МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ

до практичних занять з дисципліни

«Технології розробки комп'ютерних систем»

для студентів усіх форм навчання

за спеціальністю 122 Комп'ютерні науки

Всі цитати, цифровий, фактичний матеріал та бібліографічні відомості перевірені, написання одиниць відповідає стандартам

ЗАТВЕРДЖЕНО
кафедрою ІУС
Протокол № 1___
від "29" 08 2022

Харків 2022

Методичні вказівки до практичних занять з дисципліни «Технології розробки комп’ютерних систем» для студентів усіх форм навчання за спеціальністю 122 Комп’ютерні науки [Електронне видання] / Упоряд.: Юр’єв І.О. – Харків: ХНУРЕ, 2022. – 71 с.
.

Упорядники: І.О. Юр’єв

Рецензенти: С.Ф. Чалий, д.т.н., проф. каф. ІУС ХНУРЕ

ЗМІСТ

ВСТУП

Проектування інформаційно-управляючих систем (ІУС) починається з визначення мети проекту. Основне завдання будь-якого успішного проекту полягає в тому, щоб на момент впровадження системи і протягом всього часу її експлуатації можна було забезпечити:

- необхідну функціональність системи і ступінь її адаптації до умов функціонування, які постійно змінюються;
- необхідну масштабованість системи;
- необхідну продуктивність системи і прийнятний час реакції системи на запит;
- безвідмовну роботу системи в необхідному режимі – готовність і доступність системи для обробки запитів користувачів;
- простоту експлуатації і супроводу системи;
- необхідний рівень інформаційної безпеки.

Проектування інформаційних систем охоплює такі області як:

- проектування видів забезпечень системи відповідно до функціональних вимог до системи (проектування схеми бази даних (БД), програм, екранних форм, звітів, які забезпечуватимуть виконання запитів до даних);
- облік не функціональних вимог до системи (наприклад, використання конкретного середовища або технології: топології мережі, конфігурації апаратних засобів, певної архітектури системи, паралельних обчислень, розподіленої обробки даних тощо).

Метою комплексу практичних занять з дисципліни «Проектування організаційних і технологічних ІУС» (ПОТІ), опис якого наведено в даних методичних вказівках, є вивчення сучасних технологій і методів розробки ІУС, однією з яких є уніфікована мова моделювання (Unified Modeling Language – UML). Візуальні UML-моделі можуть застосовуватися для моделювання предметної області (ПрО) і системи, що проектується, з різноманітних точок зору на різних етапах розробки і еволюції системи.

Комплекс практичних з дисципліни ПОТІ включає основні види діаграм UML 2.0 і охоплює всі сучасні області використання UML при проектуванні ІУС відповідно до різних моделей життєвого циклу (ЖЦ) системи.

На практичних заняттях здійснюється розробка UML-моделі ро, яка доповнюється UML-моделлю ІУС, що розробляється в циклі лабораторних робіт з даної дисципліни.

Оскільки практичні заняття з дисципліни «Проектування організаційних і технологічних ІУС» орієнтовані на індивідуальну діяльність кожного студента, то розробка UML-моделей здійснюється для об'єкта автоматизації в рамках курсової роботи з даної дисципліни. Результати виконання практичних занять рекомендується використовувати в дипломній роботі спеціаліста (аналіз предметної області, розробка рішень щодо інформаційного (ІЗ), програмного (ПЗ), технічного (ТЗ) забезпечень ІУС) або магістерській атестаційній роботі (практична апробація отриманих наукових результатів). Студентові пропонується самому вибрати об'єкт автоматизації, клас і функції проектованої системи, погодивши свій вибір з викладачем.

Матеріал, представлений в даних методичних вказівках, логічно взаємозв'язаний, що дає можливість студентам побудувати UML-модель об'єкта автоматизації в рамках етапів вибраної моделі ЖЦ ІУС.

ПРАКТИЧНЕ ЗАНЯТТЯ 1.
РОЗРОБКА UML-МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ З ВИКОРИСТАННЯМ ДІАГРАМ ПРЕЦЕДЕНТІВ (USE CASE DIAGRAM)

1.1 Мета заняття

Вивчення нотації діаграми прецедентів (use case diagram) і її використання в рамках розробки моделі ПрО. Відпрацювання переходу від процесних моделей до діаграми прецедентів (діаграми варіантів використання). Розробка UML-моделі ПрО.

1.2 Методичні вказівки з організації самостійної роботи студентів

При підготовці до практичних занять необхідно повторити лекційний матеріал з таких питань: методи виділення бізнес-акторів і бізнес-варіантів використання.

1.2.1 Нотація use case diagram

На діаграмах use case застосовуються такі типи сутностей: дійові особи (actor); варіанти використання (прецедент, use case); примітки (note) і пакети (package).

Між цими сутностями встановлюються наступні типи зв’язків:

- асоціація між дійовою особою і варіантом використання;
- узагальнення між дійовими особами;
- узагальнення між варіантами використання;
- залежність між варіантами використання;
- залежність між пакетами.

Конструкція або стандартний елемент мови UML «варіант використання» застосовується для специфікації загальних особливостей поведінки системи або будь-якої сутності ПрО без розгляду внутрішньої структури цієї сутності. Будь-який варіант використання повинен мати ім'я, що відрізняє його від інших прецедентів (рис. 1.1). На практиці для іменування прецедентів використовують короткі дієслівні фрази, що позначають деяку дію системи.

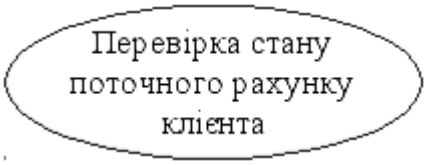

Рисунок 1.1 – Прецедент

Кожен варіант використання визначає сценарій – послідовність дій, які мають бути виконані системою, що проектується, при її взаємодії з відповідним актором. Іншими словами, прецедент служить для опису сервісів, які система надає акторові. Окрім цього, варіанти використання неявно встановлюють вимоги, що визначають, як користувачі повинні взаємодіяти з системою, аби мати можливість коректно працювати з тими сервісами, що надаються даною системою.

Будь-який з варіантів використання може бути декомпозований далі на множину підваріантів використання. Ця множина в цілому повинна визначати всі можливі сторони очікуваної поведінки системи.

Актором (actor) або дійовою особою називається будь-яка сутність, що взаємодіє з системою ззовні. З синтаксичної точки зору, дійова особа – це стереотип класифікатора, який позначається спеціальним значком (рис. 1.2).

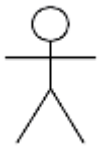

Рисунок 1.2 – Актор

Для дійової особи вказується лише ім'я, що ідентифікує її в системі. Семантично дійова особа – це множина логічно взаємозв'язаних ролей. З прагматичної точки зору головним є те, що дійові особи знаходяться поза проектованою системою (або даної частини системи). У типових випадках різні дійові особи призначаються для категорій користувачів (якщо їх удається виділити природним чином), зовнішніх програмних і апаратних засобів (якщо система взаємодіє з такими).

У мові UML існує декілька стандартних видів зв’язків між акторами і варіантами використання:

– Відношення асоціації (association relationship).

Актори взаємодіють з системою за допомогою передачі і прийому повідомлень від варіантів використання. Повідомленням є запит актора на надання сервісу від системи і реакція на здобуття цього сервісу. Ця взаємодія може бути виражена за допомогою асоціацій між окремими акторами і варіантами використання або класами. На діаграмі варіантів використання, так само як і на інших діаграмах, відношення асоціації позначається суцільною лінією між актором і варіантом використання. Ця лінія може мати додаткові умовні позначення, такі, як ім'я і кратність.

– Відношення включення (include relationship).

Відношення включення специфікує той факт, що деякий варіант використання завжди містить поведінку, визначену в іншому варіанті використання. Відношення включення, направлене від варіанту використання «А» до варіанту використання «Б», вказує, що кожен екземпляр варіанту «А» включає функціональні властивості, задані для варіанту «Б» (рис. 1.3).

Один варіант використання може бути включений в декілька інших варіантів, а також сам включати інші варіанти. Варіант використання, що включається, може бути незалежним від базового варіанту в тому сенсі, що він надає останньому деяку інкапсульовану поведінку, деталі реалізації якої приховані і можуть бути легко перерозподілені між декількома варіантами використання, що включаються.

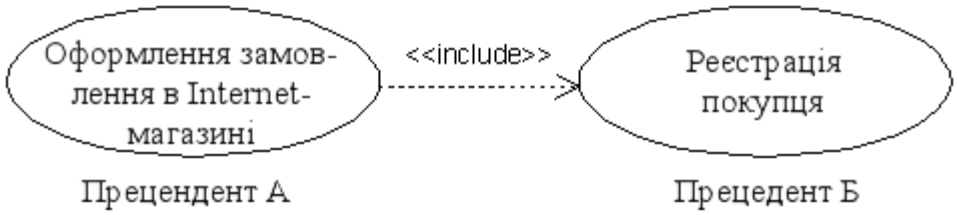

Рисунок 1.3 – Приклад графічного зображення відношення включення між варіантами використання

– Відношення розширення (extend relationship).

Відношення extend визначає взаємозв'язок одного варіанту використання з деяким іншим варіантом використання, функціональність або поведінка якого використовуються першим прецедентом не завжди, а лише при виконанні деяких додаткових умов. Так, якщо має місце відношення розширення від варіанту використання «Б» до варіанту використання «А», то це означає, що властивості екземпляра варіанту використання «А» можуть

бути доповнені властивостями варіанту використання «Б», що його розширює (рис. 1.4).

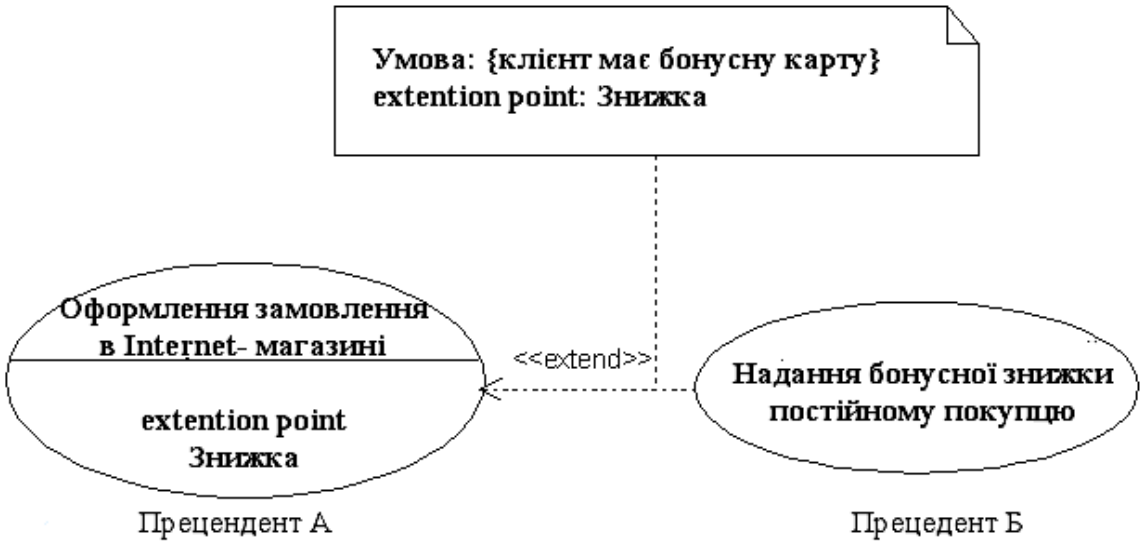

Рисунок 1.4 – Приклад графічного зображення відношення розширення між варіантами використання із заданою умовою розширення

Відношення розширення відзначає той факт, що один з варіантів використання може приєднувати до своєї поведінки деяку додаткову поведінку, визначену для іншого варіанту використання. Відносно «extend» базовий прецедент нічого не знає про прецеденти, що його розширюють, він просто надає для них точки розширення. Це істотно відрізняє «extend» від відношення «include», де базові прецеденти залишаються неповними без прецедентів, що включаються.

