки мережа може бути ненадійною, то компонент Message Queue (Черга повідомлень) встановлений так, щоб каса могла спілкуватися з сервером, коли мережа працює, і розмовляти з чергою повідомлень, коли мережа відключена.

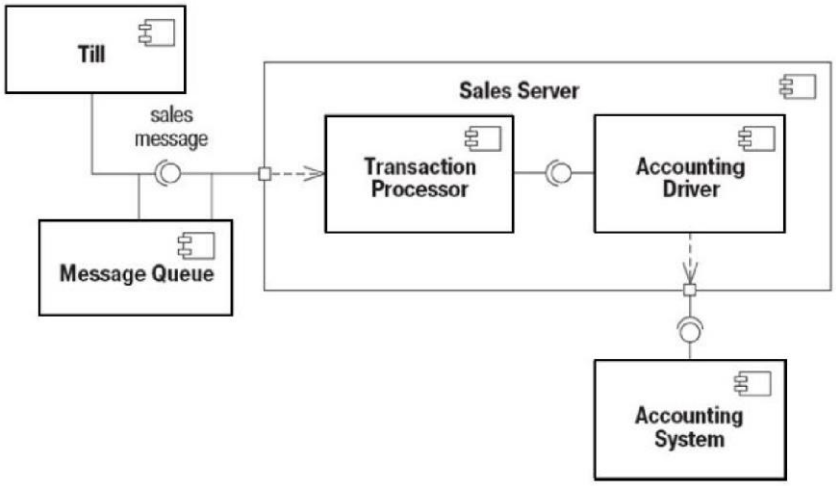


Рисунок 2.49 – Приклад простої діаграми компонентів

Тоді черга повідомлень зможе поговорити з сервером, коли мережа знову стане доступною. У результаті черга повідомлень надає інтерфейс для розмови з касою, і вимагає такий же інтерфейс для розмови з сервером. Сервер розділений на два основні компоненти: Transaction Processor (Процесор транзакцій) реалізує інтерфейс повідомлень, а Accounting Driver (Драйвер рахунків) спілкується з Accounting System (Система ведення рахунків).

Компонент у своєму складі має інтерфейсний клас, через який здійснюється доступ до інших класів об’єктів.

2.3.7.4.2 Розгортання програмної системи на апаратних засобах. Діаграма розгортання

Фізичне представлення програмної системи не може бути повним, якщо відсутня інформація про те, на якій платформі та на яких обчислювальних засобах вона реалізована. Якщо розробляється проста програма, яка може виконуватися локально на комп’ютері користувача, яка не використовує ніяких периферійних пристроїв і ресурсів, то в цьому випадку немає необхідності в розробці додаткових діаграм. Однак при розробці корпоративних додатків ситуація виглядає зовсім інакше.

По-перше, складні програмні системи можуть реалізовуватися в мережевому варіанті на різних обчислювальних платформах і технологіях доступу до розподілених баз даних. Наявність локальної корпоративної мережі

потребує розв’язання цілого комплексу додаткових завдань щодо раціонального розміщення компонентів по вузлах цієї мережі, що визначає загальну продуктивність програмної системи.

По-друге, інтеграція програмної системи з мережею Інтернет визначає необхідність розв’язання додаткових завдань при проектуванні системи, таких як забезпечення безпеки, криптозахищеності й стійкості доступу до інформації для корпоративних клієнтів. Ці аспекти залежать від реалізації проекту у формі фізично існуючих вузлів системи, таких як сервери, робочі станції, брандмауери, канали зв’язку і сховища даних.

Нарешті, технології доступу і маніпулювання даними в рамках загальної схеми «клієнт-сервер» вимагають розміщення великих баз даних у різних сегментах корпоративної мережі, їх резервного копіювання, архівування, кешування для забезпечення необхідної продуктивності системи в цілому. Ці аспекти також вимагають візуального представлення з метою специфікації програмних і технологічних особливостей реалізації розподілених архітектур.

Розподіл елементів (результатів розробки) програмного додатку за апаратними вузлами комп’ютерної системи зручно показати за допомогою діаграм розгортання (у російській (українській) літературі їх називають також діаграмами розміщення).

З точки зору реалізації проектована система складається з компонентів (представлених на діаграмах компонентів), розподілених по обчислювальним вузлам (представленим на діаграмах розміщення).

Діаграма розгортання дозволяє зобразити «фізичну» структуру програмної системи. На етапі проектування діаграма розгортання використовується для представлення фізичної сукупності вузлів як основи для реалізації системи. Діаграма розгортання складається з трьох основних елементів: артефактів, вузлів і відносин між ними.

Діаграма розгортання відображає відповідність конкретних програмних артефактів (наприклад, виконуваних файлів) обчислювальним вузлам (виконуючим обробку). Вони показують розміщення програмних елементів у

фізичній архітектурі системи і взаємодію (зазвичай мережеву) між фізичними елементами. Діаграма розгортання дозволяє краще зрозуміти фізичну архітектуру (або архітектуру розгортання).

Тобто, на діаграмі розгортання розміщаються тільки ті елементи фізичного представлення моделі, які існують під час виконання програмної системи. Ті елементи, які не використовуються на етапі виконання, на діаграмі розгортання, як правило, не показуються. Так, наприклад, компоненти з текстами програм можуть бути присутніми на діаграмі компонентів, а на діаграмі розгортання вони не вказуються.

Слід зазначити, що на відміну від діаграм логічного представлення і діаграм компонентів, діаграма розгортання проектується, як правило, в єдиному екземплярі, оскільки вона повинна цілком представляти особливості топології і реалізації усієї системи.

Графічно діаграма розгортання – це граф з вузлів, які з'єднані асоціаціями, що показують існуючі комунікації. Вузли можуть містити артефакти (виконуваних компонентів і динамічних бібліотек), що працюють у цих вузлах.

Вузол (node) є деяким фізично існуючим елементом системи, що має деякий обчислювальний ресурс. В якості обчислювального ресурсу вузла може розглядатися наявність щонайменше деякого об'єму електронної або магнітооптичної пам'яті і/або процесора. Поняття вузла також може включати і інші механічні або електронні пристрої, такі як датчики, принтери, модеми, цифрові камери, сканери і маніпулятори.

Графічно на діаграмі розгортання вузол зображується у формі тривимірного куба, який має власне ім'я, що вказується усередині цього графічного символу. Самі вузли можуть представлятися як в якості типів (ім'я вузла без підкреслення, рисунок 2.50, а), так і в якості екземплярів (рисунок 2.50, б, в) і стереотипів у формі рисунку (рисунок 2.50, г). На цьому типі вузлів не можуть розміщуватися виконувані компоненти програмної системи.

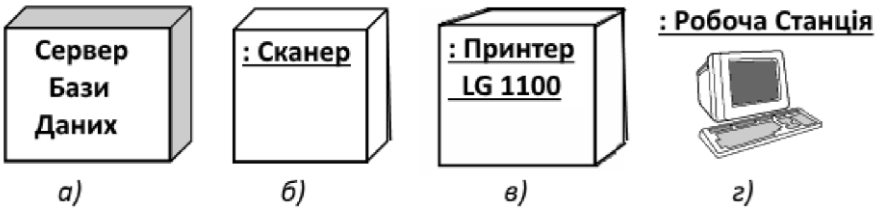


Рисунок 2.50 – Графічне зображення вузла на діаграмі розгортання

При необхідності явно вказати компоненти, які розміщуються або виконуються на окремому вузлі, можна зробити двома способами. Перший з них дозволяє розділити графічний символ вузла на дві секції горизонтальною лінією. У верхній секції записують ім'я вузла, а в нижній – розміщені на цьому вузлі компоненти (рисунок 2.51, а).

Другий спосіб дозволяє показувати на діаграмі розгортання вузли з вкладеними зображеннями компонентів (рисунок 2.51, б). Такими вкладеними компонентами можуть виступати тільки виконувані компоненти і динамічні бібліотеки.

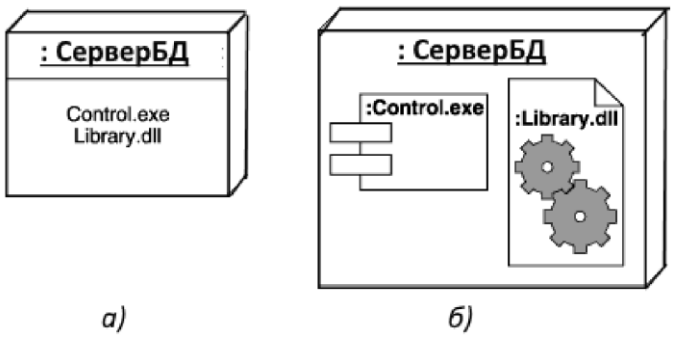


Рисунок 2.51 – Варіанти графічного зображення вузлів-екземплярів з розміщуваними на них компонентами

Рисунок 2.52 – Фрагмент діаграми розгортання зі з’єднаннями між вузлами

З’єднання (шляхи комунікації) вказують відношення між вузлами, які є різновидом асоціації. Зображуються вони відрізками ліній без стрілок. Наявність такої лінії вказує на необхідність організації фізичного каналу для обміну інформацією між відповідними вузлами.

Характер з’єднання може бути додатково специфікований приміткою, поміченою значенням або обмеженням, які визначаються згідно із загальним визначенням асоціації (рисунок 2.52). У розглянутому прикладі явно визначені не лише вимоги до швидкості передачі даних у локальній мережі за допомогою поміченого значення, але і рекомендації за технологією фізичної реалізації з’єднань у формі примітки.

Окрім з’єднань на діаграмі розгортання можуть бути присутніми відношення залежності між вузлом і розгорнутими на ньому компонентами. Подібний спосіб є альтернативою вкладеному зображенню компонентів усередині символу вузла, що не завжди зручно, оскільки робить цей символ надмірно об’ємним (рисунок 2.53).

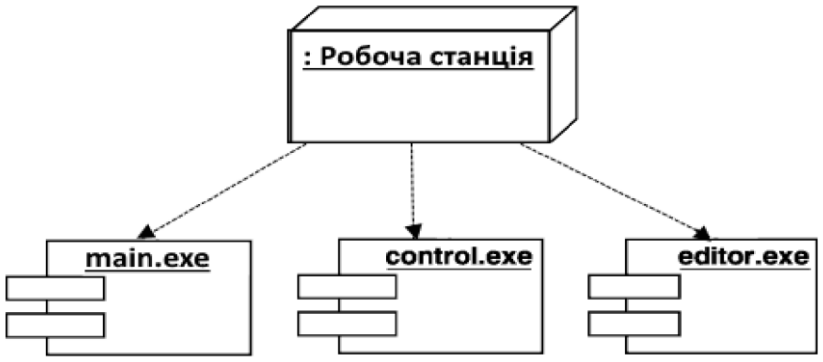


Рисунок 2.53 – Діаграма розгортання з відношенням залежності між вузлом і розгорнутими на ньому компонентами

Розглянемо фрагмент фізичного представлення системи віддаленого обслуговування клієнтів банку (рисунок 2.54). На діаграмі розгортання вузлами системи є віддалений термінал (вузол-тип) і сервер банку (вузол-екземпляр).

Вказана залежність компонента реалізації діалогу «dialog.exe» на віддаленому терміналі від інтерфейсу lAuthorise (Авторизація), реалізованого компонентом «main.exe».

У свою чергу, компонент «main.exe» розгорнутий на анонімному вузлі-екземплярі «Сервер банку». Останній залежить від компонента бази даних «Клієнти банку», який розгорнутий на цьому ж вузлі.

Примітка вказує на необхідність використання захищеної лінії зв'язку для обміну даними в цій системі. Інший варіант запису цієї інформації полягає в доповненні діаграми вузлом із стереотипом «закрита мережа».

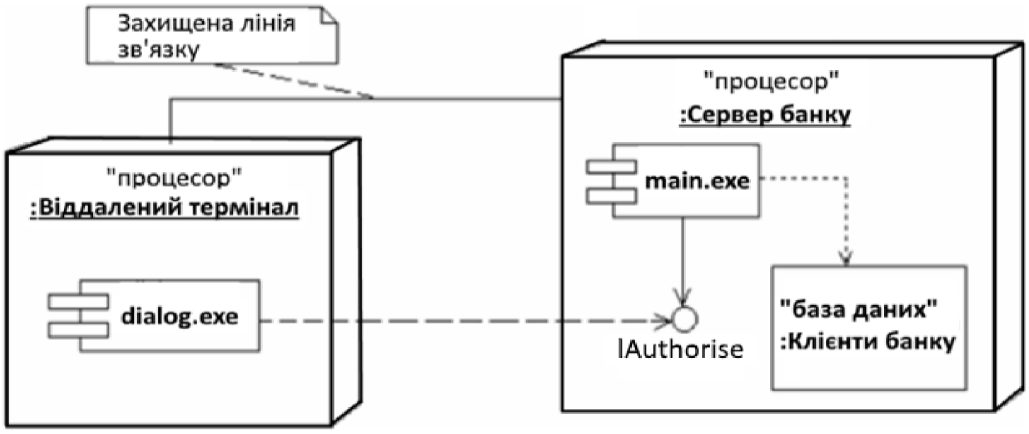


Рисунок 2.54 – Діаграма розгортання для системи віддаленого обслуговування клієнтів банку

2.3.7.5 Моделювання поведінки системи

При створенні програмної системи недостатньо відповісти на питання що робить система? і з чого вона складається? – потрібно відповісти на питання як працює система?. Відповідь на це питання дає модель поведінки. Поведінка реальної програми цілком і повністю визначається її кодом – як програма складена, так вона і виконується. Але програма – це просто запис алгоритму. Таким чином, модель поведінки (behavior model) – це опис алгоритму роботи системи [6, розділи 12, 18].

Поведінка системи визначається множиною об’єктів, що обмінюються повідомленнями, і задається діаграмами діяльності, діаграмами взаємодії діаграмами кінцевого автомата.

Для опису потоку управління, тобто послідовності виконуваних елементарних кроків при виконанні окремої операції або реалізації складного варіанта використання, зручно використовувати діаграму діяльності.

Взаємодія декількох програмних об’єктів між собою описується діаграмами взаємодії в одній з двох еквівалентних форм (діаграми послідовності або діаграми комунікації).

У UML передбачено декілька різних засобів для опису поведінки. Вибір того чи іншого засобу диктується типом поведінки, яку потрібно описати.

Для опису життєвого циклу конкретного об’єкта, поведінка якого залежить від історії цього об’єкта, або обробки асинхронних стимулів, використовується кінцевий автомат у формі діаграми станів. При цьому стани кінцевого автомата відповідають станам об’єкта, тобто різним наборам значень атрибутів, а переходи відповідають виконанню операцій.

Як і діаграми видів діяльності, діаграми кінцевих автоматів (станів) UML відображають динамічну модель системи. За їх допомогою ілюструються події і стани об’єктів – транзакцій, прецедентів, людей тощо.