– Відношення узагальнення (generalization relationship).

Відношення generalization вказує на те, що деякий варіант використання «А» може бути узагальнений до варіанту використання «В». Відношення узагальнення між варіантами використання застосовується у тому випадку, коли необхідно відзначити, що дочірні варіанти використання мають всі атрибути і особливості поведінки батьківських варіантів. У свою чергу, дочірні варіанти можуть наділятися новими властивостями поведінки, відсутніми в батьківських варіантах використання, а також уточнювати або модифікувати успадковані від них властивості поведінки.

Між окремими акторами може також існувати відношення узагальнення. Два і більше акторів можуть мати спільні властивості, тобто взаємодіяти з однією і тією ж множиною варіантів використання однаковим

чином. Наявність направленого відношення узагальнення від актора «А» до актора «В» відзначає той факт, що кожен екземпляр актора «А» є одночасно екземпляром актора «В» і має всі його властивості (рис. 1.5).

Всі сервіси системи мають бути явно визначені на діаграмі варіантів використання, і жодних інших сервісів, відсутніх на даній діаграмі, проектована система надавати не може.

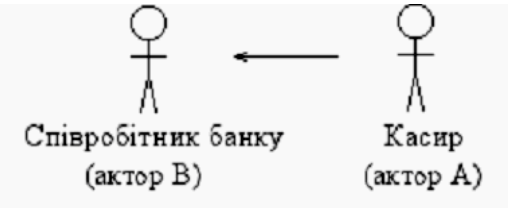

Рисунок 1.5 – Приклад графічного зображення відношення узагальнення між акторами (у різних CASE-засобах можуть застосовуватися різні способи зображення елементів діаграм)

Третім типом сутностей, що вживається на діаграмі use case, є примітка. Примітка призначена для включення в модель довільної текстової інформації, яка має безпосереднє відношення до контексту проекту, що розробляється. Такою інформацією можуть бути коментарі розробника (наприклад, дата і версія розробки діаграми або її окремих компонентів), обмеження (наприклад, на значення окремих зв'язків або екземпляри сутностей) і помічені значення. Стосовно діаграм варіантів використання примітка може мати уточнюючу інформацію, що відноситься до контексту тих чи інших варіантів використання.

Примітки можуть мати стереотипи. У UML визначено два стандартні стереотипи для приміток: «requirement» – описує загальні вимоги до системи і «responsibility» – описує відповідальність сутності (класифікатора). Примітки першого типа часто присутні на діаграмах використання, а примітки другого типа – на діаграмах класів.

Мова UML включає спеціальні механізми розширення, які дозволяють ввести в розгляд додаткові графічні позначення, орієнтовані для вирішення завдань визначеною ПрО. Приклади подібних позначень, які використовуються для моделювання бізнес-систем і можуть бути зображеними на діаграмах варіантів використання: бізнес-актор, бізнес-співробітник і бізнес-варіант використання.

Бізнес-актор (business actor) – індивідуум, група, організація, компанія або система, які взаємодіють з бізнес-системою, що моделюється, але не входять до неї, тобто не є її частиною (рис.1.6, а). Прикладами бізнес-акторів є клієнти, покупці, постачальники, партнери. Загальна властивість бізнес-акторів полягає в тому, що вони є ініціаторами або клієнтами бізнес-процесів системи.

Співробітник (business worker) – індивідуум, який діє усередині бізнес-системи, що моделюється, взаємодіє з іншими співробітниками і є учасником бізнес-процесу системи (рис.1.6, б). Прикладами співробітників є менеджери, адміністратори, касири, інженери. Загальна властивість співробітників полягає в тому те, що вони є суб'єктами і входять до складу системи.

Бізнес-варіант використання (business use case) – варіант використання, що визначає послідовність дій системи, що моделюється, направлених на виконання окремого бізнес-процесу (рис. 1.6, в). Бізнес-варіанти використання є концептуальною моделлю окремих бізнес-процесів системи.

Рисунок 1.6 – Графічні зображення бізнес-актора (а), бізнес-співробітника (б) і бізнес-варіанту використання (в)

Реалізація варіантів використання не зображується на діаграмах use case. Для моделювання логічних і фізичних аспектів реалізації призначені інші типи канонічних діаграм. Так діаграми взаємодії (interaction diagrams) допомагають виділити зв’язки між різними суб'єктами, що діють в системі. Діаграми послідовності (sequence diagram), яка є різновидом діаграми взаємодії, використовуються для моделювання часової послідовності подій в сценарії прецедентів. Коли ж предметом дослідження є зв’язки між суб'єктами, що діють, і системою, застосовуються діаграми комунікації (communication diagrams). Діаграми активності (activity diagram) використовуються при моделюванні паралельних процесів, коли виникає необхідність відобразити ту обставину, що процес може виконуватися паралельно з іншими процесами протягом всього сценарію прецеденту.

1.3 Контрольні запитання і завдання

1.3.1 Контрольні запитання

1) Що описують функціональні вимоги до системи?
2) Яку мету переслідує розробка діаграм варіантів використання?
3) Дайте визначення основним елементам діаграми використання.
4) Які зв’язки можуть існувати на діаграмі прецедентів?
5) Які типи зв’язків можуть бути використані для опису взаємодії акторів?
6) У чому полягає зміст зв’язків включення і розширення?
7) Що називається точкою розширення?

1.3.2 Завдання

Розробити UML-моделі ПрО за допомогою use case діаграм. Виділення прецедентів здійснити на основі структурних діаграм IDEF0, DFD, а виділення акторів – на основі діаграм SwimLane.

ПРАКТИЧНЕ ЗАНЯТТЯ 2.
РОЗШИРЕННЯ UML-МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ ЗА ДОПОМОГОЮ РОЗРОБКИ ДІАГРАМ ДІЯЛЬНОСТІ (ACTIVITY DIAGRAM)

2.1 Мета заняття

Вивчення нотації діаграми діяльності (activity diagram). Вживання даної діаграми для розширення UML-моделі Про за допомогою опису поведінки прецедентів. Аналіз взаємозв'язку activity diagram з іншими діаграмами мови UML в рамках розроблених моделей. Розробка діаграми activity для Про.

2.2 Методичні вказівки з організації самостійної роботи студентів

При підготовці до практичних занять необхідно використовувати розроблену на попередньому практичному занятті діаграму use case для вибраної ПрО.

2.2.1 Нотація activity diagram

Діаграма діяльності – діаграма, яка відображує поведінку об'єкта або системи з використанням моделей потоку даних і потоку управління. Activity діаграми моделюють поведінку за допомогою «гри» маркерів (token game). Ця гра описує потік маркерів, який рухається мережею вузлів і ребер згідно з певними правилами. Кожен маркер відрізняється від будь-якого іншого, навіть якщо він містить те ж значення, що й інший. Маркери на діаграмах діяльності UML можуть представляти: потік управління, об'єкт і деякі дані. Стан системи у будь-який момент часу визначається розташуванням її маркерів.

Діяльності (activity) – це системи вузлів (nodes), сполучених ребрами (edges).

Існує три категорії вузлів:

– *вузли дії (action nodes)* – представляють окремі одиниці роботи, елементарні в рамках діяльності. Оскільки ці вузли описують здійснення деяких дій, зазвичай їх імена є дієсловами;

– *вузли управління (control nodes)* – управляють потоком діяльності;

– *об'єктні вузли (object nodes)* – представляють об'єкти, використовувані в діяльності.

Ребра представляють потоки. Існує два типи ребер:

– *ребра потоків управління (control flows)* – представляють потік управління діяльності;

– *ребра потоків об'єктів (object flows)* – представляють потік об'єктів діяльності.

На activity діаграмах використовуються чотири типи вузлів дії:

– *вузол виклику дії (call action node),* який ініціює діяльність, поведінку або операцію (рис. 2.1);

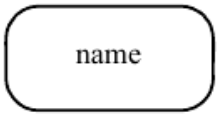

Рисунок 2.1 – Нотація вузла виклику дії

– *вузол дії передачі сигналу (send signal action)* є дією, яка на основі своїх входів створює екземпляр сигналу і передає його до цільового об'єкта (рис.2.2);

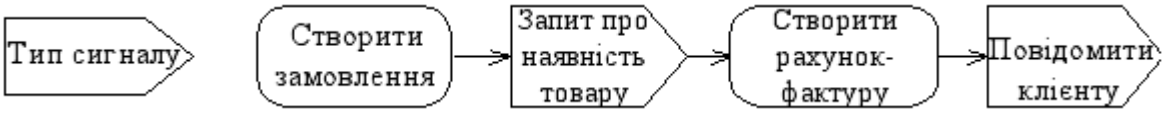

Рисунок 2.2 – Приклад вузла дії передачі сигналу

– *вузол дії прийому події (accept event action)* є дією, яка чекає настання деякої події (рис. 2.3);

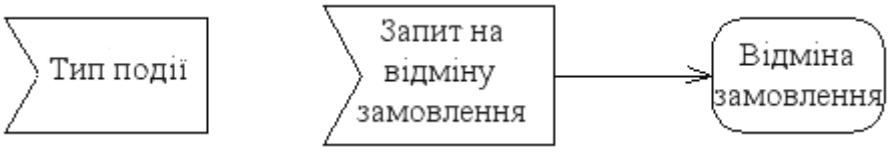

Рисунок 2.3 – Приклад вузла дії прийому події

– *вузол дії, що приймає події часу (accept time event action),* є спеціальним випадком вузла дії прийому події і реагує на настання деякого моменту часу (рис. 2.4). Цей тип вузла характеризується часовим виразом і генерує подію часу, коли цей вираз стає істинним. Поведінка такого вузла залежить від наявності вхідного ребра. Часовий вираз може вказувати на: деяку подію (наприклад, кінець фінансового року); конкретний момент часу (наприклад, 0:00 11/03/1960); часовий інтервал (наприклад, чекати 10 с.).

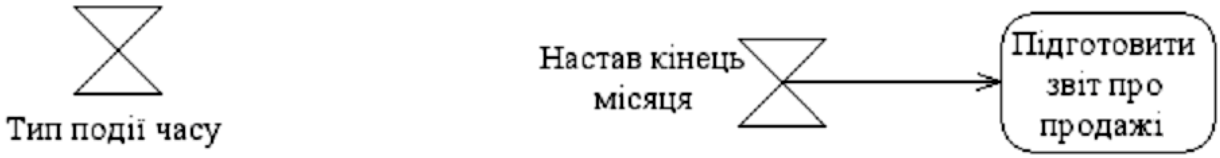

Рисунок 2.4 – Приклад вузла, що приймає подію часу

Вузли дії виконуються у випадках, коли вхідні маркери одночасно поступили на всі вхідні ребра і задовольняють всім його локальним передумовам. Після закінчення виконання вузла дії перевіряється локальна постумова. Якщо вона виконується, вузол одночасно пропонує маркери на всіх своїх вихідних ребрах.

Для контролю потоку діяльності використовуються такі вузли управління:

– *початковий вузол (initial node)* є вузлом управління, в якому починається потік при виклику діяльності;

– *вузол закінчення діяльності (activity final node)* є вузлом управління, який припиняє або зупиняє всі потоки в діяльності;

– *вузол закінчення потоку (flow final node)* є фінальним вузлом, який завершує окремий потік управління або потік об'єктів, не завершуючи діяльності, що містить його.

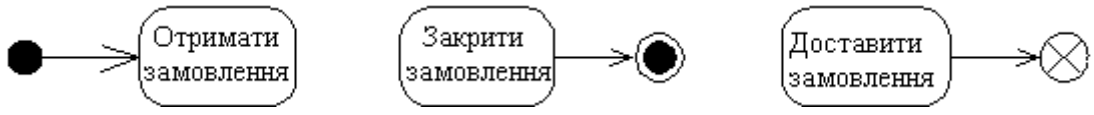

Рисунок 2.5 – Приклади початкового вузла, вузлів закінчення діяльності і закінчення потоку

– *вузол рішення (decision node) є* вузлом управління, який здійснює вибір між потоками, що з нього виходять.

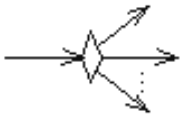

Рисунок 2.6 – Нотація вузла рішення

Вузол, відмічений стереотипом «decisionInput» (вхідні дані рішення), представляє умову ухвалення рішення. Його результат використовується сторожовими умовами на вихідних ребрах.

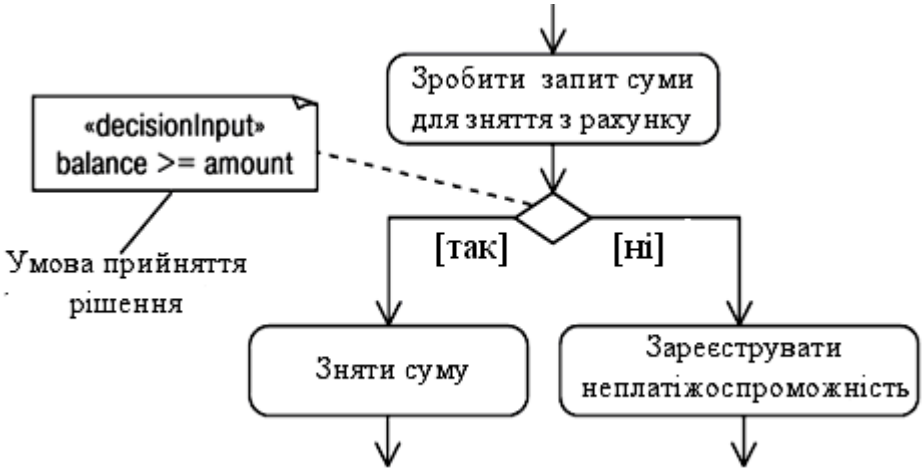

Рисунок 2.7 – Фрагмент діяльності з вузлом рішення, поміченим стереотипом «decisionInput»

– *вузол злиття (merge node)* є вузлом управління, який сполучає разом декілька альтернативних потоків.

Рисунок 2.8 – Приклад вузла злиття

– *вузол розділення (fork node)* – є вузлом управління, який розщеплює потік на декілька паралельних потоків. У цьому вузлі маркери, що поступають по вхідних ребрах, дублюються і пропонуються на всіх вихідних

ребрах одночасно. Тим самим єдиний вхідний потік розділяється на декілька паралельних вихідних потоків. У кожного вихідного ребра може бути сторожова умова, і маркер, як і у вузлах рішення, може передаватися по ребру лише в разі виконання сторожової умови. В даному випадку передбачається, що жодні з вузлів з'єднання, що знаходяться далі, не залежать від проходу маркерів, що передаються через дугу із сторожовою умовою. Якщо цього виключити не можна, то необхідно ввести вузол рішення з подальшим вузлом злиття.

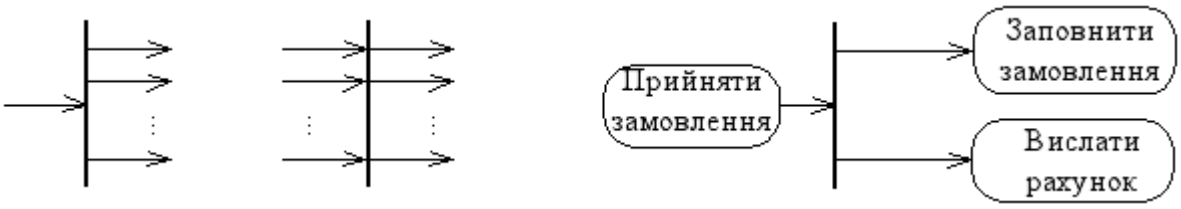

Рисунок 2.9 – Приклад вузла розділення

– *вузол з'єднання (join node)* – є вузлом управління, який синхронізує декілька потоків. Вузли з'єднання можуть мати додаткову логічну специфікацію умов, при виконанні яких вони повинні генерувати маркер на виході. Якщо для вузла з'єднання існують маркери у всіх його вхідних дугах, то дузі, що виходить, пропонуються маркери згідно з такими правилами: якщо всі маркери, що пропонуються на вхідних дугах, є маркерами управління, то вихідній дузі пропонується один маркер управління. Ці вузли синхронізують потоки: вони здійснюють операцію логічного «І» над всіма своїми вхідними ребрами. Приклади зображення вузла з'єднання з додатковою специфікацією представлені на рис. 2.10.

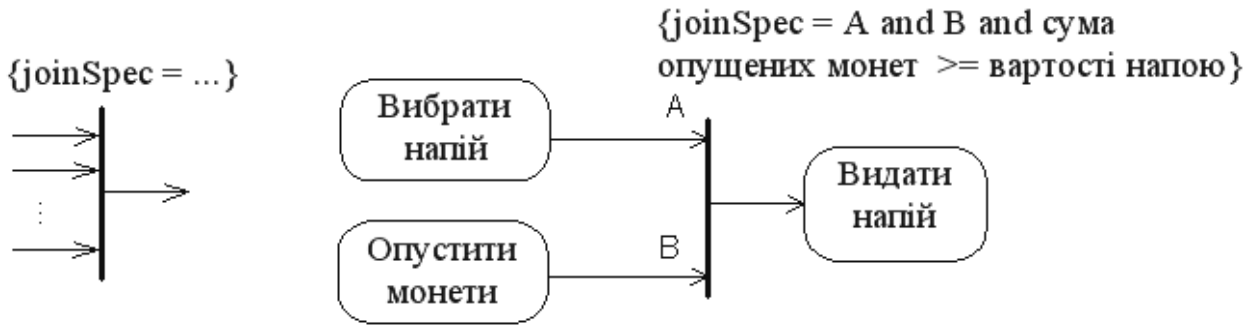

Рисунок 2.10 – Приклад вузла з'єднання

Об'єктні вузли (object node, рис. 2.11) – це спеціальні вузли, які показують, що екземпляри конкретного класифікатора доступні в даній точці діяльності. Вхідні і вихідні ребра об'єктних вузлів називають потоками об'єктів.

Семантика об'єктного вузла передбачає, що вони діють як буфери – ділянки діяльності, де можуть знаходитися об'єктні маркери в очікуванні прийняття іншими вузлами.

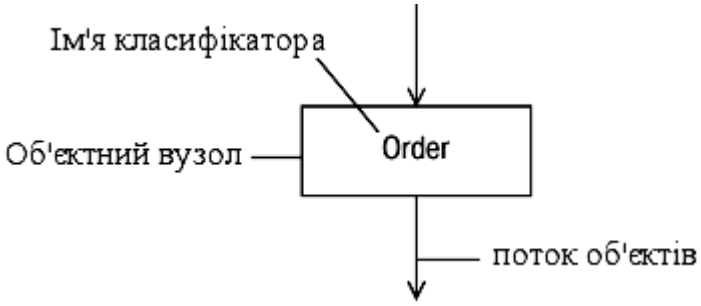

Рисунок 2.11 – Нотація object node

Існують такі різновиди вузлів об'єкту:

– вузол об'єкта для маркерів об'єктів, що знаходяться в спеціальному стані, додатково містить специфікацію цього стану, яка записується в прямих дужках нижче за ім'я типа (рис. 2.12, а);

– вузол об'єкта для маркерів, що містять множину об'єктів різних типів, містить імена всіх цих об'єктів (рис. 2.12, б);

– вузли об'єктів для сигналів; зображуються за допомогою спеціального символу, усередині якого записується ім'я типа сигналу (рис. 2.12, в)

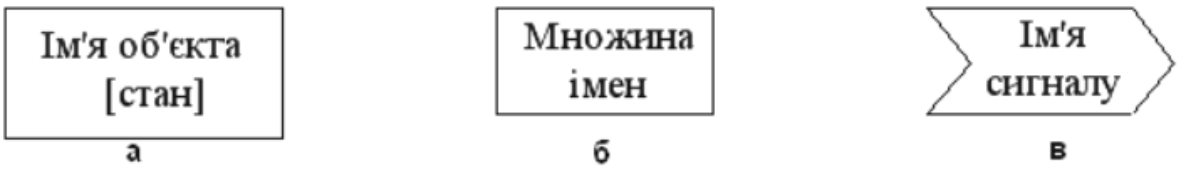

Рисунок 2.12 – Приклад об'єктних вузлів

Зазвичай кожен об'єктний вузол може утримувати нескінченне число об'єктних маркерів. Проте інколи необхідно обмежити розмір буфера. Для цього задають верхню границю (upper bound) об'єктного вузла (рис. 2.13, а). Вона показує максимальне число маркерів, які можуть утримуватися у вузлі

у будь-який момент часу. Для об'єктних вузлів можна задати порядок розташування (ordering), що визначає поведінку буфера (рис. 2.13, б), і селективну поведінку (selection behavior). Це закріплена за вузлом поведінка, згідно її якою об'єкти з вхідних потоків вибираються відповідно до деякого критерію, визначеного розробником моделі.

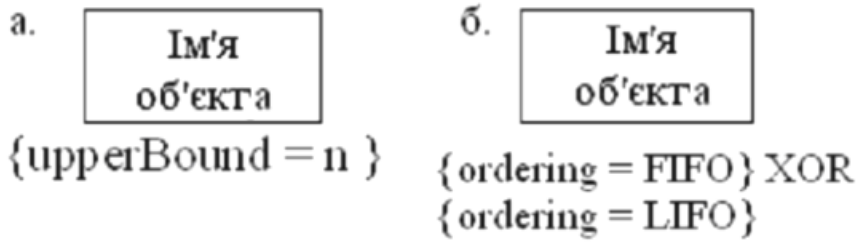

Рисунок 2.13 – Приклад об'єктних вузлів з обмеженнями

Критерій селективної поведінки задається приміткою із стереотипом «selection», як показано на рис. 2.14.

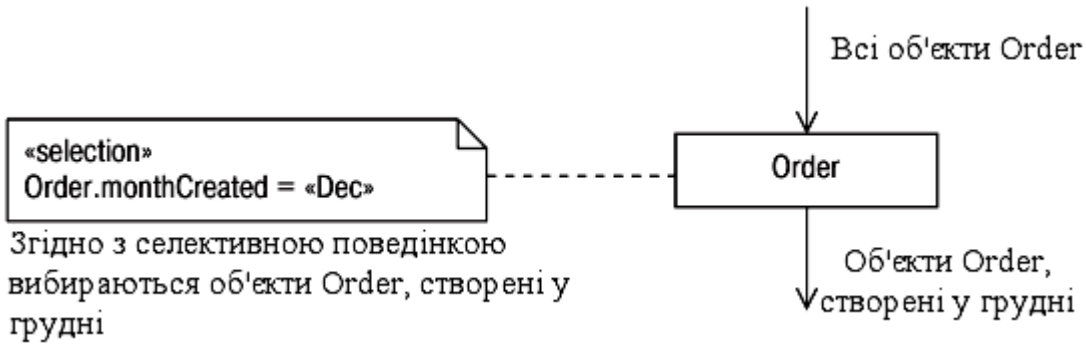

Рисунок 2.14 –Об’єктний вузол з обмеженнями

Як уже зазначалося, стан системи, що виконується, у будь-який момент часу може бути представлено розташуванням її маркерів. Об'єктні вузли можуть представляти об'єкти, що знаходяться в певному стані. Стани об'єктів, на які посилаються об'єктні вузли, можуть бути змодельовані за допомогою кінцевих автоматів. Не кожне виконання дії або передача маркера створює помітну зміну в стані системи з точки зору її кінцевих автоматів. Проте, розташування маркерів забезпечує зв'язок між діаграмами діяльності і діаграмами станів, які мають бути гарантовано погодженими для конкретного елементу моделі.