2.3.7.5.1 Діаграма діяльності

Опис функціональних вимог у вигляді варіантів використання, безсумнівно, є дуже важливим етапом у життєвому циклі розробки програмного забезпечення. Адже саме на цьому етапі відбувається узгодження із замовником того, як буде виглядати і функціонувати система. Тепер на підставі затверджених вимог необхідно систему спроектувати і реалізувати. Завдання аналітика на цьому етапі – перетворити опис варіантів використання в технічно грамотний опис структури й поводження системи, ясну архітекторам і розробникам. Таким чином, перехід від моделювання використання до інших видів моделювання полягає в уточненні, деталізації та конкретизації варіантів використання.

Якщо діаграма варіантів використання дає «вид зверху» на функціональність системи, то діаграма діяльності, навпаки, дозволяє докладно ілюструвати окремий варіант використання та його сценарії.

Діаграми діяльності, так само як і блок-схеми, – загальний та потужний засіб опису алгоритмів. Їх з успіхом також можна застосовувати для моделювання поведінки людей, пристроїв і організацій при виконанні бізнес-процесів.

Під діяльністю в даному випадку розуміють завдання (операцію), які необхідно виконати вручну або за допомогою засобів автоматизації. У теоретичному плані діаграма діяльності – це узагальнене уявлення алгоритму, що реалізує аналізований варіант використання. У загальному випадку діяльність є самостійним елементом поведінки, яка, у свою чергу, може включати інші діяльності або окремі дії.

Діаграми діяльності в UML забезпечені додатковими синтаксичними засобами, які різко підвищують їх виразність і область застосування.

Дія. Дія є іменованим елементом, якому відповідає один крок або етап при виконанні деякої діяльності. У результаті виконання дії може змінитися стан системи. Кожна дія в діяльності може виконуватися довільну кількість разів.

Вузол діяльності. Вузол діяльності є абстрактним класом для окремих точок у потоці діяльності, сполучених дугами. Вузли діяльності можуть відноситися тільки до окремої діяльності або групи діяльностей. Існує декілька видів конкретних вузлів діяльності, для кожного з яких є власна графічна нотація: вузол дії (рисунок 2.55, *а*), вузол виклику діяльності (рисунок 2.55, *б*), вузол об’єкту (рисунок 2.55, *в*) і вузли управління (рисунок 2.55, *г*).

Рисунок 2.55 – Нотація для різних вузлів діяльності

Вузол виклику діяльності зображується розміщенням символу «граблі», які мають подібність з символом ієрархії, ілюструючи той факт, що цей виклик

діяльності починає іншу діяльність. Остання може бути представлена у формі додаткової вкладеної діаграми діяльності.

Дуга діяльності. На діаграмі вузли діяльності можуть бути пов’язані між собою за допомогою дуг діяльності. Дуга діяльності є спрямованим відношенням і має в якості джерела й мети деякі вузли.

Дуга діяльності зображується суцільною лінією з тонкою стрілкою, при цьому ім’я дуги може бути відсутнім (рисунок 2.56, а). Якщо ж дуга має ім’я, то воно записується з рядкової букви і вказується поблизу стрілки (рисунок 2.56, б). Дуга діяльності може бути також зображена з використанням з’єднувача, який представляється у формі невеликого круга з ім’ям мітки (цифри або букви) для цієї дуги (рисунок 2.56, в). Це позначення прийняте виключно з метою зручності й не робить впливу на модель.

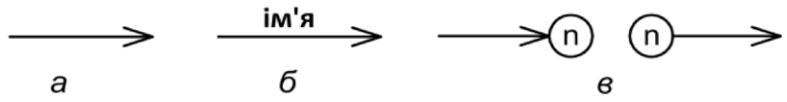


Рисунок 2.56 – Нотація для різних дуг діяльності

Вузли розгалуження і злиття. Вузол розгалуження має одне ребро, що входить, і два й більше альтернативних ребер, що виходять (рисунок 2.57, а). Маркер, що поступає по ребру буде запропонований усім ребрам, що виходять, але пройде тільки по одному з них. Вузол розгалуження – це перехрестя потоків, на якому маркер повинен вибрати тільки один шлях. Кожне вихідне ребро захищене сторожовою умовою, яка означає, що ребро прийме маркер тільки у разі виконання сторожової умови. У вузлах злиття сходяться два або більше ребер і виходить лише одне ребро. Вони об’єднують усі потоки, що входять, в один потік, що виходить (рисунок 2.57, б).

Вузли розгалуження і об’єднання (паралелізм). Вузол розгалуження розділяє потік на декілька паралельних потоків (рисунок 2.57, в). Вузол об’єднання синхронізує і об’єднує декілька потоків в єдиний потік, що виходить (рисунок 2.57, г).

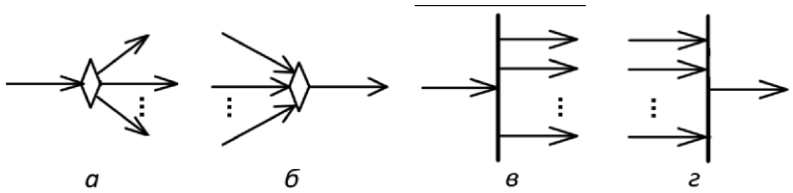


Рисунок 2.57 – Вузли розгалуження (*а*), злиття (*б*), паралельне розгалуження (*в*) і об’єднання (*г*) 111

На рисунку 2.58 наведено діаграму діяльності, побудовану на прикладі процесу обробки звернення клієнта до менеджера інтернет-магазину і подальшого оформлення замовлення.

У загальному випадку дії на діаграмі діяльності виконуються над тими або іншими об’єктами. Ці об’єкти або ініціюють виконання дій, або визначають деякий результат цих дій. При цьому дії специфікують виклики, які передаються від одного об’єкту графа діяльності до іншого.

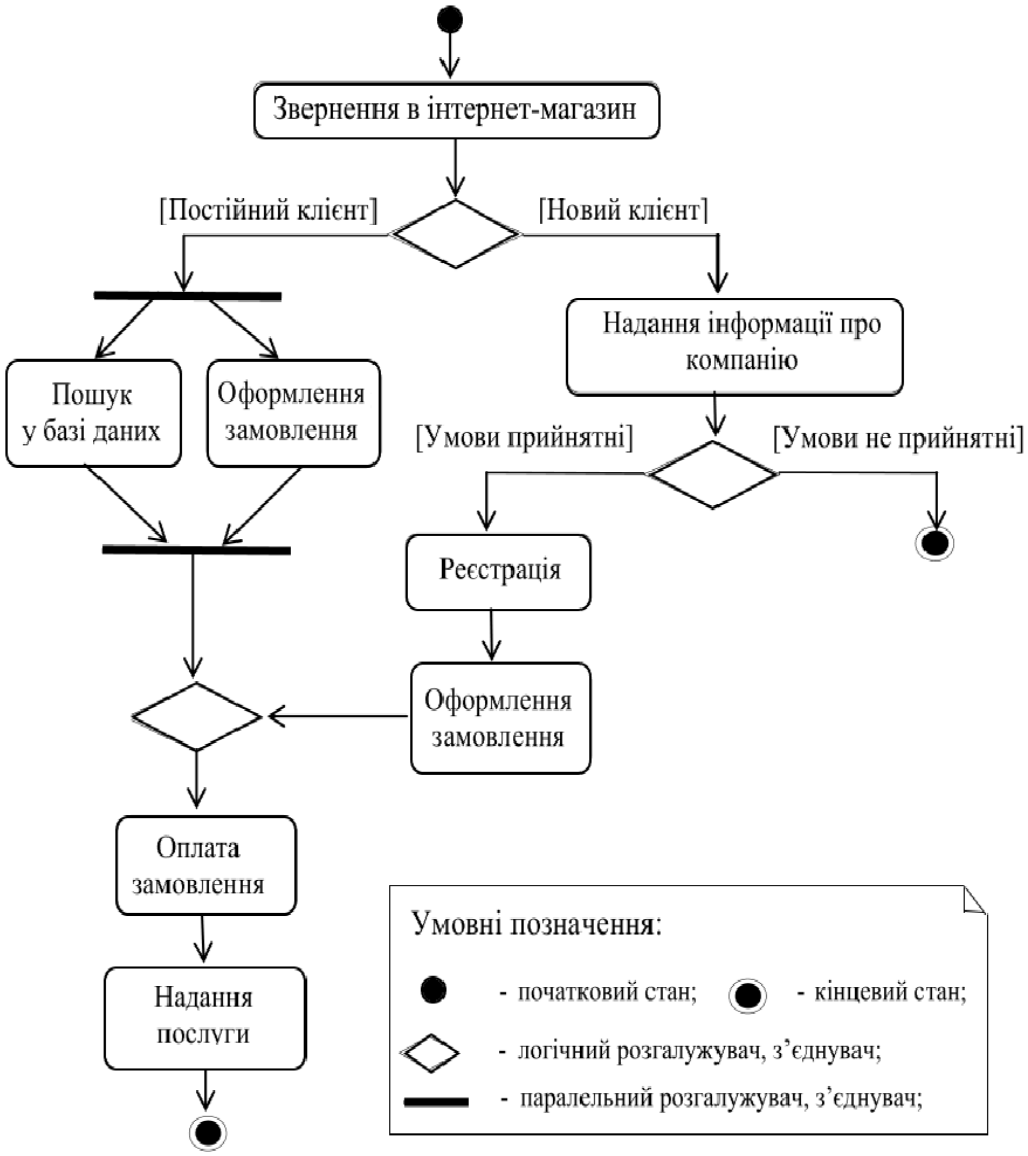


Рисунок 2.58 – Приклад діаграми діяльності

Для детальної ілюстрації особливості зображення розгалуження і паралельних підпроцесів розглянемо ще один приклад з вибором напою і приготуванням кави. Особливість цього прикладу полягає в тому, що він не вимагає ніяких додаткових пояснень в силу очевидності свого контексту (рисунок 2.59) [8]. Паралельність у даному прикладі проявляється в тому, що пошуком чашки бармен може зайнятися під час приготування кави.

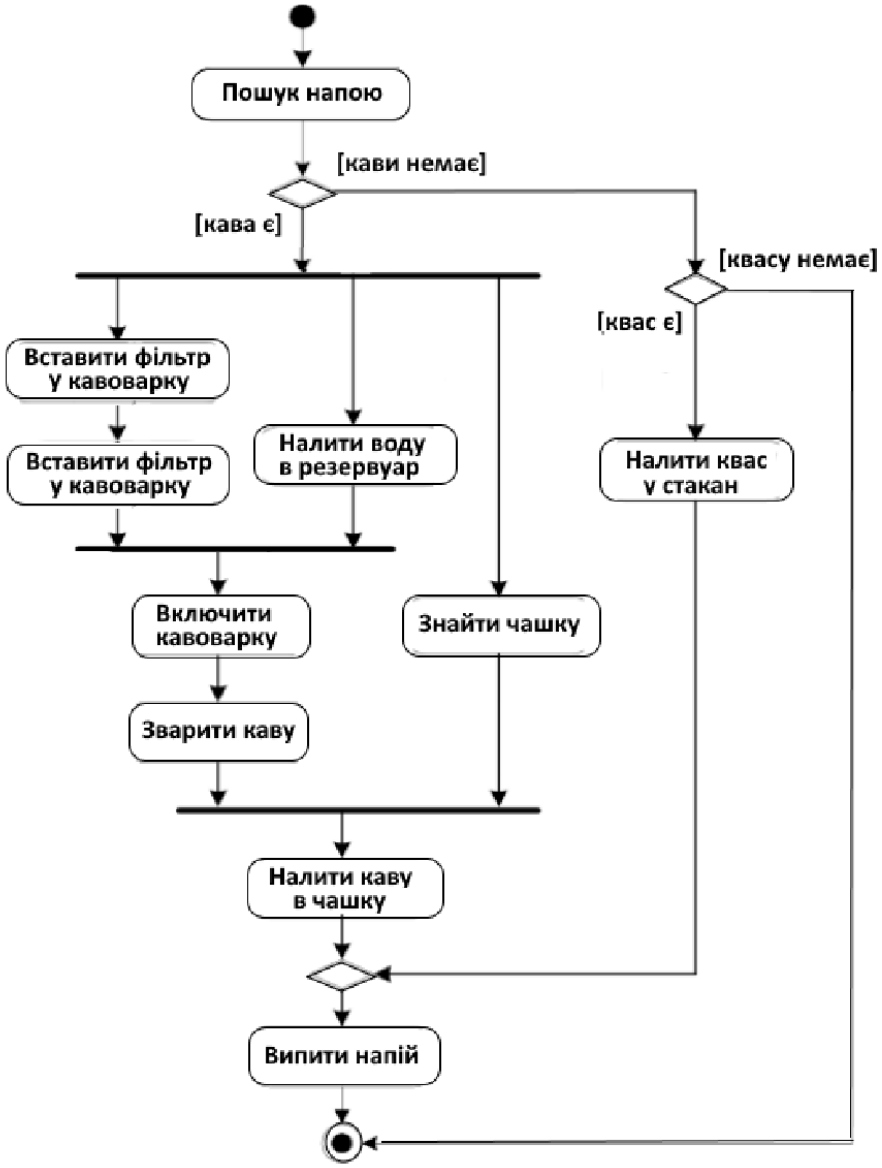


Рисунок 2.59 – Діаграма діяльності для прикладу з приготуванням напою

Діаграми діяльності можуть бути використані не лише для специфікації алгоритмів обчислень або потоків управління у програмних системах. Не менш важлива сфера їх застосування пов’язана з моделюванням бізнес-процесів. Для моделювання цих особливостей в мові UML використовується спеціальна конструкція, що отримала назву доріжки (рисунок 2.60).

При цьому усі стани дії на діаграмі діяльності діляться на окремі групи, які відділяються один від одного вертикальними лініями. Дві сусідні лінії і утворюють доріжку, а група станів між цими лініями виконується окремим підрозділом компанії (Відділ замовлень, Відділ продажів, Склад).