Для угрупування дій, які відносяться до однієї діяльності і мають деяку загальну характеристику, призначений елемент моделі розділ діяльності

(activity partition). Приклад вживання розбиття з нотації UML2 наведений на рис. 2.15.

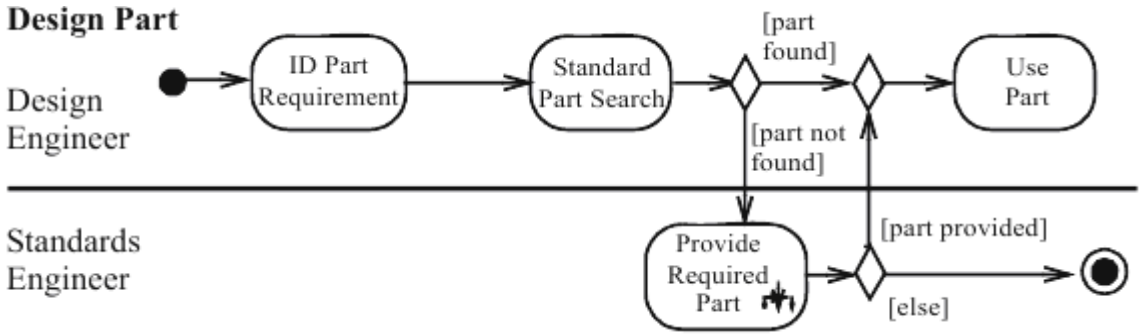

Рисунок 2.15 – Приклад вживання розбиття

Правила використання розділу діяльності на діаграмі:

– Будь-які вузол або дуга діяльності не можуть одночасно належати двом або більш розбиттям в одному і тому ж вимірі.

– Розбиття не впливає на потоки маркерів, за які несуть відповідальність екземпляри класифікаторів, представлені окремим розбиттям.

– Розбиття одного виміру і рівня вкладеності має бути представлене частинами внутрішньої структури одного і того ж класифікатора.

– Розбиття може бути представлене атрибутом, а його ід розбиття – значеннями цього атрибуту.

– Якщо розбиття має деякий вимір, то воно не може міститися ні в якому іншому розбитті.

На рис. 2.15 зображено розподіл функцій між інженером-проектувальником та інженером зі стандартизації при проектуванні частини модульної системи (Design part). Здійснюється пошук необхідної стандартизованої частини за її ідентифікатором (ID). У разі відсутності такої частини інженер зі стандартизації надає необхідну частину, яка використовується в подальшому проектуванні.

Діаграми діяльності грають важливу роль в розумінні процесів реалізації алгоритмів виконання операцій класів і потоків управління в модельованій системі. Використовувані для цієї мети традиційні схеми алгоритмів мають серйозні обмеженнями щодо представлення паралельних процесів та їх синхронізації. Вживання розділів діяльності і об'єктів відкриває додаткові можливості для наочного представлення бізнес-процесів,

дозволяючи специфікувати та відокремлювати діяльність окремих підрозділів компаній і фірм.

2.3 Контрольні запитання і завдання

2.3.1 Контрольні запитання

1. На яких етапах різних моделей ЖЦ розробляються activity діаграми?
2. Які елементи UML-моделі Про або ІУС можуть бути уточнені за допомогою діаграм активностей?
3. Які категорії вузлів існують на activity diagram? Наведіть приклади?
4. Які потоки можуть існувати на діаграмі activity?
5. Чим діаграми діяльності відрізняються від блок-схем?
6. У яких архітектурних представленнях системи (відповідно до стандарту UML Reference Manual) може застосовуватися activity diagram?
7. Проаналізуйте взаємозв'язок activity diagram з іншими діаграмами мови UML.

2.3.2 Завдання

Розробити activity-діаграми, що розширюють модель ПрО за допомогою опису поведінки прецедентів use case діаграми ПрО. При розробці діаграми слід використовувати розділення діяльностей.

ПРАКТИЧНЕ ЗАНЯТТЯ 3.
РОЗШИРЕННЯ UML-МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ ЗА ДОПОМОГОЮ РОЗРОБКИ ДІАГРАМ КЛАСІВ (CLASS DIAGRAM)

3.1 Мета заняття

Вивчення нотації діаграми класів (class diagram) і її використання в рамках розробки моделі Про. Здобуття навичок моделювання класів, розробка class-діаграми для Про.

3.2 Методичні вказівки з організації самостійної роботи студентів

При підготовці до практичних занять необхідно використовувати розроблені на попередніх практичних заняттях діаграми use case і activity для вибраної ПрО.

3.2.1 Нотація class diagram

Діаграма класів є деяким графом, вершинами якого є елементи типу «класифікатор», які зв'язані різними типами структурних стосунків (рис. 4.1). Тому діаграму класів прийнято вважати графічним представленням таких структурних взаємозв'язків логічної моделі системи, які не залежать від часу (інваріантні).

Класифікатор (classifier) – спеціальне поняття, призначене для класифікації екземплярів, які мають загальні характеристики. Прикладами класифікаторів є клас і об'єкт.

Об'єктом є окрема сутність з явно вираженими границями, яка інкапсулює стан і поведінку; екземпляр класу. Виділяють властивості, загальні для всіх об'єктів:

– ідентифікатор (identity) – відображення факту існування і унікальності об'єкта в часі і просторі.

– стан (state) – визначається значеннями атрибутів об'єкта і його зв’язками з іншими об'єктами в конкретний момент часу.

– поведінка (behavior).

Виклик операції об'єкта завжди приводить до зміни значень одного або більше його атрибутів або зв’язків з іншими об'єктами. Це може зумовити перехід станів – цілеспрямований перехід об'єкта з одного стану в інший.

Стан об'єкта може впливати і на його поведінку. Наприклад, якщо в принтері закінчилося чорнило (стан об'єкту = OutOfBlackInk), виклик операції printDocument() приведе до повідомлення про помилку. Тому поведінка printDocument() є залежною від стану.

Основні позначення діаграми класів приведені на рис. 3.1.

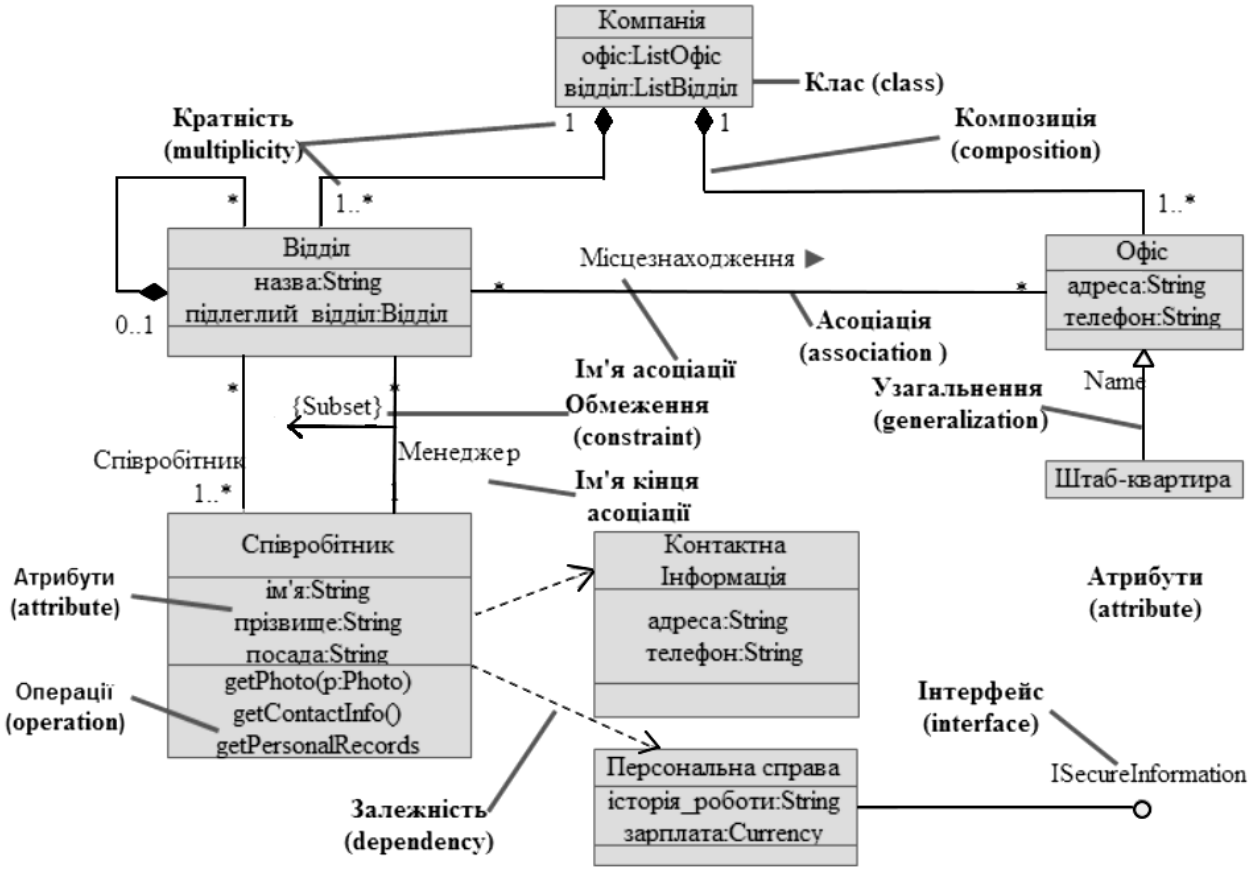

Рисунок 3.1 – Основні позначення на діаграмі класів

Клас (class) – елемент моделі, який описує безліч об'єктів, що мають однакові специфікації характеристик, обмежень і семантики. Характеристика (feature) – поняття, призначене для специфікації особливостей структури і поведінки екземплярів класифікаторів. Структурна характеристика (structural feature) є характеристикою класифікатора, який типізується. Ця характеристика специфікує структуру його екземплярів. Характеристика поведінки (behavioral feature) є характеристикою класифікатора, яка специфікує деякий аспект поведінки його екземплярів. Операція – це опис частини поведінки. Реалізація цієї поведінки називається методом (method).

Між класом і об'єктами цього класу встановлюється взаємозв'язок «instantiate» (створити екземпляр). В даному випадку використовується один

з трьох механізмів розширення в UML – стереотип. Стереотип «instantiate» перетворює звичайну залежність на зв’язок конкретизації між класом і об'єктами цього класу (рис.3.2).

Розрізняють такі різновиди класів:

– *абстрактний (abstract)* клас не може мати екземплярів або об'єктів, для позначення його імені використовується похилий шрифт (курсив);

– *активний клас (active class)* – клас, кожний екземпляр якого має свою власну нитку управління;

– *пасивний клас (passive class)* – клас, кожний екземпляр якого виконується в контексті деякого іншого об'єкта.

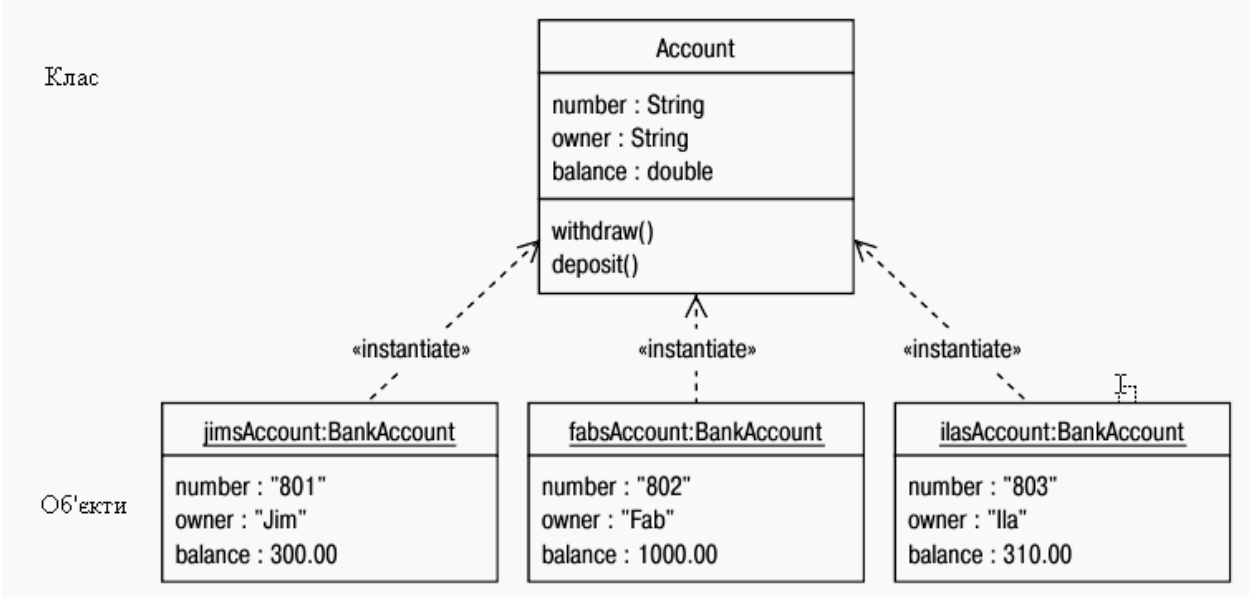

Рисунок 3.2 – Відношення «instantiate»

Кваліфіковане ім'я класу (qualified name) використовується для того, щоб явно вказати, до якого пакету відноситься той чи інший клас. Для цього застосовується спеціальний символ як роздільник імені – подвійна двокрапка “::”. Ім'я класу без символу роздільника називається простим ім'ям класу.

У візуальному синтаксисі обов'язковою частиною класу є лише ім'я. Всі останні частини і доповнення необов'язкові (рис. 3.3). Ім'я класу має бути унікальним в межах пакету, який описується деякою сукупністю діаграм класів, і записується в стилі „UpperCamelCase”. Рекомендується як імена класів використовувати іменники, записані без пропусків.

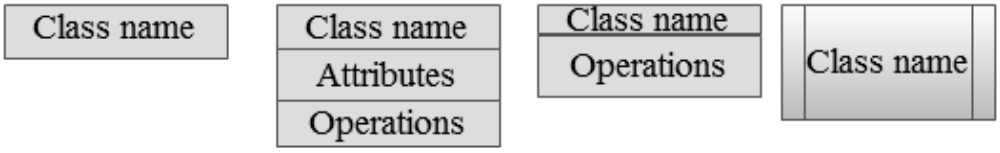

Рисунок 3.3 – Варіанти зображення класу на діаграмі

У аналітичних моделях зазвичай необхідно показувати лише: ім'я класу, ключові атрибути, ключові операції і стереотипи. Помічені значення, параметри операцій, видимість і вихідні значення найчастіше не показують. Єдиною обов'язковою частиною UML-синтаксису для атрибутів є ім'я атрибуту. Атрибут (attribute) класу служить для представлення окремої структурної характеристики або властивості, яка є загальною для всіх об'єктів даного класу. У мові UML кожному атрибуту класу відповідає окремий рядок тексту:

<атрибут> ::= [<видимість>] [/] <ім'я> [: <тип атрибуту>] [[<кратність>]] [= <початкове значення>] [{<модифікатор атрибуту [, <модифікатор атрибуту>]* }]

Видимість (visibility) може приймати одне з чотирьох можливих значень і відображуватися або за допомогою спеціального символу, або відповідного ключового слова (<видимість>::= ‘+’ | ‘–‘ | ‘#’ | ‘~’). Доповнення видимості (visibility) застосовується до атрибутів і операцій класу, до імен ролей в зв’язках. На стадії аналізу на діаграмі класів зазвичай не наводять модифікатори видимості.

Елементи запису атрибуту:

– “/” означає, що атрибут є похідним (derive) або обчислюваним. Значення похідного атрибуту може бути обчислене на основі значень інших атрибутів цього або інших класів. При використанні похідних атрибутів розробник повинен явно вказати процедуру або операцію для обчислення їх значень.

– <ім'я> (name) є рядком тексту, який використовується як ідентифікатор відповідного атрибуту і тому має бути унікальною в межах даного класу. Ім'я атрибуту є єдиним обов'язковим елементом в позначенні атрибуту, повинно починатися з рядкової (малою) букви і, як правило, не повинно містити пропусків. Імена атрибутів записуються в стилі „lowerCamelCase”. Зазвичай як імена атрибутів використовуються іменники

або іменні групи, тому що атрибути вказують на деяку «сутність», наприклад баланс рахунку.

– <тип атрибуту> (attribute type) є ім'я класифікатора, який є типом даного атрибуту.

– <кратність> (multiplicity) атрибуту характеризує загальну кількість конкретних значень для атрибуту, які можуть бути задані для об'єктів даного класу.

– <початкове значення> (default value) – деяке вираження, яке служить для завдання початкового значення або значень даного атрибуту у момент створення окремого екземпляра відповідного класу. В аналізі початкові значення використовуються лише тоді, коли вони можуть виразити або позначити важливе бізнес-обмеження.

– <модифікатор атрибуту> (attribute modifier) є текстовим вираженням, яке додає додаткову семантику даному атрибуту (стереотипи або помічені значення).

Приклади запису атрибутів:

+ ім’яСпівробітника: String {readOnly}

~ датаНародження : Date {readOnly}

/вікСпівробітника : Integer

– заробітнаПлатня : Currency = 500.00

Операція (operation) класу служить для представлення окремої характеристики поведінки, яка є загальною для всіх об'єктів даного класу. Імена операцій записуються в стилі „lowerCamelCase” і зазвичай є дієсловами. Поєднання імені операції, типів всіх її параметрів і типа, що повертається, утворює *сигнатуру операції*. Загальний формат сигнатури операції класу такий:

<операція> ::= [<видимість>]<ім'я операції> ([<список параметрів>]) [:[<тип результату, що повертається>] { <властивість операції> [, <властивість операції>]* }]

Елементи запису операції:

– <видимість> ::= ‘+’ | ‘-‘ | ‘#’ | ‘~’;

– <ім'я операції> (operation name) є рядком тексту, який використовується як ідентифікатор відповідної операції і тому має бути унікальною для кожної операції даного класу;

– <список параметрів> (parameter list) є переліком розділених комами формальних параметрів операції. Імена параметрів записуються в стилі

„lowerCamelCase” і зазвичай є іменниками. Кожен параметр має певний тип, який є класом або примітивним типом;

– <тип результату, що повертається> (return type) специфікує тип значення, що повертається даною операцією;

– <властивості операцій> є розширенням операції.

Найчастіше використовуються такі властивості:

1. redefines <ім'я операції> – дана операція перевизначає деяку успадковану операцію з ім'ям <ім'я операції>;

2. query – дана операція не змінює стану модельованої системи і, відповідно, не має побічного ефекту;

3. ordered – значення параметра, що повертається, є впорядкованими;

4. unique – значення параметра, що повертається, не можуть повторюватися.

– <обмеження> – вираження, яке специфікує деяке обмеження, вживане до даної операції.

Використовуються також передумови і постумови операції.

Приклади запису операцій:

+додати(in номерТелефону : Integer [*] {unique});

–змінити(in заробітнаПлата : Currency);

+створити() : Boolean;

toString(return : String);

toString( ) : String.

Для семантичного з'єднання класів, яке забезпечує можливість обміну повідомленнями між ними, використовуються такі зв'язки:

– асоціація (association);
– агрегація (aggregation);
– композиція (composition);
– узагальнення (generalization);
– залежність(dependency);
– реалізація (realization).

Асоціація – довільний взаємозв'язок між класами. Асоціації можуть мати: ім'я асоціації, імена ролей, кратність, можливість навігації (рис. 3.4, 3.5).

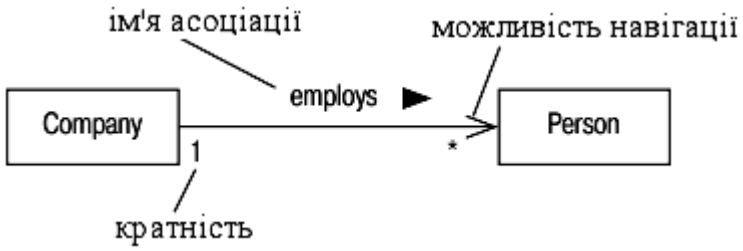

Рисунок 3.4 – Відображення асоціації з використанням її імені

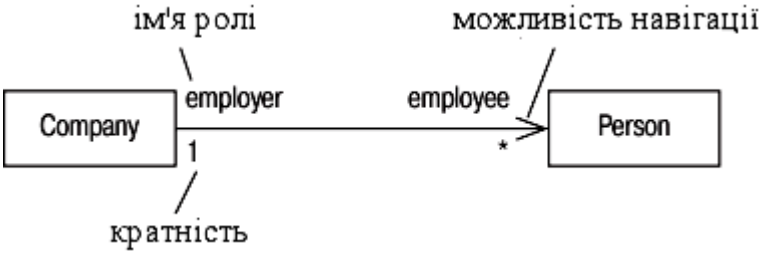

Рисунок 3.5 – Відображення асоціації з використанням імені ролей

Ім'я кінця асоціації специфікує роль (role), яку грає клас, розташований на відповідному кінці даної асоціації. Кратність кінця асоціації є одним з механізмів розширення відношення і специфікує можливу кількість екземплярів відповідного класу, яка може співвідноситися з одним екземпляром класу на іншому кінці цієї асоціації. Кратність зв'язку описує бізнес-правила, вимоги і обмеження. Її використання на діаграмах може виявляти необґрунтовані припущення відносно ПрО, які необхідно усувати якомога раніше.

Можливість навігації (navigability) вказує на можливість проходу від об'єкта вихідного класу до одного або більш об'єктів залежно від кратності цільового класу. Зміст навігації в тому, що повідомлення можуть посилатися лише в напрямі, в якому вказує стрілка. Найбільш поширеним стилем представлення навігації на діаграмі є позначення двоспрямованої асоціації лінією без стрілок, а односпрямованою – лінією з однією стрілкою. З точки зору реалізації на ОО мовах програмування, можливість навігації має на увазі, що у вихідного об'єкта є об'єктне посилання на цільовий об'єкт. Вихідний об'єкт може використовувати це посилання для відправки повідомлень цільовому об'єкту.

Для реалізації відношення типа „багато до багатьох” використовують клас-асоціацію. Клас-асоціація (association class) – це елемент моделі, який має властивості як асоціації, так і класу. Він призначений для специфікації

додаткових властивостей асоціації у формі атрибутів і операцій класу (рис. 3.6). Екземпляри класу-асоціації – це зв'язки, в яких є атрибути і операції.

Спеціальною формою зв’язку асоціації є агрегація (aggregation), яка, у свою чергу, теж має спеціальну форму – відношення композиції (composition).