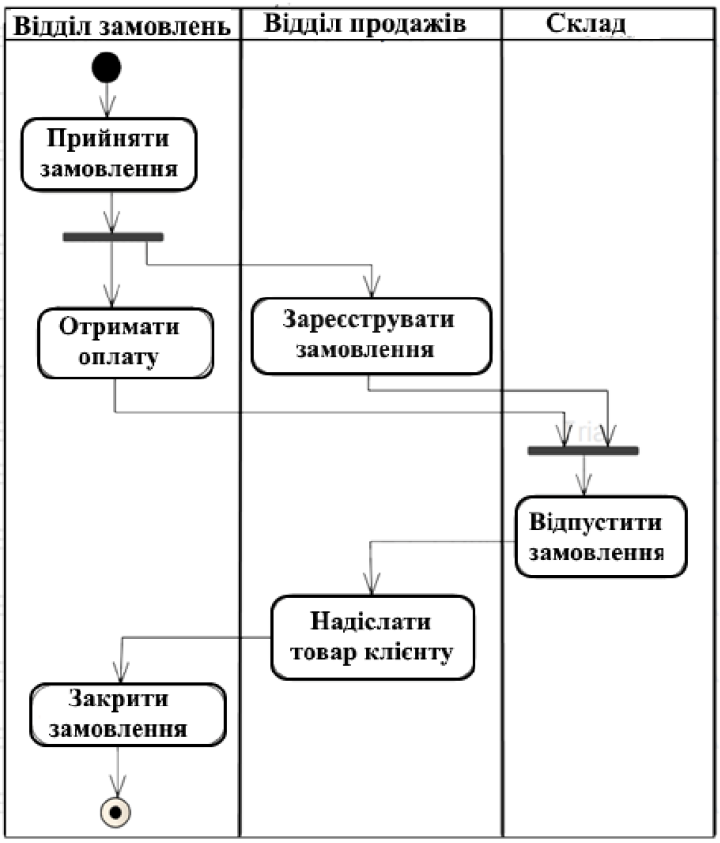


Рисунок 2.60 – Варіант діаграми діяльності з доріжками

Підрозділами компанії є відділ прийому й оформлення замовлень, відділ продажів і склад. Цим підрозділам відповідатимуть три доріжки на діаграмі діяльності, кожна з яких специфікує зону відповідальності підрозділу. У цьому випадку діаграма діяльності містить в собі не лише інформацію про послідовність виконання робочих дій, але й про те, який підрозділ торгової компанії повинен виконувати ту або іншу дію.

Відділом прийому і оформлення замовлень фіксується отримане від клієнта замовлення, після чого він реєструється у відділі продажів. Потім здійснюється розпаралелювання діяльності на два потоки. Перший з них залишається в цьому ж відділі та пов’язаний з отриманням оплати від клієнта за замовлений товар. Другий ініціює виконання дії з реєстрації замовлення у відділі продажів (модель товару, розміри, рік випуску та ін.) і передачі товару на склад.

Проте видача товару зі складу починається тільки після того, як від клієнта буде отримана плата за товар (перехід-злиття). Потім виконується підготовка

товару до відправки і його відправка клієнтові у відділі продажів. Після завершення цих дій замовлення закривається у відділі прийому і оформлення замовлень.

2.3.7.5.2 Діаграма послідовності

Діаграми взаємодії призначені для моделювання поведінки шляхом опису взаємодії об'єктів для виконання деякої задачі або досягнення певної мети. У програмній системі об'єкти повинні взаємодіяти один з одним, посилаючи повідомлення. Взаємодія – це така поведінка на основі обміну повідомленнями між набором об'єктів, яка забезпечує реалізацію вимог до системи.

Діаграми взаємодії застосовуються на різних рівнях моделювання: як для опису поведінки окремих операцій, так і цілих варіантів використання. Даний тип діаграм дозволяє описувати не тільки взаємодію програмних об'єктів (примірників класів), а й взаємодію примірників інших класифікаторів: дійових осіб, варіантів використання, компонентів тощо.

Діаграми взаємодії зображуються в декількох різних графічних формах, з яких найважливішими є діаграми послідовності і діаграми комунікації.

Діаграми послідовності та діаграми комунікації семантично еквівалентні, хоча графічно виглядають зовсім по-різному. Семантично ці діаграми еквівалентні тому, що описують одне і те ж: послідовність передачі повідомлень між об'єктами у процесі їх взаємодії.

Виглядають по-різному вони тому, що в діаграмі послідовності графічно підкреслюється впорядкованість в часі переданих повідомлень, у той час як в діаграмі комунікації на передній план висувається структура зв'язків між об'єктами, за якими передаються повідомлення.

Діаграма послідовності призначена для моделювання поведінки у формі опису протоколу сеансу обміну повідомленнями між взаємодіючими екземплярами класифікаторів під час виконання одного з можливих сценаріїв.

На діаграмі послідовності вважається виділеним один напрямок, відповідно до течії часу. За умовчанням прийнято, що час тече зверху вниз, але це не обов'язково, наприклад, можна вважати, що час тече зліва направо, обумовивши

це спеціальним коментарем.

Паралельно осі часу від усіх учасників взаємодії об’єктів відходить пряма пунктирна лінія, яка називається лінією життя. Лінія життя представляє об’єкт у взаємодії. Якщо стрілка відходить від лінії життя об’єкта, то це означає, що даний об’єкт відправляє повідомлення, а якщо стрілка повідомлення входить у лінію життя, то це означає, що даний об’єкт отримує повідомлення. Якщо ж стрілка перетинає лінію життя об’єкта, то це нічого не значить – повідомлення пролетіло повз. Якщо у процесі взаємодії об’єкт закінчує своє існування, то лінія життя обривається і в цьому місці ставиться косий хрест. Над стрілкою повідомлення вказується текстова частина.

Фокус управління призначений для виділення активності об’єктів і зображується на діаграмі у вигляді вузького прямокутника, верхня сторона якого є початком отримання фокусу управління об’єкту (початок активності), нижня сторона – закінченням фокусу управління (закінчення активності). Періоди активності можуть чергуватися з періодами пасивності або очікування (декілька фокусів управління).

Ініціатором взаємодії в системі може бути актор або зовнішній користувач. Актор може мати власне ім’я або залишатися анонімним. Найчастіше актор і його фокус управління існують в системі постійно.

Об’єкт може ініціювати рекурсивну взаємодію з самим собою. На діаграмі послідовності рекурсія – це невеликий прямокутник, приєднаний до правої сторони фокусу управління того об’єкту, для якого зображується ця рекурсивна взаємодія (рисунок 2.61).

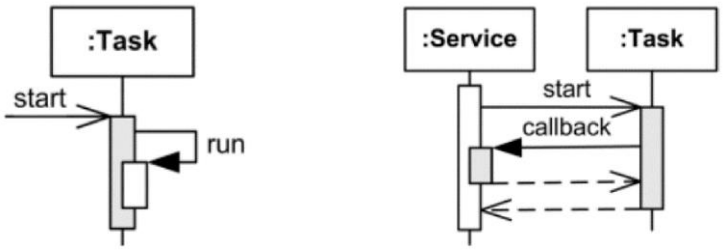


Рисунок 2.61 – Рекурсія на діаграмі послідовності

Повідомлення – закінчений фрагмент інформації, що направляється одним об’єктом іншому. Прийом повідомлення ініціює виконання певних дій, спрямованих на рішення окремої задачі тим об’єктом, якому це повідомлення відправлене. Повідомлення не лише передає деяку інформацію, але і вимагає або припускає виконання очікуваних дій від приймаючого об’єкту.

Повідомлення можуть мати аргументи або параметри, залежно від конкретних значень яких може бути отриманий різний результат. Значення параметрів окремих повідомлень можуть містити умовні вирази – розгалуження або альтернативні шляхи основного потоку управління

Повідомлення зображуються горизонтальними стрілками, що сполучають лінії життя або фокуси управління двох об’єктів на діаграмі послідовності.

Різні типи повідомлень зображені на рисунку 2.62 (а – синхронний виклик – відправка повідомлення і очікування реакції на нього; б – асинхронний виклик – відправка повідомлення і подальше продовження роботи без очікування реакції; в – повідомлення реакції – показує повідомлення у відповідь для повідомлення синхронного виклику; г – повідомлення видалення – використовується для видалення лінії життя іншого об’єкту).

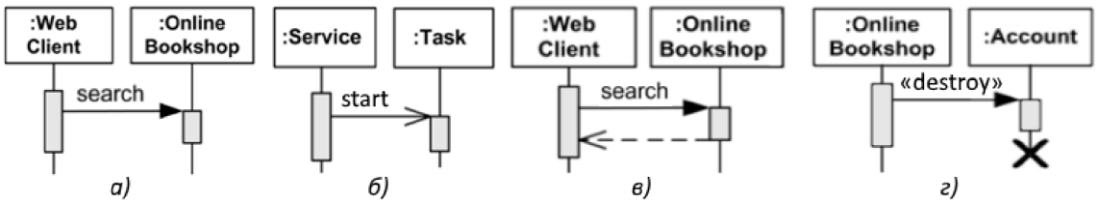


Рисунок 2.62 – Типи повідомлень на діаграмі послідовності

Коментарі або примітки можуть включатися в діаграми послідовності, асоціюючись з окремими об’єктами або повідомленнями. Використовується стандартне позначення для коментаря у вигляді прямокутника із загнутим правим верхнім кутом.

Для запису часових обмежень використовуються фігурні дужки, які розміщуються поряд з початком стрілки відповідного повідомлення, Часові

обмеження можуть відноситися до виконання певних дій об’єктами і до повідомлень, специфікуючи умови їх передачі або прийому, наприклад:

- {час_прийому_повідомлення, час_відправки_повідомлення < 1 с};
- {час_очікування_відповіді < 5 с};
- {час_передачі_пакету < 10 с};
- {об’єкт_1. час_подання_сигналу_тривоги > 30 с}.

Графічно діаграма послідовності – це двовимірна схема, яка показує узагальнені об’єкти (ролі), які розміщені вздовж горизонтальної осі, і повідомлення, впорядковані за часом, уздовж вертикальної осі (лінія життя). Прецеденти визначають, як зовнішні виконавці взаємодіють з програмною системою. У процесі цієї взаємодії виконавцем генеруються системні події, які являють собою запити на виконання деякої системної операції.

На діаграмі послідовності зображуються виключно ті об’єкти, які безпосередньо беруть участь у взаємодії і не показуються можливі статичні асоціації з іншими об’єктами.

Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об’єктів в часі. При цьому діаграма послідовності має як би два виміри.

Один вимір – зліва направо у вигляді вертикальних ліній, кожна з яких зображує лінію життя окремого об’єкта, що бере участь у взаємодії. Графічно кожен об’єкт зображується прямокутником і розташовується у верхній частині своєї лінії життя.

Другий вимір діаграми послідовності – це вертикальна часова вісь, спрямована зверху вниз. Початковому моменту часу відповідає сама верхня частина діаграми. При цьому взаємодія об’єктів реалізується за допомогою повідомлень, які надсилаються одними об’єктами іншим (рисунку 2.63).

Ми вже визначили класи сценарію Оформлення замовлення з нашого прикладу на діаграмі класів, тепер за допомогою діаграми послідовності покажемо, як взаємодіють об'єкти цих класів в часі.

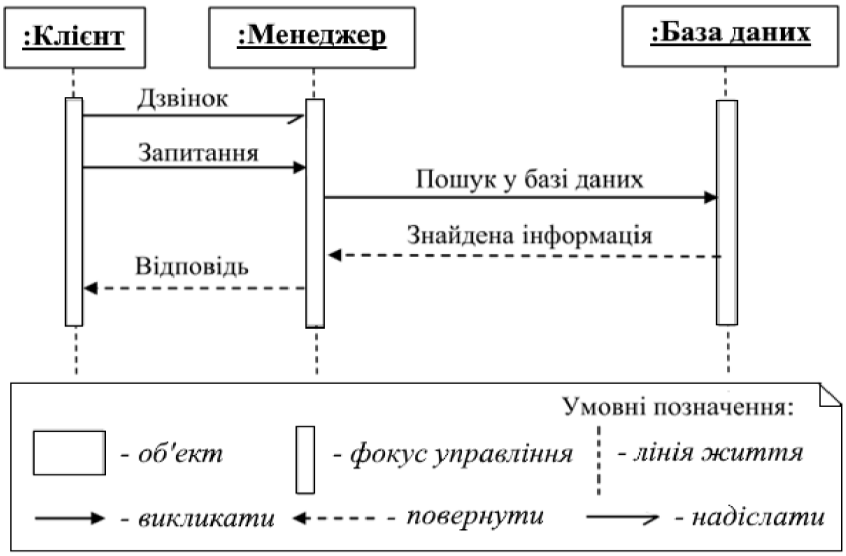


Рисунок 2.63 - Фрагмент діаграми послідовності

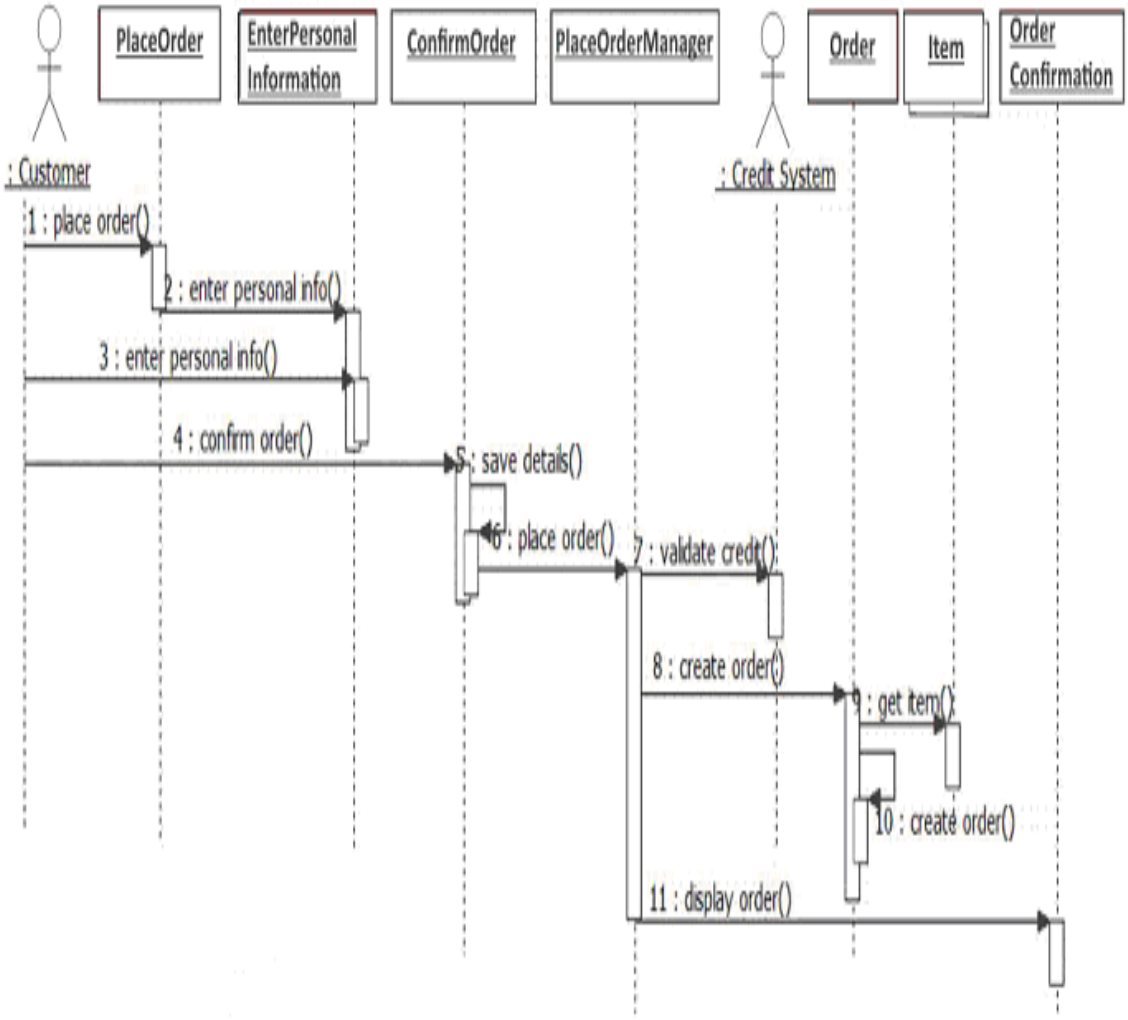


Рисунок 2.64 – Діаграма послідовності сценарію Оформлення замовлення

Складемо діаграму послідовності для випадку, коли покупець успішно оформляє замовлення (рисунок 2.64).