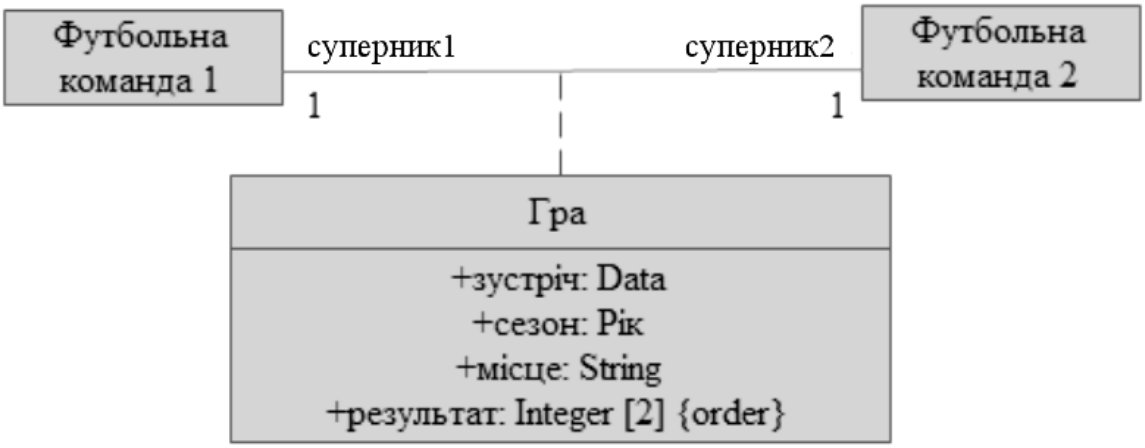

Рисунок 3.6 – Клас-асоціація

Агрегація застосовується для представлення системних взаємозв'язків типа „частини-ціле” і показує, з яких компонентів складається система і як вони зв'язані між собою. Цей зв'язок описує декомпозицію або розбиття складної системи на простіші складові частини, які також можуть бути декомпозовані, якщо в цьому виникне необхідність. Агрегації притаманні властивості транзитивності і асиметрії. Транзитивність означає, що якщо клас А містить клас В, а клас В містить клас С, то клас А містить клас С. Асиметрія означає, що якщо А містить В, то В не може містити А. Графічно агрегація зображується суцільною лінією, один з кінців якої є не зафарбований усередині ромб, що вказує на «клас-ціле».

Зв'язок композиції є різновидом агрегації. Композиція служить для виділення спеціальної форми відношення „частини-ціле”, при якому компоненти в деякому розумінні знаходяться усередині цілого. Композиція має додаткову властивість – залежність існування (existence dependency), тобто із знищенням цілого знищуються і всі його складові частини. Графічно відношення композиції зображується суцільною лінією, один з кінців якої є зафарбований усередині ромб, що вказує на «клас-ціле».

Залежність (dependency) позначає зв'язок між двома або більше елементами моделі, при якому зміна одного елементу (постачальника) може

вплинути або надати інформацію, необхідну іншому елементу (клієнтові). Відношення залежності графічно зображується пунктирною лінією між відповідними елементами зі стрілкою на одному з її кінців, при цьому стрілка направлена від класу-клієнта залежності до незалежного класу або класу-постачальника. UML 2 визначає три основні типи залежностей: використання (usage), абстракція (abstraction) і доступ (permission).

Залежність використання відображує той факт, що клієнт використовує деякі з доступних сервісів постачальника для реалізації власної поведінки.

Для моделювання залежностей між сутностями, що знаходяться на різних рівнях абстракції, використовують залежності абстракції. Як приклад можна привести клас аналітичної моделі і той же клас в проектній моделі. Існує чотири залежності абстракції: «trace», «substitute», «refine» і «derive».

Залежність «trace» (відобразити) часто використовується, аби проілюструвати відношення, в якому постачальник і клієнт представляють одне поняття, але знаходяться в різних моделях. Наприклад, постачальник і клієнт можуть знаходитися на різних стадіях розробки. Постачальник міг би бути аналітичним представленням класу, а клієнт – детальнішою проектним представленням. «Trace» можна також використовувати, аби показати відношення між функціональною вимогою, такою як «банкомат повинен забезпечувати можливість зняття готівки аж до досягнення кредитного ліміту карти», і прецедентом, що підтримує цю вимогу.

Залежність «substitute» (замістити) показує, що клієнт під час виконання може замінювати постачальника. Вона ґрунтується на спільності контрактів і інтерфейсів клієнта і постачальника, тобто вони повинні надавати один і той же набір сервісів. «Substitute» спеціально розроблена для використання в середовищах, що не підтримують спеціалізації/узагальнення.

У той час, як залежність «trace» встановлюється між елементами різних моделей, «refine» (уточнити) може використовуватися між елементами однієї і тієї ж моделі. Наприклад, в моделі може бути дві версії класу, одна з яких оптимізована за продуктивністю. Оскільки оптимізація продуктивності є різновидом уточнення, це відношення між двома класами можна змоделювати як залежність «refine» з приміткою, що описує суть уточнення.

Залежність «derive» (отримати) використовується, коли необхідно явно показати можливість здобуття однієї сутності як похідної від іншої. Наприклад, в наявному класі BankAccount є список Transaction (транзакція), в якому кожна Transaction містить Quantity (кількість) грошей. Завжди можна обчислити поточний баланс, підсумовуючи Quantity по всіх Transaction.

Для відображення можливості доступу однієї сутності до іншої використовуються залежності доступу. Існує три види залежності доступу: «access», «import» і «permit».

Залежність «Access» (доступ) дозволяє одному пакету доступ до всього відкритого вмісту іншого пакету. Проте кожний пакет визначає простір імен, і зі встановленням відношення «access» простори імен залишаються ізольованими. Елементи клієнтського пакету при зверненні до елементів пакету-постачальника повинні використовувати імена шляхів (pathnames).

Залежність «import» (імпорт) концептуально аналогічна «access», за винятком того, що простір імен постачальника об'єднується з простором імен клієнта. Це забезпечує елементам клієнта можливість організовувати доступ до елементів постачальника без необхідності вказувати в іменах елементів ім'я пакету. Проте інколи це може призводити до конфліктів імен, якщо імена елементу клієнта і елементу постачальника збігаються.

Залежність «permit» (вирішити) забезпечує можливість керованого порушення інкапсуляції, але в цілому цього відношення слід уникати. Клієнтський елемент має доступ до елементу-постачальника незалежно від оголошеної видимості останнього. Часто залежність «permit» встановлюється між двома родинними класами, коли клієнтському класу вигідно (ймовірно, по причинах продуктивності) мати доступ до закритих членів постачальника.

Узагальнення (generalization) є звичайним відношенням між загальнішим елементом (предком) і більш спеціалізованим елементом (нащадком). Це відношення описує ієрархічну побудову класів і спадкоємності їх властивостей і поведінки. При цьому передбачається, що клас-спадкоємець має всі властивості і поведінку класу-предка, а також має свої власні властивості і поведінку, які відсутні у класу-предка. На діаграмах відношення узагальнення позначається суцільною лінією з трикутною стрілкою на одному з кінців, що вказує на загальніший клас (клас-предок або суперклас).

Діаграма класів може також містити інтерфейси, пакети і шаблони. Інтерфейс визначає границю між специфікацією того, що робить абстракція, і реалізацією того, як вона це робить. Інтерфейс – це набір операцій, використовуваних для специфікації послуг класом або компонентом.

Інтерфейс класу є декларацією функціональності (оголошенням методів без реалізації їх тіл), яку повинен надавати клас, що реалізовує цей інтерфейс, і інколи – декларацією набору атрибутів, які мають бути в ньому визначені.

У загальному випадку інтерфейс графічно зображується колом, яке з'єднується з компонентом відрізком лінії без стрілок, – коротка нотація, в якій можна не показувати атрибути і операції (рис. 4.7). При цьому ім'я інтерфейсу, яке обов'язково повинне починатися із заголовної букви «I», записується поряд з колом. Крім того, інтерфейс на діаграмі компонентів може бути змальований у вигляді прямокутника класу із стереотипом «interface» і секцією підтримуваних операцій (нотація в стилі «класу», рис. 4.8). Як правило, цей варіант позначення використовується для представлення внутрішньої структури інтерфейсу.

Розрізняють два способи зв'язку інтерфейсу і класу.

– Інтерфейс, що надається (що експортується, підтримуваний, provided interface), що надається, – інтерфейс, який компонент реалізує, тобто цей компонент надає його як сервіс іншим компонентам. У нотації в стилі класу відношення реалізації представляється у вигляді пунктирної лінії з не заштрихованою стрілкою (відношення реалізації, рис. 3.8), тоді як в короткій нотації це суцільна лінія без стрілки (рис. 3.7).

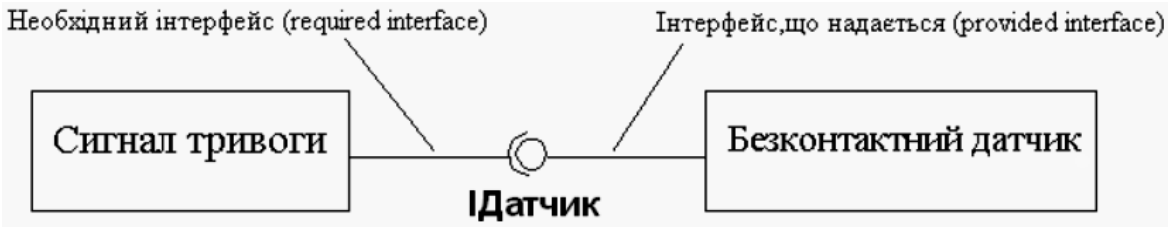

Рисунок 3.7 – Представлення інтерфейсу в кратній нотації

– Необхідний інтерфейс (що імпортується, required interface) – інтерфейс, який необхідний компоненту від свого оточення для реалізації заявленої функціональності або поведінки. Якщо ж компонент використовує деякий інтерфейс, який реалізується іншим компонентом, то такий інтерфейс для першого компонента називається інтерфейсом, що імпортується. Особливість інтерфейсу, що імпортується, полягає в тому, що на діаграмі компонентів це відношення зображується за допомогою залежності (рис. 3.8), які зображуються також лінією із стрілкою, направленою від залежного елементу (клієнта) до незалежного елементу (джерела).

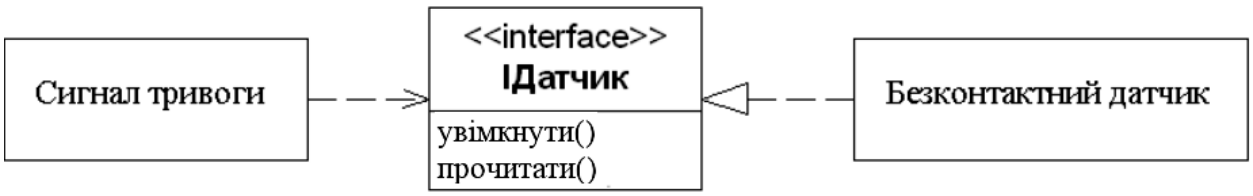

Рисунок 3.8 – Представлення інтерфейсу в нотації класу

Для представлення семантичного відношення між класифікаторами, при якому один з них описує контракт, а інший гарантує його виконання, використовується відношення реалізації (Realization). Реалізації використовується в двох ситуаціях – в контексті інтерфейсів і в контексті кооперацій. Зображується реалізація у вигляді пунктирної лінії з великою не зафарбованою стрілкою, яка вказує на класифікатор, що визначає контракт.

При моделюванні ПЗ шаблон (template) або клас, який параметризується (parameterized class), призначені для позначення такого класу, який має один (або більш) нефіксований формальний параметр. Він визначає ціле сімейство або множину класів, кожний з яких може бути отриманий пов'язанням цих параметрів з дійсними значеннями. Частіше шаблоном виступає деякий суперклас, параметри якого уточнюються в його класах-нащадках.

При аналізі ПрО ознаками правильно виділеного класу аналізу є такі:

– ім'я класу відображає його призначення;

– клас є чіткою абстракцією, яка моделює один конкретний елемент ПрО;

– в класу аналізу невеликий, визначений набір обов'язків: обов'язок – це контракт, який клас має перед своїми клієнтами. Обов'язок – семантично зв'язний набір операцій (інтерфейс);

– клас має високу внутрішню зв'язність (cohesion) – всі властивості класу служать реалізації його призначення;

– клас має низьку зв'язаність з іншими класами (coupling) – для реалізації свого призначення клас повинен взаємодіяти з невеликим числом класів.

3.3 Контрольні запитання і завдання

3.3.1 Контрольні запитання

1. На яких етапах розробляються діаграми класів?
2. Що називається відношенням асоціації, що характеризують його атрибути і кратність? Наведіть приклади і дайте опис існуючих стереотипів.
3. Охарактеризуйте відношення залежності і його стереотипи.
4. Охарактеризуйте поняття інтерфейсу. На якому з базових принципів ООП заснований механізм інтерфейсів?
5. Охарактеризуйте поняття класу асоціації.
6. Які існують підходи до виявлення класів?
7. Проаналізуйте взаємозв'язок діаграм класів (class diagram) з іншими діаграмами мови UML.

3.3.2 Завдання

Розробити діаграми класів, що розширюють моделі ПрО за допомогою опису їх логічної структури. Класи аналізу ПрО повинні точно і чітко проектуватися в об'єкти реального світу. При аналізі джерелом класів є ПрО. Прецеденти, описи вимог, глосарії і будь-яка інша інформація можуть використовуватися як джерело класів аналізу.

ПРАКТИЧНЕ ЗАНЯТТЯ 4.
РОЗШИРЕННЯ UML-МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ ЗА ДОПОМОГОЮ РОЗРОБКИ ДІАГРАМ ПОСЛІДОВНОСТЕЙ (SEQUENCE DIAGRAM)

4.1 Мета заняття

Вивчення нотації діаграми послідовностей (sequence diagram) і її використання в рамках розробки моделі Про. Розробка sequence-діаграми для Про.

4.2 Методичні вказівки з організації самостійної роботи студентів

При підготовці до практичних занять необхідно використовувати всі розроблені на попередніх практичних заняттях діаграми для вибраної ПрО.

4.2.1 Нотація sequence diagram

Діаграма послідовності – діаграма, яка служить для представлення взаємодії елементів моделі у формі послідовності повідомлень і відповідних подій на лініях життя об'єктів. *Взаємодія (interaction)* – одиниця поведінки деякого класифікатора, яка концентрує увагу на спостережуваному обміні інформацією між елементами, що є учасниками цієї взаємодії. Цей класифікатор, який називають контекстним класифікатором (context classifier), надає контекст взаємодії.

Графічна нотація представлення взаємодії – прямокутник, який також називається фреймом (frame) діаграми. У верхньому лівому кутку прямокутника фрейму зображується невеликий п'ятикутник, в який поміщається ключове слово „sd”, за яким слідує ім'я взаємодії і його параметри. Порядок настання подій уздовж ліній життя має значення для позначення послідовності, в якій ці настання події відбуваються. Проте абсолютні відстані між подіями на лініях життя не мають семантики. Іншими словами, sequence-діаграма має два виміри. Один – вертикальна часова вісь, направлена зверху вниз, де початковий момент часу розташований у верхній частині діаграми. При цьому масштаб на осі часу не вказується, оскільки діаграма послідовності моделює лише впорядкованість взаємодій в часі типа «раніше-пізніше». Всі об'єкти наносяться на діаграму з дотриманням деякого порядку ініціалізації повідомлень.

Другий вимір sequence діаграми – зліва направо у вигляді вертикальних ліній, кожна з яких змальовує лінію життя окремого об'єкту, що бере участь у взаємодії. Тут необхідно встановити, які об'єкти існуватимуть постійно, а які тимчасово – лише на період виконання ними необхідних дій.

Лінія життя (lifeline) представляє одного учасника взаємодії, тобто вона уявляє, як екземпляр конкретного класифікатора бере участь у взаємодії (рис.4.1).

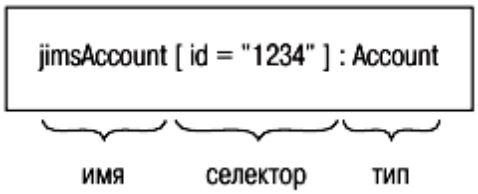

Рисунок 4.1 – Синтаксис lifeline

Кожна лінія життя має необов'язкове ім'я, тип і необов'язковий селектор. Ім'я використовується для звернення до лінії життя у взаємодії. Тип – ім'я класифікатора, екземпляр якого представляє лінія життя. Селектор – логічна умова, яка може використовуватися для вибору єдиного екземпляра, що задовольняє цій умові. Якщо селектора немає, лінія життя належить довільному екземпляру класифікатора. Селектори дійсні, лише якщо кратність типа більше одиниці, тобто існує множина екземплярів, з яких можна вибирати. Наприклад, селектор вибирає екземпляр класу Account, id якого – «1234».

Щоб взаємодія була повною, мають бути визначені повідомлення (messages), що посилаються між лініями життя. Повідомлення є особливим типом комунікації між двома лініями життя. Така взаємодія може включати:

– виклик операції – повідомлення виклику;

– створення або знищення екземпляра – повідомлення створення або знищення;

– відправку сигналу.

Повідомлення виклику, що отримується лінією життя, є запитом на виклик операції, яка має аналогічну повідомленню сигнатуру. Таким чином, для кожного повідомлення виклику, що поступає на лінію життя, в класифікаторі цієї лінії життя повинна існувати відповідна операція. Коли лінія життя посилає повідомлення, в ній знаходиться фокус управління (focus of control), або активація (activation). У міру розвитку взаємодії в часі

активація переміщається між лініями життя. Цей рух називають потоком управління (flow of control).

Таким чином, повідомлення – це елемент моделі, призначений для представлення окремої комунікації між лініями життя якоїсь взаємодії. Ім'я повідомлення має такий синтаксис:
<ідентифікатор-повідомлення>::= ([<атрибут>=] <ім'я операції або сигналу> [([<аргумент> [,<аргумент>]*)] [:<значення, що повертається>]),
де <аргумент> ::= (<[ім'я параметра>=] <значення аргументу>) | (<атрибут> =< ім'я, що повертається > [: <значення аргументу>] | *. Тут символ '*' означає нескінченність.

Сорт повідомлення (message sort) – характер комунікації, яка лежить в основі генерації повідомлення*:*

– synchCall – синхронне повідомлення, яке відповідає синхронному виклику операції. Синхронні повідомлення зазвичай представляють виклики методів і зображується суцільною лінією із закрашеною стрілкою;

– asynchCall – асинхронне повідомлення, яке відповідає асинхронному виклику операції, зображується суцільною лінією з відкритою стрілкою у формі букви "V";

– asynchSignal – асинхронний сигнал, що відповідає деякій асинхронній дії, зображується суцільною лінією з відкритою стрілкою у формі букви "V";

– відповідь (reply) на виклик методу, зображується пунктирною лінією з відкритою стрілкою у формі букви "V";

– повідомлення створення об'єкту (object creation) також зображується пунктирною лінією з відкритою стрілкою у формі букви "V".

При синхронному повідомленню відправник чекає завершення виконання одержувачем запрошуваної операції. При асинхронному відправник не чекає, а переходить до наступного етапу. Для моделей ПрО відмінність між синхронними й асинхронними повідомленнями зазвичай є високим рівнем деталізації. Зазвичай всі повідомлення відображують як синхронні.

Синхронні повідомлення показують пряму послідовність викликів операцій, тоді як асинхронні повідомлення вказують на можливий паралелізм. При проектуванні відмінність між синхронними і асинхронними повідомленнями може мати значення для створення паралельних потоків управління. На рівні аналізу реалізацій прецедентів повідомлення повернення можна показувати або не показувати – за бажанням розробника.

Зазвичай вони не мають особливого значення, і їх наносять на діаграму, лише якщо це не ускладнює її.

Повідомлення знищення об'єкту показують суцільною лінією з відкритою стрілкою і стереотипом «destroy» (знищити). Знищення означає, що екземпляр класифікатора, якому належить цільова лінія життя, більше недоступний для використання. Якщо в лінії життя є «хвіст», він повинен завершуватися великим хрестом в точці знищення. Для знищення об'єктів немає значення, що повертається.

Вигляд повідомлення (message kind):

– complete – повне повідомлення, для якого існує подія передачі і подія прийому, зображується розглянутим раніше чином залежно від сорту повідомлення.

– lost – втрачене повідомлення, для якого існує подія передачі і відсутня подія прийому, зображується у формі невеликого чорного круга на кінці стрілки повідомлення. Воно інтерпретується як повідомлення, яке ніколи не досягне свого місця призначення.

– found – знайдене повідомлення, для якого існує подія прийому і відсутня подія передачі, зображується у формі невеликого чорного круга на початковому кінці повідомлення. Воно інтерпретується як повідомлення, ініціатор якого знаходиться за межами області опису.

Зазвичай в аналізі знайдені і втрачені повідомлення можуть бути проігноровані. Приклад використання основних елементів sequence-діаграми представлений на рис. 4.2.

Витягнуті прямокутники, розташовані на пунктирній частині лінії життя, показують інтервал часу, у який на даній лінії життя знаходиться фокус управління. Повідомлення, що отримується екземпляром, може зумовити зміну його стану. Перебування екземплярів на лініях життя об'єктів можна показати за допомогою інваріантів стану (state invariants). Додавання інваріантів стану на діаграму послідовностей є методом аналізу, який дозволяє фіксувати ключові стани ЖЦ лінії життя. Вони відображають важливі стани системи і можуть бути основою для автоматів.

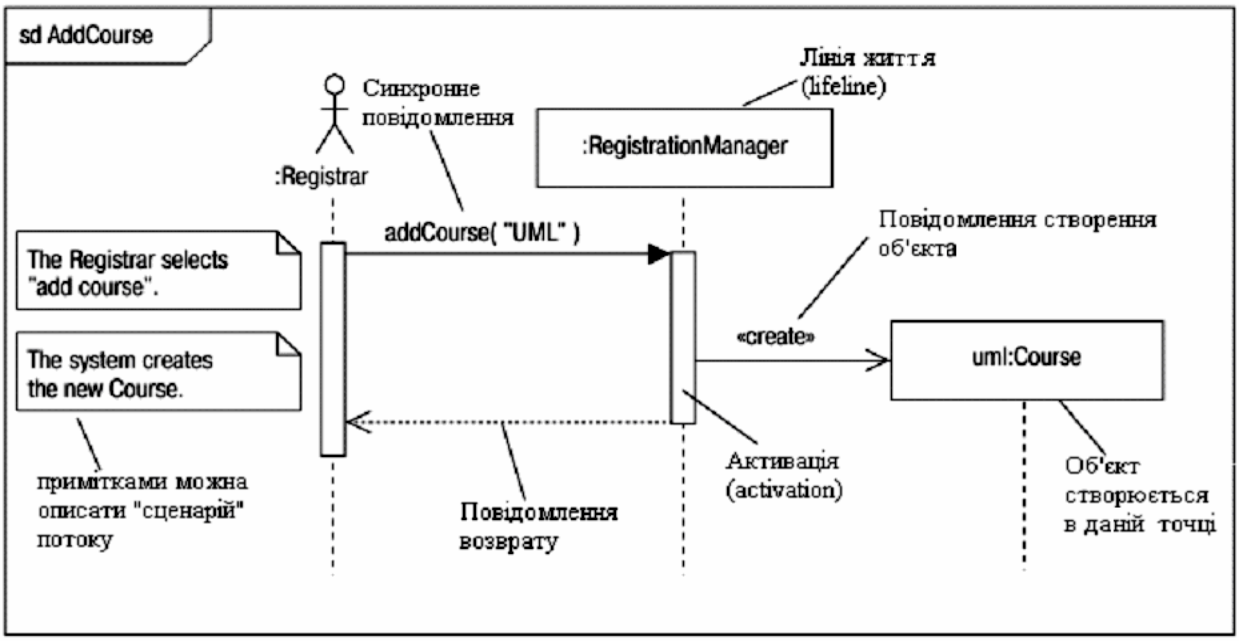

Рисунок 4.2 – Основні елементи sequence-діаграми

На лінії життя об'єкта може бути також вказана її тривалість. Тривалість (duration) специфікує відстань в часі між двома часовими виразами, які відповідають двом моментам часу. Інтервал тривалості (duration interval) визначає діапазон часу між двома подіями.

Обмеження на тривалість (duration constraint) визначає обмеження, яке відноситься до деякого інтервалу лінії життя. Приклад обмеження тривалості показаний на рис. 4.3.