Покупець вибирає опцію «Оформити замовлення» (place order), при цьому викликається деякий об'єкт PlaceOrder (це буде граничний об'єкт, що належить відповідному граничному класу). Далі відкривається форма введення особистих даних покупця і його кредитної картки (EnterPersonalInformation), на ній покупець вводить своє ім'я, адресу, телефон, адресу електронної пошти (enter personal information) і кредитні дані.

Інформація приймається і відкривається форма підтвердження замовлення (ConfirmOrder), покупець підтверджує, що згоден з реквізитами замовлення (confirm order), деталі замовлення зберігаються для подальшого використання (save the details). Фокус управління передається деякому управляючому об'єкту (PlaceOrderManager), який звертається до зовнішньої кредитної системи (Credit System) для проведення платежу. Якщо платіж пройшов успішно (а саме такий сценарій ми зараз і розглядаємо), то PlaceOrderManager посилає повідомлення (create order) створити об'єкт Замовлення (Order), потім викликає форму підтвердження замовлення (OrderConfirmation). Об’єкт Замовлення (Order) звертається до об’єктів Товар (Item) для того, щоб отримати інформацію про товари і створює замовлення. Процес завершується.

Зверніть увагу, що символ об’єкту Товар(Item) на діаграмі послідовності відрізняється від символів інших об’єктів, оскільки був завдань множинний екземпляр класу. Дійсно, замовлення може складатися з декількох товарів, це означає, що об’єкту Замовлення(Order) необхідно отримати інформацію про декілька об’єктів Товар(Item). Для цього використовуємо нотацію UML множинного екземпляра класу, який представляє одним значком декілька об'єктів.

Діаграми послідовності вважають найпопулярнішим засобом представлення детальних вимог на етапі аналізу. Мета перетворення – підвищити рівень точності, формалізувати вимоги, привести їх до виду, необхідною для розв’язання задач проектування. На рисунку 2.65 приведений приклад ще одної детальної діаграми послідовності «Зняття грошей через банкомат».

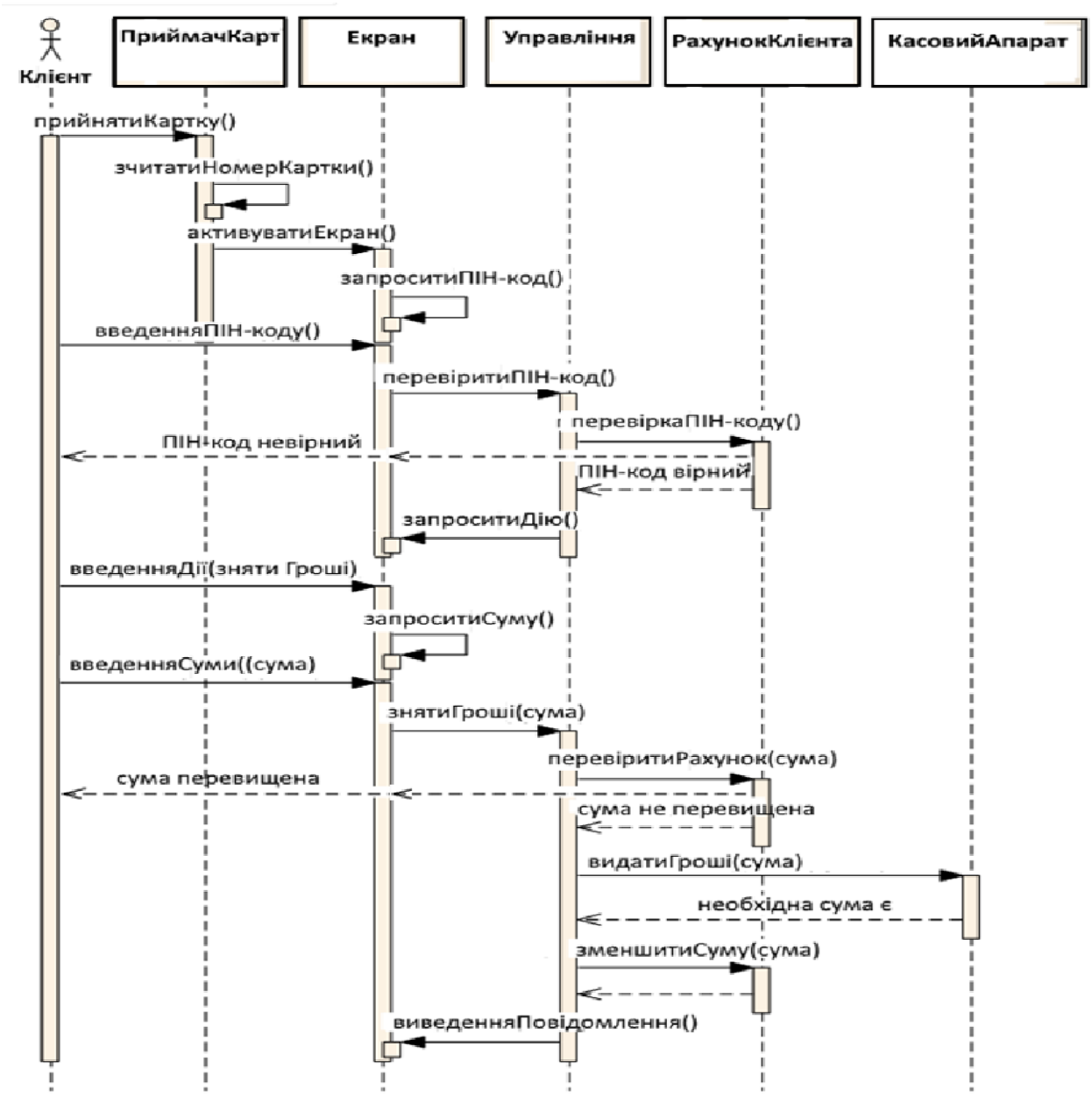


Рисунок 2.65 – Діаграми послідовності «Зняття грошей через банкомат»

2.3.7.5.3 Діаграма комунікації

Діаграма комунікації (у UML 1 – діаграма кооперації) так само як і діаграма послідовності описує поведінку як взаємодію, тобто як протокол обміну повідомлень між об’єктами. Один і той же об’єкт може брати участь у різних взаємодіях, граючи в них різні ролі. Таким чином, взаємодія завжди відбувається в певному контексті, який визначається множиною учасників у взаємодії об’єктів і зв’язків.

Замість того щоб малювати кожного учасника у вигляді лінії життя і

показувати послідовність повідомлень, розташовуючи їх по вертикалі, як це робиться в діаграмах послідовності, комунікаційні діаграми допускають довільне розміщення великої кількості об’єктів, дозволяючи малювати зв’язки, що показують відношення об’єктів, і використовувати нумерацію для представлення послідовності повідомлень.

Для створення діаграми комунікації треба розташувати об’єкти, що беруть участь у взаємодії, у вигляді вершин графа. Потім зв’язки, що з’єднують ці об’єкти, зображуються у вигляді дуг цього графа. Зв’язки доповнюються повідомленнями, які об’єкти приймають і посилають повідомлення.

Діаграма комунікації характеризуються двома особливостями, що відрізняють їх від діаграм послідовностей: шляхом (зв’язком) і порядковим номером повідомлення. Числа перед іменами повідомлень вказують на відносну послідовність повідомлень.

Шлях або зв’язок (Link) використовується для опису зв’язку одного об’єкту з іншим. Зв’язки не мають власних імен і кратності, але можуть використовувати такі стереотипи:

- «association» – вказує, що зв’язок є асоціацією;
- «parameter» – вказує, що зв’язок є параметром деякого методу;
- «local» – вказує, що зв’язок є локальною змінною методу, зона видимості якої обмежена тільки сусіднім об’єктом;
- «global» – вказує, що зв’язок є глобальною змінною, зона видимості якої поширюється на усю діаграму комунікації;
- «self» – вказує, що зв’язок є зв’язком рефлексії об’єкту з самим собою (зображується петлею у верхній частині прямокутника об’єкту).

Порядковий номер повідомлення використовується для позначення тимчасової послідовності повідомлень.

Для сценарію Оформлення замовлення, для якого мі вже склали діаграму послідовності, побудуємо діаграму кооперації, помістивши на неї усі ті ж об'єкти (рисунок 2.66).

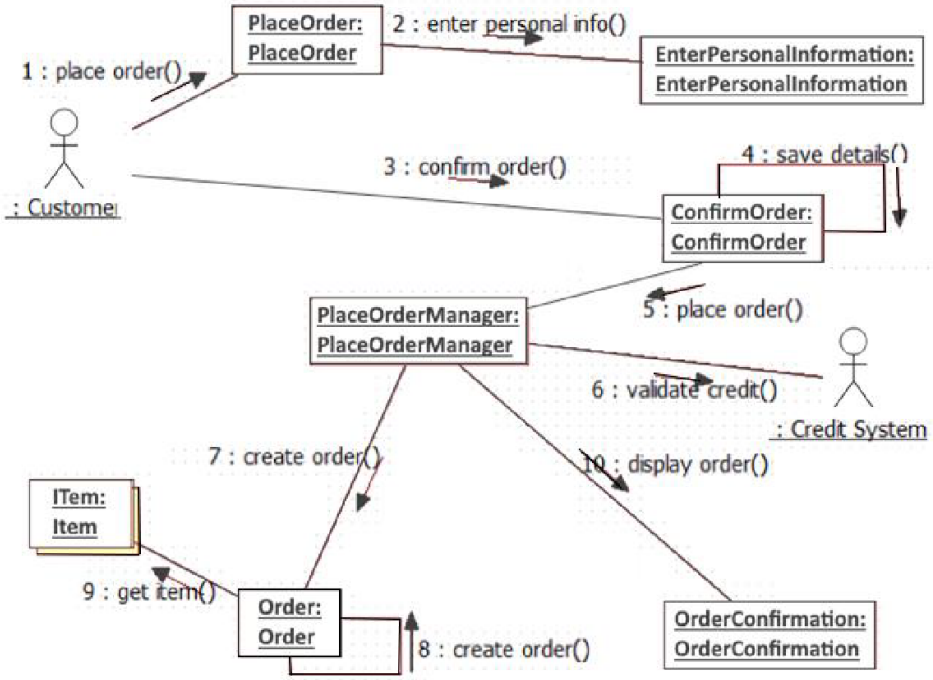


Рисунок 2.66 – Діаграма комунікації сценарію Оформлення замовлення

Розглянемо ще один приклад діаграми комунікації, як варіант моделі комунікації внутрішніх частин банкомату при знятті готівки (рисунок 2.67) [8].

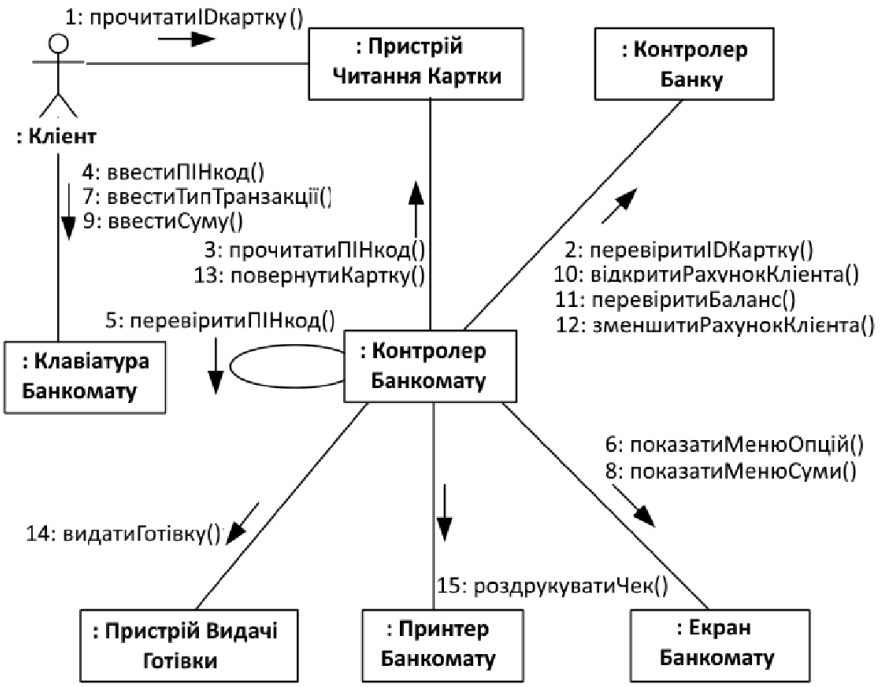


Рисунок 2.67 – Діаграма комунікації для банкомату

Усі зв'язки на діаграмі специфіковані, на їх кінцях вказана інформація у формі ролей зв'язків. Щоб відобразити структурну організацію потоків управління, на діаграмі зображені усі повідомлення з вказівкою їх порядку і семантичних особливостей.

2.3.7.5.4 Діаграма скінченного автомату

Діаграма кінцевого автомата (діаграма станів) відображає кінцевий автомат, виділяючи потік управління, наступний від стану до стану. Кінцевий автомат – поведінка, яка визначає послідовність станів в ході існування об’єкту. Більш точно, діаграма визначає для кожного стану об’єкту – дію, яка виконується об’єктом при отриманні ним сигналу про подію. Один і той же об’єкт може виконувати різні дії у відповідь на одну і ту ж подію в залежності від стану об’єкту. Виконання дії, як правило, викликає зміну стану. Тобто, у цілому моделювання станів ґрунтується на поняттях «подія» і «стан».

Модель станів може включати декілька діаграм станів по одному на кожен клас, поведінка якого в часі важлива для проектованого додатку. Клас, що має декілька станів, має важливу поведінку в часі. Якщо клас має тільки один стан, то його поведінку в часі можна ігнорувати.

Діаграма станів UML більш наочні й виразні в порівнянні з класичними уявленнями автоматів, але їх застосування вимагає більшої підготовленості користувача і пред’являє більш високі вимоги до «кмітливості» і «уважності» інструментів моделювання.

Діаграма кінцевого автомата задає поведінку системи як цілісної, єдиної сутності, вона моделює життєвий цикл єдиного об’єкту. У силу цього автоматний підхід зручно застосовувати для формалізації динаміки окремого, важкого для розуміння блоку системи.

Діаграма станів має схожу семантику з діаграмою діяльності, тільки діяльність тут замінена станом, переходи символізують дії. Таким чином, якщо для діаграми діяльності відмінність між поняттями «Діяльність» і «Дія» полягає в можливості подальшої декомпозиції, то на діаграмі станів діяльність символізує стан, в якому об’єкт знаходиться тривалий час, у той час як подія моментальна.

Зазвичай діаграми кінцевих автоматів застосовуються для моделювання поведінки подієво-керованих об’єктів.

Застосовувати діаграму станів необхідно тільки для тих класів, які проявляють цікаву поведінку, коли побудова діаграми станів допомагає зрозуміти, як все відбувається. Автомати станів можна також використовувати при моделюванні поведінки графічного інтерфейсу, як реакції на дії користувача.

У загальному випадку кінцевий автомат представляє динамічні аспекти модельованої системи у вигляді орієнтованого графа, вершини якого відповідають станам, а дуги – переходам. При цьому поведінка моделюється як послідовне переміщення по графу станів від вершини до вершини по дугах, що зв’язують їх, з урахуванням їх орієнтації.

На діаграмі станів час знаходження системи в тому або іншому стані явно не враховується, проте передбачається, що послідовність зміни станів впорядкована в часі. Іншими словами, кожен подальший стан може настати пізніше передуючого йому стану.

Стан і його графічне зображення. Стан – умова або ситуація в ході життєвого циклу об’єкту, впродовж якого він задовольняє логічній умові, виконує певну діяльність або чекає події. У формулюванні задач стани часто відповідають дієсловам або дієприслівникам (чекає, додзвонюється) або виконанню деякої умови (включений, нижче запланованої ціни).

Об’єкти класу мають кінцеве число можливих станів. У конкретний момент часу кожен об’єкт може знаходитися рівно в одному стані. Впродовж часу свого існування об’єкти можуть проходити через один або декілька станів.

Стан на діаграмі зображується округленим прямокутником (рисунок 2.68), який може бути розділений на дві секції горизонтальною лінією. Якщо вказана лише одна секція, то в ній записується тільки ім’я стану (рисунок 2.68, а). Інакше в першій з них записується ім’я стану, а у другій – список деяких внутрішніх дій або переходів у цьому стані (рисунок 2.68, б). При цьому під дією в мові UML розуміють деяку атомарну операцію, виконання якої призводить до зміни стану або повернення деякого значення (наприклад, «істина» або «хиба»).

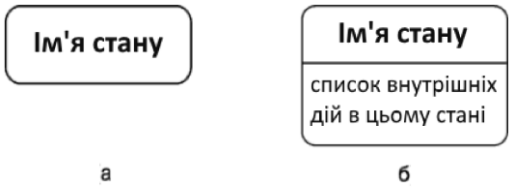


Рисунок 2.68 – Графічне зображення станів на діаграмі кінцевого автомата

Дія. Дія призводить до зміни стану системи, і може бути реалізоване за допомогою передачі повідомлення об’єкту, модифікацією зв’язку або значення атрибуту. Кожна дія записується у вигляді окремого рядка і має наступний формат: мітка дії / вираз дії.

Якщо список виразів дії порожній, то мітка дії вказується без роздільника «/». Перелік міток дій в мові UML фіксований, причому ці мітки не можуть бути використані в якості імен подій:

1 Вхідна дія (entry action) – дія, яка виконується у момент переходу в цей стан.

2 Дія виходу (exit action) – дія, яка вироблена при виході з цього стану.

3 Внутрішня діяльність (do activity) – виконання об’єктом операцій або процедур, які вимагають певного часу.

Як приклад стану розглянемо аутентифікацію клієнта для доступу до ресурсів інформаційної системи (рисунок 2.69, б).

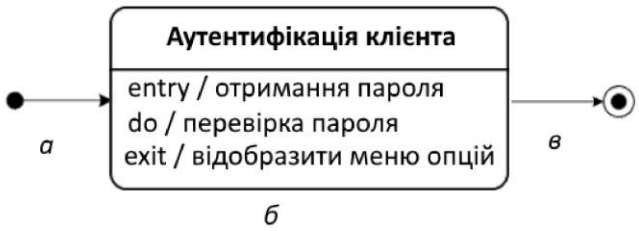


Рисунок 2.69 – Приклад стану з непорожньою секцією внутрішніх дій

Окрім звичайних станів на діаграмі станів можуть розміщуватися псевдостани.

Псевдостан – це вершина в кінцевому автоматі, яка має форму стану, але не має поведінки. Прикладами псевдостанів, які визначені в мові UML, є початковий і кінцевий стани.

Початковий стан – різновид псевдостану, який позначає початок виконання процесу зміни станів кінцевого автомата або знаходження модельованого об'єкту у складеному стані (див. рисунок. 2.69, а).

Кінцевий стан – різновид псевдостану, який обозначає припинення процесу зміни станів кінцевого автомата або знаходження модельованого об'єкту у складеному стані (див. рисунок 2.69, в).

Подія і перехід. Перебування модельованого об'єкту або системи в першому стані може супроводжуватися виконанням деяких внутрішніх дій або діяльності. При цьому зміна поточного стану об'єкту буде можлива або після завершення цих дій (діяльності), або при виникненні деяких зовнішніх подій. В обох випадках говорять, що відбувається перехід об'єкту з одного стану в інший.

Подія (event) – це подія, що трапилася в певний момент часу. Іншими словами, події – це зовнішні дії. Наприклад, натиснення кнопки або виліт рейсу. Події відбуваються в результаті виконання деякої дії в системі або її оточенні, тому в описі задачі вони часто відповідають дієсловам у минулому часі (живлення включене) або виконанню деякої умови (температура підвищилася).

Формально, подія є специфікацією факту, що має місце у просторі і в часі.

Про події говорять, що вони «відбуваються», при цьому окремі події мають бути впорядковані в часі. Після настання події не можна вже повернутися до попередніх, якщо така можливість явно не передбачена в моделі.

Перехід – відношення між двома станами, яке вказує на те, що об'єкт в першому стані повинен виконати певні дії і перейти у другий стан.

В окремих випадках виникає необхідність явно показати ситуацію, коли перехід може мати декілька початкових станів або цільових станів. Такий перехід дістав назву – паралельний перехід.

Введення в розгляд паралельних переходів може бути обумовлене необхідністю синхронізувати і/або розділити окремі процеси управління на

паралельні потоки (нитки) без специфікації додаткової синхронізації в паралельних кінцевих підавтоматах.

Графічно такий перехід зображується вертикальною рискою, аналогічно позначенню переходу в мережах Петрі. Якщо паралельний перехід має дві або більше дуг, що виходять з нього, то його називають розділенням. Якщо ж він має дві або більше дуг, що входять, то його називають злиттям. Текстовий рядок специфікації паралельного переходу записується поряд з рискою і стосується всіх дуг, що входять або виходять.

У загальному випадку поведінка паралельних кінцевих підавтоматів відбувається незалежно один від одного, що дозволяє, наприклад, моделювати багатозадачність у програмних системах.

В окремих ситуаціях може виникнути необхідність врахувати в моделі синхронізацію настання окремих подій і спрацьовування відповідних переходів. Для цієї мети в мові UML є псевдостан, який називається синхронізуючим станом або станом синхронізації.

Стан синхронізації – це псевдостан в кінцевому автоматі, який використовується для синхронізації паралельних областей кінцевого автомата. Синхронізуючий стан позначається невеликим колом, усередині якого розміщений символ зірочки «*». Він використовується спільно з переходом-злиттям або переходом-розділенням для того, щоб явно вказати події в інших кінцевих підавтоматах, що роблять безпосередній вплив на поведінку цього підавтомату.

Повернемося до нашого прикладу Internet-магазину. Покупець оформляє замовлення. Клас Замовлення, окрім інших атрибутів має атрибут «статус». Простежимо динаміку руху замовлень в системі за допомогою діаграми кінцевого автомату, складеної для класу Замовлення.

По умові нашого завдання дані про зроблене замовлення поступають співробітникові відділу продажів, який перевіряє оплату, реквізити замовлення і передає його комірникові на комплектацію. Комірник, перевіривши наявність

замовлених товарів і зібравши замовлення, якщо це можливо, робить відмітку про готовність.

Замовлення видається зі складу комірником. Комірник видає замовлення і відмічає в системі, що замовлення видане. Покажемо на діаграмі перехід замовлення між станами (рисунок 2.70).

Рисунок 2.70 – Діаграма станів об'єкту класу Замовлення

При нормальному завершенні внутрішньої діяльності генерується відповідна подія. У нашому прикладі при оформленні замовлення він має бути сплачений (вхідна дія Сплатити замовлення). Обробка замовлення має на увазі перевірку оплати і наявності товарів (діяльність Перевірити оплату і наявність), перехід в один із станів (На комплектації, Укомплектований, Виданий) означає зміну статусу замовлення. Умова [Покупець не забрав замовлення] викликає перехід в стан Розформований при цьому виконується дія Повернути гроші на картку (рисунок 2.71).

Рисунок 2.71 – Остаточна діаграма станів об'єкту Замовлення

Розглянемо ще одну діаграму станів з паралельними процесами. Так, наприклад, при включенні комп'ютера з деякою мережевою ОС паралельно починається виконання декількох процесів (рисунок 2.72). Зокрема, на діаграмі кінцевого автомату після включення комп'ютера паралельно відбувається перевірка пароля користувача і запуск різних служб.

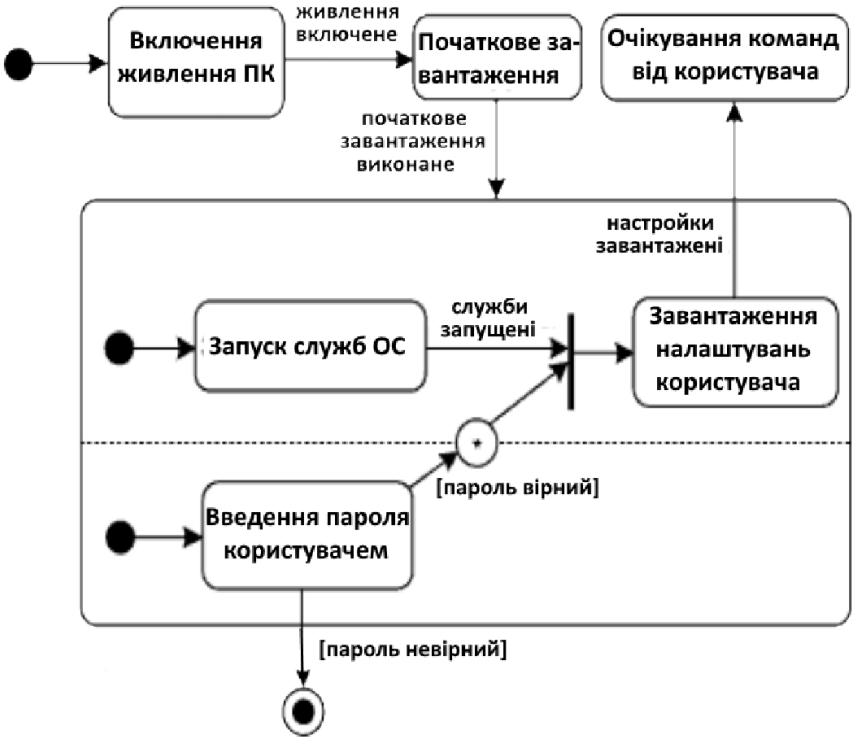


Рисунок 2.72 – Діаграма кінцевого автомату включення комп'ютера

При цьому робота користувача на комп'ютері стане можливою тільки у разі успішної його автентифікації, інакше комп'ютер може бути вимкнений. Розглянуті особливості синхронізації цих паралельних процесів враховані на відповідній діаграмі станів за допомогою синхронізуючого стану.

Іноді розробку діаграми станів, особливо в умовах дефіциту часу, опускають, оскільки часто відбувається дублювання інформації, представленої на діаграмах послідовності та комунікації (кооперації).

2.3.8 Розділ 3 Розробка та тестування програмного забезпечення

Передбачається вибір програмного середовища, розробка алгоритмів, та кодування. Обов’язково необхідно відобразити процес розробки бази даних та проектування інтерфейсу користувача

Процес розробки програмного забезпечення необхідно будувати з використанням технологій гнучкого програмування, хмарних технологій, технологій розробки web-сервісів.

2.3.8.1 Розробка програмного комплексу

2.3.8.1.1 Обґрунтування вибору засобів реалізації

Вибір програмних та апаратних засобів для розв’язання поставлених задач проводиться на підставі постановки задачі, розроблених структури та алгоритму функціонування розроблюваного об’єкту. Необхідно навести аргументи на користь вибору середовища розробки, мови (мов) програмування, СКБД та інших необхідних засобів, а також конфігурації апаратних засобів. Вибір засобів розробки повинен бути спрямованим на оптимізацію процесу розробки (можуть розглядатися часові, вартісні, функціональні та інші параметри).

2.3.8.1.2 Опис структурної (функціональної) схеми

Загальний структурний опис системи (програми) має відображати основні структурні компоненти та зв’язки між ними. Переважно відображається структура за функціональними ознаками. При необхідності наводяться структурні схеми всієї системи та її складових. Опис подається у текстовому та графічному вигляді.

Результат уточнення структури може бути представлений у вигляді структурної та функціональної схем, які дають досить повне уявлення про проектоване програмне забезпечення.

Структурна схема програмної системи визначає основні функціональні частини системи, їх взаємозв’язки та призначення. Під функціональною частиною розуміють складову частину схеми (елемент): підсистему, функціональну групу або інші структурні компоненти.

Структурна схема призначена для відображення загальної структури системи, тобто її основних блоків, вузлів, частин та головних зв’язків між ними. Із

структурної схеми повинно бути зрозуміло, навіщо потрібний даний елемент і як він працює в основних режимах роботи, як взаємодіють його частини. Позначення елементів структурної схеми можуть обиратись довільно, хоча загальноприйнятих правил виконання схем слід дотримуватись.

Більш повне уявлення про проектоване програмне забезпечення з точки зору взаємодії його компонентів між собою і з зовнішнім середовищем дає функціональна схема.

Функціональна схема – це схема взаємодії компонентів программного забезпечення з описом інформаційних потоків, складу даних у потоках і зазначенням використовуваних файлів і пристроїв. Для зображення функціональних схем використовують спеціальні позначення (графічні символи), встановленні стандартом (ГОСТ 19.701-90 – Единая Система Программной Документации, рисунок 2.73).

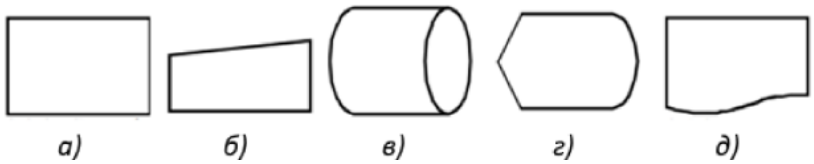


Рисунок 2.73 – Основні графічні символи для опису функціональної схеми програмного забезпечення: а) – процес, підсистема; б) – ручне введення; в) – пристрій пам’яті з прямим доступом; г) – дисплей; д) – документ

Функціональні схеми інформативніші ніж структурні. На рисунок 2.74 приведені структурна (а) та функціональна (б) схеми програмної системи.

Усі компоненти структурних і функціональних схем мають бути описані на різних рівнях деталізації, при цьому число рівнів залежить від розмірів і складності завдання обробки даних. Рівень деталізації повинен бути таким, щоб різні частини і взаємозв’язок між ними були зрозумілі в цілому.

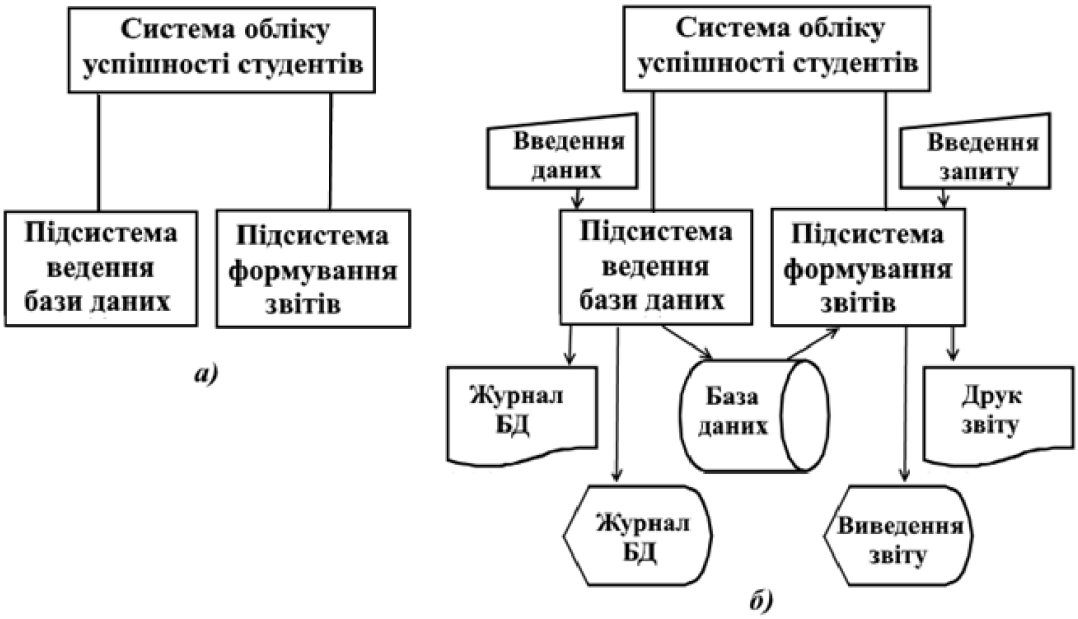


Рисунок 2.74 – Приклад схем програмної системи: а) – структурної; б) – функціональної

2.3.8.1.3 Опис логічної схеми системи

У цьому підрозділі повинна бути описана послідовність дій, що відбувається у проектованому об’єкті, яка призводить до того чи іншого результату. Звичайною формою опису логічної схеми є опис алгоритму функціонування системи або програми, який виконується в текстовому вигляді, а потім подається його графічна інтерпретація у вигляді блок-схеми алгоритму (або у вигляді діаграм діяльності UML [6]).

Якщо об’єкт має складну структуру, рекомендується подавати узагальнену блок-схему алгоритму всієї системи та блок-схеми алгоритмів її компонентів. Ступінь деталізації опису визначається за погодженням з керівником роботи, але представлена логічна структура повинна однозначно описувати основні інформаційні процеси, які відбуваються у проектованому об’єкті, та їхній характер: операції зберігання, обробки, відображення та передачі даних, циклічні структури, переходи за умовою тощо.

2.3.8.1.4 Розробка бази даних

Невід'ємною частиною чисельної категорії інформаційних систем є бази даних (БД). База даних є єдиним і великим сховищем даних. Це сховище

визначається один раз, а потім багаторазово (і спільно) використовується різними користувачами або функціональними частинами системи.

Замість окремих файлів з надлишковими даними всі дані в БД зібрані разом, з мінімальною часткою надмірності. Зазвичай БД належить не окремому клієнту, а розглядається як загальний ресурс, що розділяється.

Збережені в БД дані мають певну логічну структуру, описувану підтримуваною моделлю даних. У даний час найпоширенішою моделлю даних є реляційна модель. Перевагами реляційної моделі є простота, гнучкість, зрозумілість, зручність реалізації. Однак при збільшенні числа таблиць помітно знижується швидкість роботи з БД.

При моделюванні БД послідовно застосовують три типи моделей:

1 Концептуальні моделі даних. Ці моделі створюють на першому кроці моделювання і використовують для розробки понять проблемної області з точки зору замовника. Основними елементами тут є бізнес-моделі.

2 Логічні моделі даних. Логічні моделі створюють на другому кроці моделювання. Вони фіксують вимоги до системи з точки зору розробника і описують логічну організацію системи з базою даних, що реалізує ці вимоги (в термінах сутностей даних і відносин між сутностями). Основними елементами логічних моделей є діаграми класів.

3 Фізичні моделі даних. Фізичні моделі створюють на третьому кроці моделювання. З їх допомогою проектують внутрішню схему бази даних, зображуючи таблиці даних, атрибути (стовпці) таблиць і відносини між таблицями, якщо потрібно, то і додаткові індекси.

Мається на увазі, що атрибут сутності з логічної моделі автоматично перетворюється у відповідний атрибут (стовпець) таблиці фізичної моделі. Стовпці таблиці розділяють на ключові або неключові. У свою чергу, ключовий стовпець може бути первинним ключем, зовнішнім ключем або ж комбінацією первинного та зовнішнього ключа.

Для побудови схеми бази даних можна використовувати як розширення UML для моделювання баз даних [6, розділ 12], так і будь-які автоматизовані системи для побудови концептуальної, логічної або фізичної моделі БД.

2.3.8.1.5 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача є важливою стадією конструювання. Інтерфейс забезпечує взаємодію користувача та програмно-технічного забезпечення ІС.

Ефективність інтерфейсу визначається здатністю проектувальника передбачити вимоги користувачів, які виникнуть на першому та наступному етапах використання інформаційної системи. Необхідно передбачити зростання (зміну) вимог до програмно-технічного комплексу в майбутньому та забезпечити здатність ІС їх виконання без доопрацювання системи.

Процес проектування інтерфейсу користувача в роботі повинен бути відображений шляхом висвітлення результатів виконання вимог, запропонованих відомим проектувальником інтерфейсів Якобом Нільсеном [7], та описаних у роботі [6, розділ 13].

2.3.8.1.6 Опис розробки програмних компонентів

У даному підрозділі необхідно представити опис розроблених модулів або інших структурних компонентів, а також навести відомості про призначення модуля, зв’язки з іншими компонентами розробки, особливості реалізації алгоритму. Опис супроводжується фрагментами тексту модуля з необхідними коментарями. Опис розробки програмних компонентів повинен містити достатні відомості щодо використаних методів, технологій, стандартних та запозичених компонентів тощо.

2.3.8.2 Тестування системи

Результатом цього етапу (5-10 сторінок) є підтвердження коректності виконання своїх завдань інформаційною системою в цілому та її програмним забезпеченням зокрема. Тестування використовує критерій коректності виконавчого додатку для ідентифікації дефектів або для того, щоб показати

мінімальний рівень прийнятності. Даний процес складається, як мінімум, з модульного, інтеграційного або системного тестування [6, розділ 14].

Тестування розробленої системи, програми або програмного комплексу у загальному випадку проводиться за схемою, етапи якої передбачають:

- визначення об’єкту тестування;
- визначення мети тестування;
- вибір технології тестування;
- розробка програми тестування;
- проведення тестування за розробленою програмою;
- підведення висновків з тестування.

Повинні також бути розроблені та застосовані:

- тести для класів, які базуються на реалізації поведінки;
- тести для класів і малих кластерів класів, які часто утворюють шаблони, відповідальні за реалізацію конкретних зовнішніх або внутрішніх функцій;
- тести для об’єктно-орієнтованої системи загалом, що базується на її перевірці в робочому режимі.

Тестування розробленої системи зазвичай проводять на таких основних рівнях тестування.

2.3.8.2.1 Модульне тестування

Модульне тестування – це процес перевірки окремих програмних процедур (модулів) і підпрограм, що входять до складу програм. Модульне тестування проводиться безпосереднім розробником і дозволяє перевіряти всі внутрішні структури і потоки даних у кожному модулі. Цей вид тестування є частиною етапу розробки.

2.3.8.2.2 Інтеграційне тестування

Інтеграційне тестування проводиться для перевірки спільної роботи окремих модулів і передує тестуванню всієї системи як єдиного цілого. У ході інтеграційного тестування перевіряються зв'язки між модулями, їх сумісність і функціональність. Воно здійснюється незалежним тестувальником і входить до складу етапу тестування.

2.3.8.2.3 Системне тестування

Системне тестування призначене для перевірки програмної системи в цілому, її організації та функціонування на відповідність специфікаціям вимог замовника. Його проводить незалежний тестувальник після успішного завершення інтеграційного тестування.

2.3.8.2.4 Приймальне тестування

Приймальне тестування проводиться організацією, що відповідає за інсталяцію, супровід програмної системи та навчання кінцевого користувача.

Особливості цього етапу кваліфікаційної роботи залежать від специфіки теми та завдання на виконання кваліфікаційної роботи.

У цьому ж розділі (кінцевий підрозділ – «4.х Реалізація та впровадження результатів») подають приклади впровадження створеної системи. Подаються зображення інтерфейсу, вигляд вихідних даних, характеристики процесів, які використовують спроектований програмний продукт.

Даний розділ може включати такі етапи як впровадження (або рекомендації до впровадження) і вдосконалення системи.

Розділ тексту має закінчуватися підрозділом – Висновки до розділу (без нумерації підрозділу) не більш 1-2 сторінки. Розмір одного висновку приблизно – один абзац (3-7 рядків).

2.3.8.3 Приклади впровадженого програмного комплексу.

Скріншоти інтерфейсу при виконанні послідовності кроків програми. Інструкція користувача.

Гнучке програмування. Більшість гнучких методик націлені на мінімізацію ризиків, шляхом зведення розробки до серії коротких циклів. Існує немало методів найбільш поширені Agile-методи та Scrum-методи Kanban, Extreme Programming та інших.. Розробки впровадження та адаптації процесів гнучкої методології розробки до специфічних завдань, стадії розробки проекту, ADAPTація, моделі впровадження Scrum, спринт, моделі MSF, методика Test-Driven-Development, а також моделювання з використанням UML [9, 10, 11, 12].

Хмарні технології. Методика використання хмарних технологій та підходи, cloud computing, таксономія хмар, розподіл ролей хмарної сфери діяльності, складові хмарного бізнесу, використовувані в Google Apps, Open Class, Piazza [13, 14, 15].

Розробка WEB сервісів. При розробці дотримуються такої послідовності:

- аналіз конкурентів (що є на ринку і як зробити краще);
- проектування WEB-сервісу (прототип + специфікація на розробку);
- R&D (розробки і попередня розробка WEB-сервісів, проектування БД);
- дизайн (концепт дизайну і дизайн усіх сторінок);
- FRONT-END експертиза верстка (HTML-верстка дизайну і висновок даних на фронт);
- BACK-END розробка (проектування БД, зв'язків, робота з даними, серверна частина розробки) або розробка WEB-сервісу (робота з даними, серверна частина розробки) API і документація (розробка та тестування API інтернет-сервісу);
- інтеграція (реалізуємо інтеграцію з внутрішніми системами і з CRM);
- тестування (і обов'язкове покриття тест-кейсами);
- запуск WEB-сервісу (настройка серверів, перенесення в реальний час).

Серверне WEB-програмування. технології побудови інтернет-додатків; особливості створення клієнтської і серверної частини (HyperText Markup Language, Cascading Style Sheets, Document Object Model, JavaScript, JQuery, AJAX); особливості проектування інтернет-додатків, Технологія MVC, Технології об'єктно-реляційних відображень (ORM) [16, 17, 18].

Основи інтеграції інформаційних потоків. Розглядають питання інтеграції електронних документальних інформаційних потоків, породжуваних в мережі Інтернет. Опираючись на математичні моделі, застосування теорії інформаційного пошуку та концепції глибинного аналізу тексту (Text Mining) до інформаційних потоків, які служать основою технології їх інтеграції, побудови сучасного інформаційного сервісу. Велика увага приділяється напрямку обробки текстової

інформації - Text Mining, яке включило в себе технологічні і методологічні підходи контент-аналізу, комп'ютерної лінгвістики, штучного інтелекту [19].

Студент повинен давати оцінку повноти вирішення поставлених задач, оцінку достовірності одержаних результатів (характеристик, параметрів), їх порівняння з аналогічними результатами вітчизняних і зарубіжних праць, обґрунтування потреби додаткових досліджень, негативні результати, які обумовлюють необхідність припинення подальших досліджень. *Обсяг третього розділу має бути не більше 40% обсягу КРБ (тобто 20-30 сторінок).*

2.3.9 Висновки

У розділі «Висновки» викладають найважливіші наукові та практичні результати, одержані в КРБ, які повинні містити формулювання розв'язаної наукової задачі, її значення для науки і практики. Далі формулюють рекомендації щодо наукового та практичного використання здобутих результатів.

У першому абзаці коротко оцінюють стан питання. Далі у висновках розкривають методи вирішення поставленої в КРБ наукової задачі, їх практичний аналіз, порівняння з відомими розв'язаннями.

Необхідно наголосити на якісних і кількісних показниках здобутих результатів, обґрунтувати достовірність результатів, викласти рекомендації щодо їх використання. *Обсяг висновків має бути не більше 2-3.*

2.3.10 Список використаних джерел

Розділ «Список використаних джерел» містить перелік джерел, що використовувались бакалавром при виконанні КРБ, оформлених відповідно до вимог нормативних документів. При написанні КРБ здобувач повинен обов'язково посилатися на авторів і джерела, з яких запозичив матеріали або окремі результати. У разі використання запозиченого матеріалу без посилання на автора та джерело КРБ знімається з розгляду незалежно від стадії проходження. Обсяг використаних джерел має бути не менше 40 посилань.

Список використаних джерел слід формувати у порядку появи посилань у тексті.

2.3.11 Додатки

До додатків за необхідності доцільно включати допоміжний матеріал:

− проміжні математичні доведення, формули та розрахунки;

− таблиці допоміжних цифрових даних;

− протоколи й акти випробувань, впровадження, розрахунки економічного ефекту;

− інструкції та методики, опис алгоритмів і програм вирішення задач на сучасному комп'ютері, розроблених у КРБ;

− допоміжні ілюстрації.

Додатки оформлюють як продовження КРБ на наступних її сторінках або у вигляді окремої частини (книги), розміщуючи їх у порядку появи посилань у тексті КРБ.

Якщо додатки оформлюють на наступних сторінках КРБ, кожен із таких додатків повинен починатися з нової сторінки. Додаток повинен мати заголовок, надрукований угорі малими літерами з першої великої симетрично відносно тексту сторінки. Посередині рядка над заголовком малими літерами з першої великої друкується слово «Додаток» і велика літера, що позначає додаток.

Додатки слід позначати послідовно великими літерами української абетки, за винятком літер Ґ , Є, І, І, Й, О, Ч, Ь. наприклад, Додаток А, Додаток Б. Один додаток позначається як Додаток А.

При оформленні додатків окремою частиною (книгою) на титульному аркуші під назвою КРБ друкують великими літерами слово «ДОДАТКИ»

Текст кожного додатка за необхідності може бути поділений на розділи й підрозділи, які нумерують у межах кожного додатка. Перед кожним номером ставлять позначення додатка (літеру) і крапку, наприклад, А.2 – другий розділ додатка А; В.3.1 - перший підрозділ третього розділу додатка В.

Ілюстрації, таблиці та формули, розміщені в додатках, нумерують у межах кожного додатка, наприклад: рисунок Д.1.2 - другий рисунок першого розділу додатка Д). Формула (А.1) — перша формула додатка А.

3. ОФОРМЛЕННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

3.1 Загальні вимоги

Текст пояснювальної записки КРБ оформлюється відповідно до вимог, викладених в [1].

Заголовки структурних частин «ЗМІСТ», «ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ», «ВСТУП», «РОЗДІЛ», «ВИСНОВКИ», «СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ», «ДОДАТКИ» друкують великими літерами симетрично до набору. Заголовки підрозділів друкують маленькими літерами (крім першої великої) з абзацного відступу. Крапку в кінці заголовка не ставлять [20].

Якщо заголовок складається з двох або більше речень, їх розділяють крапкою.

Заголовки пунктів друкують маленькими літерами (крім першої великої) з абзацного відступу в розрядці у підбір до тексту. В кінці заголовка, надрукованого в підбір до тексту, ставиться крапка.

Відстань між заголовком (за винятком заголовка пункту) та текстом повинна дорівнювати трьом інтервалам.

Кожну структурну частину КРБ треба починати з нової сторінки.

3.2 Нумерація

Нумерацію сторінок, розділів, підрозділів, пунктів, підпунктів, рисунків (малюнків), таблиць, формул подають арабськими цифрами без знака № [20].

Першою сторінкою КРБ є титульний аркуш, який включають до загальної нумерації сторінок. На титульному аркуші номер сторінки не ставлять, на наступних сторінках номер проставляють у правому нижньому куті сторінки без крапки в кінці.

Такі структурні частини КРБ , як зміст, перелік умовних позначень, вступ, висновки, список використаних джерел не мають порядкового номера. Всі аркуші, на яких розміщені згадані структурні частини КРБ, нумерують звичайним чином. Не нумерують лише їх заголовки, тобто не можна друкувати: «1. ВСТУП» або «б.

ВИСНОВКИ». Номер розділу ставлять після слова «РОЗДІЛ», після номера крапку не ставлять, потім з нового рядка друкують заголовок розділу.

Підрозділи нумерують у межах кожного розділу. Номер підрозділу складається з номера розділу і порядкового номера підрозділу, між якими ставлять крапку. В кінці номера підрозділу повинна стояти крапка, наприклад: «2.3.» (третій підрозділ другого розділу). Потім у тому ж рядку наводять заголовок підрозділу.

Пункти нумерують у межах кожного підрозділу. Номер пункту складається з порядкових номерів розділу, підрозділу, пункту, між якими ставлять крапки. В кінці номера повинна стояти крапка, наприклад: «1.3.2.» (другий пункт третього підрозділу першого розділу). Потім у тому ж рядку наводять заголовок пункту.

Пункт може не мати заголовка.

Підпункти нумерують у межах кожного пункту за такими ж правилами, як пункти.

3.3 Ілюстрації та таблиці

Ілюстрації (фотографії, креслення, схеми, графіки, карти) і таблиці необхідно подавати безпосередньо після тексту, де вони згадані вперше, або на наступній сторінці. Ілюстрації і таблиці, розміщені на окремих сторінках КРБ, включають до загальної нумерації сторінок. Таблицю, рисунок або креслення, розміри якого більше формату А4, враховують як одну сторінку і розміщують після згадування у тексті [20].

Ілюструють КРБ за продуманим тематичним планом, що допомагає запобігти невиправданим пропускам ілюстрацій до найважливіших тем. Кожна ілюстрація має відповідати тексту, а текст - ілюстрації.

Ілюстрації позначають словом «Рисунок» і нумерують послідовно в межах розділу, за винятком ілюстрацій, поданих у додатках.

Номер ілюстрації повинен складатися з номера розділу і порядкового номера ілюстрації, між якими ставиться крапка. Наприклад: Рисунок 1.2 (другий рисунок першого розділу). Номер ілюстрації, її назву і пояснювальні підписи розміщують послідовно під ілюстрацією. Якщо в розділі КРБ подано одну

ілюстрацію, то її нумерують за загальними правилами. Основними видами ілюстративного матеріалу в КРБ є: технічний рисунок, блок-схема алгоритму, фотографія, діаграма, графік.

Підпис під ілюстрацією зазвичай має чотири основні елементи:

− найменування графічного сюжету, що позначається скороченим словом «Рисунок»;

− порядковий номер ілюстрації, який вказується без знаку номера арабськими цифрами;

− тематичний заголовок ілюстрації із стислою характеристикою зображеного;

− експлікацію, яка будується так: деталі рисунку позначають цифрами, які виносять у підпис і супроводжують текстом. Експлікація не замінює загального найменування сюжету, а лише пояснює його.

Приклад:

Рисунок 1.11 - Структурна схема системи:

1 – загальний вид;

2 – база даних;

3 – модель об’єкта.

Не варто оформлювати посилання на ілюстрації як самостійні фрази, в яких лише повторюється те, що міститься у підпису. В тому місці, де викладається тема розміщують посилання у вигляді виразу в круглих дужках «(рисунок 3.1)» або зворот типу: «…як це видно з рисунка 3.1» або «…як це показано на рисунку 3.1»

Таблиці нумерують послідовно (за винятком таблиць, поданих у додатках) и межах розділу. В правому верхньому куті над відповідним заголовком таблиці розміщують напис « Таблиця» із зазначенням її номера. Номер таблиці повинен складатися з номера розділу і порядкового номера таблиці, між якими ставиться крапка, наприклад: «Таблиця 1.2» (друга таблиця першого розділу).

Якщо в розділі одна таблиця, її нумерують за загальними правилами.

Таблиця (номер)

Назва таблиці

Головка						Заголовок граф
						Підзаголовки граф
Боковик (заголовки рядків)						
	Графи (колонки)					

Кожна таблиця повинна мати назву, яку розміщують над таблицею симетрично до тексту. Назву і слово «Таблиця» починають з великої літери.

Назву наводять жирним шрифтом.

За логікою побудови таблиці її логічний суб'єкт, або підмет (позначення тих предметів, які в ній характеризуються), розміщують у боковику, головці, чи в них обох. Кожен заголовок над графою стосується всіх даних цієї графи, кожен заголовок рядка в боковику - всіх даних цього рядка.

Заголовок графи має бути по можливості коротким. Слід уникати повторів тематичного заголовка в заголовках граф, одиниці виміру зазначати у заголовку, виносити до узагальнюючих заголовків слова, що повторюються.

Однорідні числові дані розміщують так, щоб їх класи збігалися; неоднорідні - посередині графи; лапки використовують тільки замість однакових слів, які стоять одне під одним.

Заголовки граф повинні починатися з великих літер, підзаголовки – з маленьких, якщо вони складають одне речення із заголовком, і з великих, якщо вони є самостійними. Висота рядків повинна бути не меншою 8 мм. Графу з порядковими номерами рядків до таблиці включати не треба.

Таблицю розміщують після першого згадування про неї в тексті (наприклад: «у таблиці 1.2.»), так, щоб її можна було читати без повороту КРБ або з поворотом за стрілкою годинника. Таблицю з великою кількістю рядків можна переносити на наступну сторінку. При перенесенні таблиці на наступну сторінку назву вміщують тільки над її першою частиною. Таблицю з великою кількістю граф можна ділити на частини і розміщувати одну частину під іншою в межах однієї сторінки. Якщо рядки або графи таблиці виходять за формат сторінки, то в першому випадку в кожній частині таблиці повторюють її головку, в другому – боковик.

При перенесенні частини таблиці на іншу сторінку слово «Таблиця» і номер її вказують один раз справа над першою частиною таблиці, над іншими частинами пишуть слова «Продовж, табл.» і вказують номер таблиці, наприклад: «Продовж, таблиці 1.2».)

Примітки до тексту і таблиць, в яких наводять довідкові і пояснювальні дані, нумерують послідовно в межах однієї сторінки. Якщо приміток на одному аркуші кілька, то після слова «Примітки» ставлять двокрапку, наприклад:

Примітки:

1....

2....

Якщо є одна примітка, то її не нумерують і після слова «Примітка» ставлять крапку.

3.4 Формули

Формули в КРБ (якщо їх більше однієї) нумерують у межах розділу. Номер формули складається з номера розділу і порядкового номера формули в розділі, між якими ставлять крапку. Пояснення значень символів і числових коефіцієнтів треба подавати безпосередньо під формулою в тій послідовності, в якій вони наведені у формулі. Значення кожного символу і числового коефіцієнта треба подавати з нового рядка. Перший рядок пояснення починають зі слова «де» без двокрапки [20].

Номери формул позначають арабськими цифрами в круглих дужках біля правого поля сторінки без крапок від формули до її номера на рівні відповідної формули, наприклад: (3.1) (перша формула третього розділу). Номер, який не вміщується у рядку з формулою, переносять у наступний рядок нижче формули Номер формули при її перенесенні вміщують на рівні останнього рядка. Якщо формулу взято в рамку, то номер такої формули записують зовні рамки з правого боку навпроти основного рядка формули. Номер формули дробу подають на рівні основної горизонтальної риски формули [20].

Номер групи формул, розміщених на окремих рядках і об'єднаних фігурною дужкою (парантезом), ставиться справа від вістря парантеза, яке знаходиться в середині групи формул і спрямовано в сторону номера [20].

3.5 Блок-схеми

Блок-схеми є одним з найбільш зручних способів запису алгоритмів і являють собою різновид рисунку. Алгоритм при цьому зображується у вигляді послідовних блоків, які наказують виконати окремі функції, та зв'язків між ними. В середині блоків вміщується інформація, що пояснює дії, які блоки виконують. Хід обчислювального процесу відображується лініями зв'язку між блоками схеми. Напрямки ліній зв'язку обов'язково позначаються стрілками у тому випадку, якщо вони направлені справа наліво або знизу догори. У інших випадках стрілки можуть бути опущені [20].

3.6 Цитування та посилання на використані джерела

При написанні КРБ студент повинен посилатися на джерела, матеріали або окремі результати з яких наводяться в КРБ, або на ідеях і висновках яких розроблюються задачі, питання у КРБ. Посилатися слід на останні видання публікацій. На більш ранні видання можна посилатися лише в тих випадках, коли наявний у них матеріал не включений до останнього видання.

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

скорочення наведеного витягу може спотворити зміст. Загальні вимоги до цитування такі [20]:

– 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 має 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 КРБ до окремих слів або думок з цитованого тексту, то після них у круглих дужках ставлять знак оклику або знак питання;

– коли автор КРБ, 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 КРБ, а весь текст застереження вміщується у круглі дужки. Варіантами таких застережень є: (курсив наш. - *М.Х.),* (підкреслено мною. - *М.Х.),* (розбивка моя. - *М.Х.)*

3.7 Захист кваліфікаційної роботи бакалавра

Захист КРБ та оформлення відповідних документів регулюються положенням про екзаменаційну комісію, та нормативними документами Університету.

Захист КРБ проводиться на відкритому засіданні Екзаменаційної комісії (ЕК) за участю не менше як половини її складу з обов'язковою присутністю голови комісії.

Процедура захисту передбачає [21]:

- презентація (доповідь) студентом змісту та результатів своєї роботи;
- відповіді студента на запитання членів ЕК та осіб, присутніх на захисті;
- оголошення відгуку наукового керівника та рецензента;
- заключне слово студента (за бажанням);
- оголошення рішення комісії про оцінку роботи.

Доповіді студентів необхідно підготувати заздалегідь у формі виступу, в якому доцільно висвітлити такі важливі питання: обґрунтування актуальності теми розробки; мета, завдання, об'єкт, предмет розробки; що вдалося встановити, виявити, довести; якими методами це досягнуто; елементи новизни у теоретичних положеннях та в практичних рекомендаціях; з якими труднощами довелося зіткнутися в процесі розробки, які положення не знайшли підтвердження.

У виступі мають міститися також відповіді на основні зауваження наукового керівника та рецензента. Доповідь студента на захисті КРБ – 7-10 хвилин.

Завершуючи доповідь, випускник має відзначити: які його розробки та висновки впроваджені або намічені до впровадження; де ще можна, на його думку, застосувати результати розробки; яка фактична чи очікувана соціально-економічна ефективність запропонованих ним заходів.

Під час доповіді слід звертатися до презентаційного матеріалу, коротко пояснюючи його зміст. Захист КРБ фіксується у протоколі ЕК.

Студент готує до захисту матеріал у вигляді презентації за допомогою програмного продукту Microsoft Office PowerPoint, аналогічних програмних продуктів та роздаткового матеріалу, який містить таблиці, графіки, діаграми, схеми тощо, на які посилається студент у своїй доповіді, а також основні висновки та пропозиції, сформульовані в результаті розробки. Необхідну кількість та зміст ілюстрацій студент визначає самостійно, але погоджує з

науковим керівником. Презентація може складатися з 10-14 слайдів. Доцільно на слайдах розміщувати до 12-14 рядків розміром шрифту не менше ніж 20 пт.

Основні можливі назви слайдів або листів графічного матеріалу КРБ для захисту можуть бути такими (приклад):

- тема, студент, керівник роботи;
- актуальність роботи, мета, завдання, об'єкт, предмет розробки;
- вимоги до розроблювальної програми (системи);
- аналіз існуючих аналогів;
- вибір засобів розробки;
- проект розроблювальної програми (системи);
- реалізація розроблювальної програми (системи);
- тестування розроблювальної програми (системи);
- висновки.

Роздатковий матеріал оформлюється на окремих аркушах формату А4. На титульній сторінці необхідно вказати тему КРБ, її виконавця та керівника.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 «Про внесення змін до методичних рекомендацій щодо розроблення стандартів вищої освіти» (у редакції наказ Міністерства освіти і науки України від 01.10.2019 №1254.

2 Гломозда Д.К. Координація в асинхронних обчислювальних мережах: автореф. дис. на здобуття наук. ступеня канд. тех. наук : спец. 01.05.03 "Математичне та програмне забезпечення обчислювальних машин і систем" / Д. К. Гломозда. – Київ, 2011. – 20 с.

3 Рибчак З.Л. Методи та засоби підтримки прийняття рішень формування та розвитку територіальних громад: автореф. дис. на здобуття наук. ступеня канд. тех. наук: спец. 01.05.03 "Математичне та програмне забезпечення обчислювальних машин і систем" / С.В. Титенко. — Львів, 2019. — 23 с.

4 Авраменко В.С., Голуб С.В., Салапатов В.І. Дипломне проектування. Освітній ступінь «бакалавр» : Навчально-методичний посібник / В. С. Авраменко, С.В. Голуб, В.І. Салапатов. – Черкаси: Черкаський національний університет ім. Б. Хмельницького, 2018. – 213 с.: 76 іл.

5 Rosenberg D., Scott K. Application of object modeling using UML and case analysis / D. Rosenberg, K. Scott. - Per. from English. – M.: DMK Press, 2002. – 160 p. Publisher: Addison Wesley First Edition June 14, 2001 ISBN: 0-201-73039-1, 176 pages

6 Авраменко В. С. Авраменко С. А. Проектування інформаційних систем / В. С. Авраменко, А. С. Авраменко. – Черкаси: Чабаненко Ю. А, 2017. – 434 с.

7 Нильсен Я. Веб-дизайн. Книга Якоби Нильсена / Якоб Нильсен. – К.: Символ-Плюс, 2006. – 512 с.

8 Leonenkov A. V. UML 2 ISBN-10 : 5941578784

9 Extreme Programming: Statement of the Process. From the first steps to the victorious end. K. Auer, R. Miller. Addison-Wesley Professional; 1st edition (October 1, 2001) Language: English Paperback: 376 pages ISBN-10: 0201616408 ISBN-13: 978-0201616408

10 Schwaber, Ken, and Mike Beedle. "Agile Software Development with Scrum." (2001).

11 Kent Beck Extreme programming: development through testing. Extreme Programming Explained Publisher: First Edition September 29, 1999 ISBN: 0201616416, 224 pages

12 Ambler S. Agile Modeling. Effective Practices for Extreme Programming and the Unified Process. Wiley & Sons, New York, 2002.

13 Облачные технологии и образование: под общ. ред. З.С. Сейдаметовой. – Симферополь: «ДИАЙПИ», 2012. – 204 с.

14 Хмарні технології [Електронний ресурс]. – Точка доступу: URL: http://j.parus.ua/ua/358/ – Хмарні технології.

15 Google Cloud Computing [Електронний ресурс]. – Точка доступу: URL: https://cloud.google.com – How Google Cloud Platform works.

16 Neil A. B. Gray Server Web Programming: Wiley; 1st edition (June 2, 2003) English 576 pages ISBN-10 0470850973

17 Vincent W. Django for Beginners Learn Web Development With Django 2.0, 2018 – 324p.

18 Eve Astrid Andersson, Philip Greenspun, Andrew Grumet Software Engineering for Internet Applications

19 Ландэ Д.В. Основы интеграции информационных потоков: Монография. – К.: Инжиниринг, 2006. – 240 с.

20 Методичні вказівки до оформлення та виконання магістерської випускної роботи для студентів спеціальності 8.080403 "Програмне забезпечення автоматизованих систем" освітньо-кваліфікаційного рівня "магістр" [Текст] /Укл.: А.А. Рідкокаша, О.Б. Півень ; М-во освіти і науки України, Черкас, держ. технол. ун-т. - Черкаси : ЧДТУ, 2010. - 27 с.

21 О.Б. Данченко, О.С. Шарова Методичні вказівки до виконання магістерської кваліфікаційної роботи. - К.: Університет економіки та права "КРОК", 2017. – 30 с.

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій і систем

Кафедра програмного забезпечення автоматизованих систем

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра

на тему: «Синтез предикторів в програмних системах із функціями прогнозування»

Виконав: студент 4 курсу, групи ПЗ-1804

спеціальності

121 «Інженерія програмного забезпечення»

(шифр і назва напряму підготовки)

Студент Луговий І.О.

(прізвище та ініціали)

Керівник Голуб С.В.

(прізвище та ініціали)

Рецензент Шубніков К.В.

(прізвище та ініціали)

Черкаси 2023

ДОДАТОК Б

Черкаський державний технологічний університет

повне найменування вищого навчального закладу

Факультет інформаційних технологій і систем

Кафедра програмного забезпечення автоматизованих систем

Освітній рівень бакалавр

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри ПЗАС, професор

_____ С. Голуб

«___» _____ 2022 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Прокопенко Тимофій Борисович

(прізвище, ім'я, по батькові)

1. Тему проекту (роботи) Розробка програмного забезпечення для одночасного перегляду відеоконференції в Zoom та Viber на платформі Android

Керівник проекту (роботи) Первунінський Станіслав Михайлович д.т.н., професор

(прізвище, ім'я , по батькові, науковий ступінь, вчене звання)

Затверджені наказом Черкаського державного технологічного університету від « 10 » жовтня 2022 року №333/01

2. Строк подання студентом проекту (роботи) _____

3. Вхідні дані до проекту (роботи) стандарти програмного забезпечення; процеси управління; вимоги до проекту; календарне планування проекту; управління ризиками проекту; управління ресурсами

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ;

Теоретичний аналіз поняття відеоконференції і особливості її застосування;

Розробка концепція проекту;

Управління проектом розробки відеоконференції для платформи Android

Оцінювання ефективності;

Висновки;

Список використаних джерел;

Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових робіт проекту;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посади консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1			
2			
3			

7. Дата видачі завдання 02 грудня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної роботи	Строк виконання етапів кваліфікаційної роботи	Примітки
1	Постановка задачі	05.12.2022	виконано
2	Підготовка завдання	13.12.2022	виконано
3	Погодження завдання	16.12.2022	виконано
4	Затвердження завдання	19.02.2022	виконано
	Основна стадія		
1	Підбір матеріалів	27.02.2022	виконано
2	Аналіз шляхів вирішення поставленої задачі	04.02.2022	виконано
3	Розрахунок основних параметрів роботи	10.03.2022	виконано
4	Вибір кінцевого варіанту проектного рішення	17.03.2022	виконано
5	Оформлення первісної редакції роботи	25.03.2022	виконано
	Заключна стадія		

1	Узгодження прийнятих проектних рішень з керівником	31.04.2022	виконано
2	Оформлення пояснювальної записки роботи в кінцевій редакції	13.05.2022	виконано
3	Попередній захист роботи	15.05.2022	виконано
4	Затвердження роботи	02.06.2022	
5	Рецензування роботи	06.06.2022	
6	Захист роботи	11.06.2022	

Студент _____ Прокопенко Т.Б.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Первунінський С.М.

(підпис) (прізвище та ініціали)

ДОДАТОК В

Відгук

на кваліфікаційну роботу бакалавра

Студент (ки):_____

факультету _____

кафедри _____групи _____

на тему: _____

_____

Спеціальності _____

(шифр і назва спеціальності)

Освітньої програми _____

Обсяг роботи: _____

кількість сторінок пояснювальної записки (основної частини) _____

кількість листів презентації _____

Короткий зміст кваліфікаційної роботи та прийнятих рішень: _____

_____

_____

_____

_____

_____

головні цілі роботи_____

_____

_____

_____

відповідність виконаного завдання кваліфікаційної роботи_____

_____

_____

_____

_____

ступінь самостійності при виконанні роботи _____

(характеристика виконання кожного

_____

розділу роботи, ступінь використання останніх досягнень науки і передових методів роботи

_____

_____

_____

_____

_____

рівень підготовленості студента до прийняття сучасних рішень _____

_____

умінь аналізувати необхідні літературні джерела, приймати правильні інженерні рішення,

_____

застосовувати сучасні системні та інформаційні технології, проводити моделювання, обробляти та

_____

аналізувати результати експерименту найбільш важливі теоретичні і практичні результати апробації їх

_____

(участь у конференціях, семінарах, оформлення патентів, найбільш важливі теоретичні і

_____

практичні результати, публікацій в наукових журналах)

_____

_____

_____

_____

висновок про відповідність кваліфікаційної роботи завданню: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Недоліки роботи: _____

_____

_____

_____

_____

_____

_____

Визнати, що здобувач вищої освіти _____

(прізвище та ініціали)

виконав(ла) кваліфікаційну роботу бакалавра з оцінкою:

за національною шкалою _____; ECTS ____ (____)

Присвоїти _____

(прізвище, ім’я, по батькові)

ступінь вищої освіти _____

кваліфікацію _____

за спеціальністю _____

освітня програма _____

Керівник кваліфікаційної роботи

_____ _____ _____

(посада, вчені звання, ступінь) (підпис) (ініціали, прізвище)

ДОДАТОК Г

Рецензія

на кваліфікаційну роботу бакалавра

Студент (ки): _____

факультету _____

кафедри _____групи _____

на тему: _____

_____

Спеціальності _____

(шифр і назва спеціальності)

Освітньої програми _____

Обсяг кваліфікаційної роботи: _____

кількість сторінок пояснювальної записки (основної частини)_____

кількість листів презентації _____

Короткий зміст кваліфікаційної роботи та прийнятих рішень:_____

_____

_____

_____

_____

головні цілі роботи_____

_____

_____

_____

відповідність виконаного завдання кваліфікаційної роботи _____

_____

_____

ступінь самостійності при виконанні роботи _____

(характеристика виконання кожного

_____

розділу роботи, ступінь використання останніх досягнень науки і передових методів роботи

_____

_____

_____

_____

актуальність теми проекту _____

_____

умінь аналізувати необхідні літературні джерела, приймати правильні інженерні рішення,

_____

застосовувати сучасні системні та інформаційні технології, проводити фізичне або математичне моделювання,

_____

обробляти та аналізувати результати експерименту найбільш важливі теоретичні і практичні результати

_____

апробації їх (участь у конференціях, семінарах, оформлення патентів, найбільш важливі теоретичні і практичні

_____

журналах)

_____

висновок про відповідність роботи завданню: _____

_____

характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки

_____

і передових методів роботи

_____

_____

_____

_____

_____

_____

_____

Негативні сторони:_____

_____

_____

_____

_____

_____

_____

Висновок про міру інженерної підготовки студента _____

_____

_____

_____

Загальний висновок _____

_____

_____

_____

Визнати, що здобувач вищої освіти _____

(прізвище та ініціали)

виконав(ла) кваліфікаційну роботу з оцінкою:

за національною шкалою _____; ECTS ____ (____)

Присвоїти _____

(прізвище, ім’я, по батькові)

ступінь вищої освіти _____

кваліфікацію _____

за спеціальністю _____

освітня програма _____

Рецензент_____

(прізвище, ім’я по батькові, місце роботи, посада, вчений ступінь, звання)

_____

«____» _____ 202_ р.

_____

(підпис)

ДОДАТОК Д

Зразок оформлення анотації

АНОТАЦІЯ

Кваліфікаційна робота бакалавра на тему «_____» містить ____ сторінок, _____ таблиць, _____ рисунків, список використаних джерел з _____ найменувань, _____додатків.

Метою виконання кваліфікаційної роботи бакалавра є

_____

_____

Головні завдання при проектуванні інформаційної системи

_____

Об'єктом роботи є

_____

___

Предметом розробки є

_____

Вирішено наступні поставлені задачі

_____

_

КЛЮЧОВІ СЛОВА:

ДОДАТОК Е

Зразок оформлення списку використаних джерел

Книги

Один автор

Коренівський Д. Г. Дестабілізуючий ефект параметричного білого шуму в неперервних та дискретних динамічних системах / Коренівський Д. Г. — К. : Ін-т математики, 2006. — 111с. — (Математика та її застосування) (Праці / Ін-т математики НАН України ; т. 59).

Два автори

Ромовська З. В. Сімейне законодавство України / З. В. Ромовська, Ю. В. Черняк. — К. : Прецедент, 2006. — 93 с. — (Юридична бібліотека. Бібліотека адвоката) (Матеріали до складання кваліфікаційних іспитів для отримання Свідоцтва про право на заняття адвокатською діяльністю ; вип. 11).

Три автори

Акофф Р. Л. Идеализированное проектирование: как предотвратить завтрашний кризис сегодня. Создание будущего организации / Акофф Р. Л., Магидсон Д., Зддисон Г. Д. ; пер. с англ. Ф. П. Тарасенко. —Днепропетровск : Баланс Бизнес Букс, 2007. —XIЛI, 265 с.

П’ять і більше авторів

Психология менеджмента / [Власов П. К., Липницкий А. В., Лущихина И. М. и др.]; под ред. Г. С. Никифорова. — [3-е изд.]. —Х. : Гуманитар. центр, 2007. — 510 с.

Без автора

Проблеми типологічної та квантитативної лексикології : [зб.наук.праць / наук. ред. Каліущенко В. та ін.]. — Чернівці: Рута, 2007. — 310 с.

Багатотомний документ

Межгосударственные стандарты : каталог в 6 т. / [сост. Ковалева И. В., Рубцова Е. Ю. ; ред. Иванов В. Л.]. —Львов : НТЦ «Леонорм-Стандарт», 2005.— (Серия «Нормативная база предприятия»). Т. 1.—2005.—277 с.

Матеріали конференцій, з’їздів

Ризикологія в економіці та підприємництві : зб. наук. праць за матеріалами міжнар. наук.-практ. конф., 27-28 берез. 2001 р. / М-во освіти і науки України, Держ. податк. адмін. України [та ін.]. — К. : КНЕУ : Акад. ДПС України, 2001. — 452 с.

Препринти

Панасюк М. І. Про точність визначення активності твердих радіоактивних відходів гамма-методами / Панасюк М. І., Скорбун А. Д., Сплошной Б. М. — Чорнобиль : Ін-т пробл. безпеки АЕС НАН України, 2006. — 7, [1] с. — (Препринт / НАН України, Ін-т пробл. безпеки АЕС ; 06-1).

Депоновані наукові праці

Разумовский В. А. Управление маркетинговыми исследованиями в регионе / В. А. Разумовский, Д. А. Андреев. — М., 2002. — 210 с. — Деп. в ИНИОН Рос. акад. наук 15.02.02, № 139876.

Законодавчі та нормативні документи

Кримінально-процесуальний кодекс України : за станом на 1 груд. 2005 р. / Верховна Рада України. — Офіц. вид. — К. : Парлам. вид-во, 2006. — 207 с. — (Бібліотека офіційних видань).

Стандарти

Якість води. Словник термінів: ДСТУ ISO 6107-1:2004 — ДСТУ ISO 6107-9:2004. — [Чинний від 2005-04-01]. — К.: Держспоживстандарт України, 2006. — 181 с. — (Національні стандарти України).

Автореферати дисертацій

Нгуен Ші Данг. Моделювання і прогнозування макроекономічних показників в системі підтримки прийняття рішень управління державними фінансами: автореф. дис. на здобуття наук. ступеня канд. техн. наук: спец. 05.13.06 «Автоматиз. системи упр. та прогрес. інформ. технології» / Нгуен Ші Данг. — К., 2007.—20 с.

Електронні ресурси

Бібліотека і доступність інформації у сучасному світі: електронні ресурси в науці, культурі та освіті : (підсумки 10-ї Міжнар. конф. «Крим-2003») [Електронний ресурс] / Л. Й. Костенко, А. О. Чекмарьов, А. Г. Бровкін, І. А. Павлуша //

Бібліотечний вісник — 2003. — № 4. — С. 43. — Режим доступу до журн.: http://www.nbuv.ua/articles/2003/03klinko.htm.