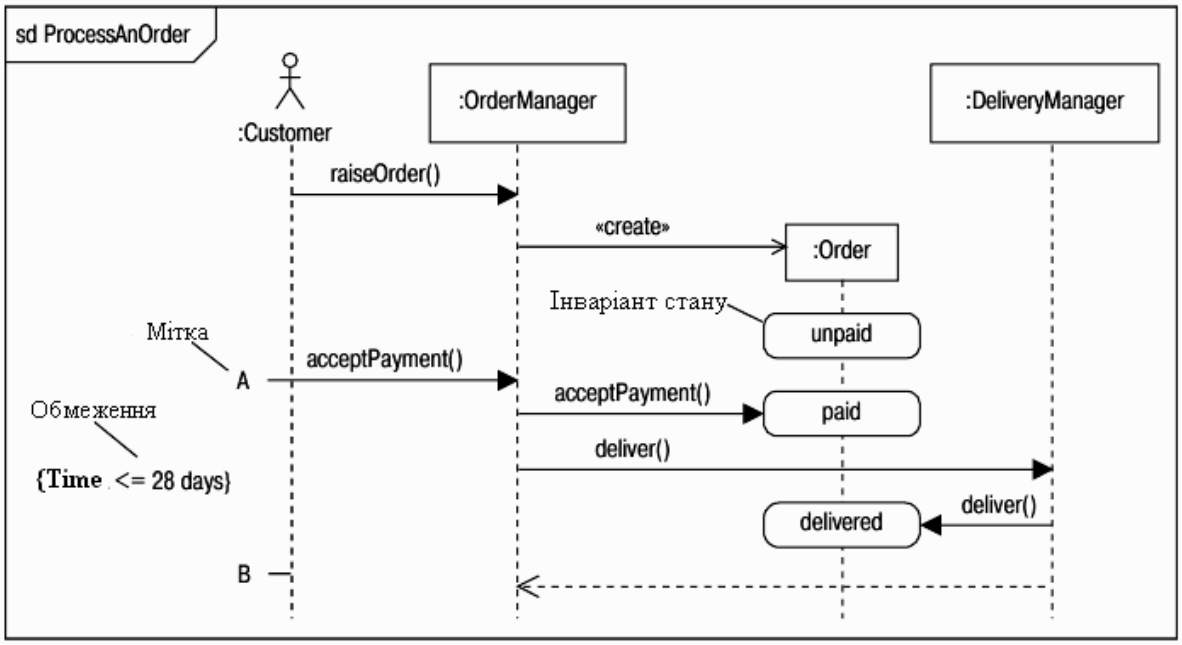

Рисунок 4.3 – Sequence-діаграма прецеденту «Обробка замовлення»

Наприклад, лінія життя об'єкта :Customer має дві мітки, A і B, і обмеження між мітками B і A {Time <= 28 days}. Це говорить про те, що інтервал часу між точками A і B не повинен перевищувати 28 днів. Обмеження часу (time constraint) є спеціальним обмеженням, записаним у формі інтервалу часу.

Sequence-діаграми можна розділити на області, звані комбінованими фрагментами (combined fragments). У кожного комбінованого фрагмента є один оператор (operator), один або декілька операндів (operands) і нуль або більш сторожових умов (guard conditions), як представлено на рис. 4.4. Сторожова умова – це логічний вираз і операнд, який виконується тоді і лише тоді, коли цей вираз вірний.

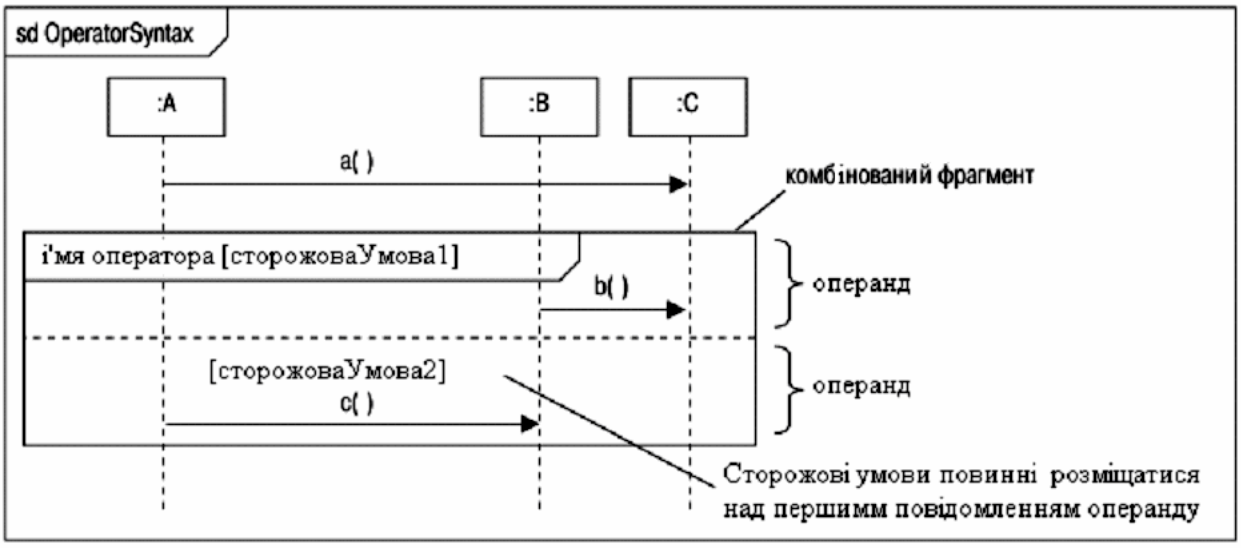

Рисунок 4.4 – Синтаксис combined fragments

У таблиці 6.1 надані існуючі оператори комбінованих фрагментів.

Таблиця 4.1 – Перелік існуючих операторів комбінованих фрагментів sequence-діаграми

№	Оператор	Повне ім'я	Семантика
1	opt	Option	Єдиний операнд виконується, якщо умова істинна (як if … then).
2	alt	alternatives	Виконується той операнд, умова якого істинна. Замість логічного виразу можна використати ключове слово else (як select... case).

Продовження таблиці 4.1

1	2	3	4
3	loop	loop	Має спеціальний синтаксис: loop min, max [condition] – повторює операнд мінімальну кількість разів, потім, поки умова істинна, повторюю ще (max – min) число разів.
4	break	break	Якщо сторожова умова вірна, виконується операнд, а не вся остання взаємодія, в яку він входить
5	ref	reference	Комбінований фрагмент посилається на іншу взаємодію.
6	par	parallel	Всі операнди виконуються паралельно.
7	critical	critical	Операнд виконується автоматично без переривання.
8	seq	weak sequencing	Всі операнди виконуються паралельно за умови виконання такого обмеження: послідовність вступу подій на одну лінію життя від різних операндів така ж, як і послідовність операндів. В результаті спостерігається слабка форма впорядкування; звідси назва (weak sequencing – слабке впорядкування).
9	strict	strict sequencing	Операнди виконуються в строгій послідовності.
10	neg	negative	Операнд демонструє невірні взаємодії. Застосовується, коли необхідно показати, що не повинно статися.
11	ignore	ignore	Перераховує повідомлення, які навмисно виключені з взаємодії. Розділений комами список імен проігнорованих повідомлень у фігурних дужках поміщається після імені оператора, наприклад {m1, m2, m3}. Наприклад, взаємодія може представляти тестовий приклад, в якому прийнято рішення ігнорувати деякі повідомлення.

Продовження таблиці 4.1

1	2	3	4
12	consider	consider	Перераховує повідомлення, навмисно включені у взаємодію. Розділений комами список імен повідомлень у фігурних дужках поміщається після імені оператора. Наприклад, взаємодія може представляти тестовий приклад, в який вирішено включити підмножину можливих повідомлень.
13	assert	assertion	Операнд є єдиною можливою допустимою поведінкою в даний момент взаємодії, будь-яка інша поведінка була б помилковою. Означає, що деяка поведінка повинна мати місце в певній точці взаємодії.

Найважливіші з них: opt, alt, loop, break, ref, par і critical. Оператори par і critical мають відношення до паралелізму, що відноситься до проектування. Інші оператори використовуються рідко.

Галуження здійснюється за допомогою операторів opt і alt (рис.4.5).

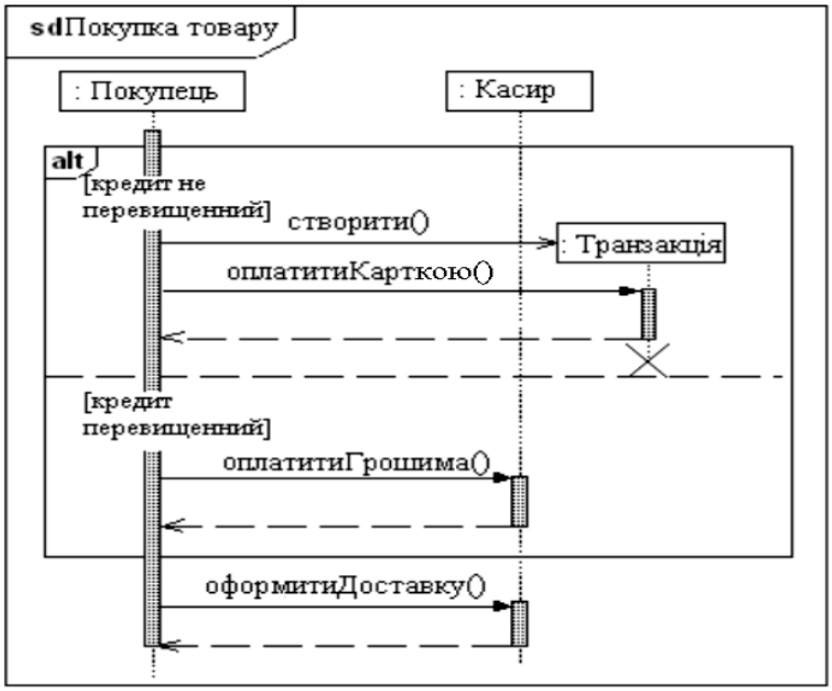

Рисунок 4.5 – Приклад використання оператора взаємодії alt

Оператор взаємодії opt специфікує необов'язковий комбінований фрагмент (option), який надає вибір поведінки, коли або виконується єдиний операнд, або жоден операнд не виконується. Необов'язковий комбінований фрагмент складається з одного операнда із сторожовою умовою. Операнд виконується, якщо виконана сторожова умова.

Оператор взаємодії alt специфікує комбінований фрагмент альтернативи (alternatives), який представляє деякий вибір поведінки. Вибраним може бути лише один з операндів. Вибраний операнд повинен мати явний або неявний вираз сторожової умови, яка в цій точці взаємодії повинна набувати значення «істина». Якщо операнд не має жодної сторожової умови, то вважається, що сторожова умова має значення «істина». Операнд, помічений сторожовою умовою [else], позначає заперечення диз'юнкції всіх інших сторожових умов цього комбінованого фрагмента.

Організація ітерацій здійснюється операторами loop і break. Оператор взаємодії loop специфікує комбінований фрагмент-цикл (loop), який є циклічним повторенням деякої послідовності повідомлень. Операнди циклу повторюються кілька разів. Додаткова сторожова умова може включати нижню і верхню межі числа повторень циклу, а також деякий логічний вираз.

Оператор циклу має такий синтаксис:

<цикл>::=loop[(<minint> [, <maxint>])]

де <minint>::= ненегативне натуральне число, яке позначає мінімальну кількість ітерацій циклу;

<maxint>::= натуральне число, яке позначає максимальну кількість ітерацій циклу. Значення <maxint> має бути більше або рівне <minint> | *. Тут символ ‘*’ означає нескінченність.

У синтаксисі loop необхідно звернути увагу на такі моменти: оператор loop без max, min або condition є нескінченним циклом; якщо задано лише min, значить, max = min; condition зазвичай є логічним виразом, але може бути і довільним текстом.

Оператор взаємодії break специфікує комбінований фрагмент «Завершення» (рис. 4.6).

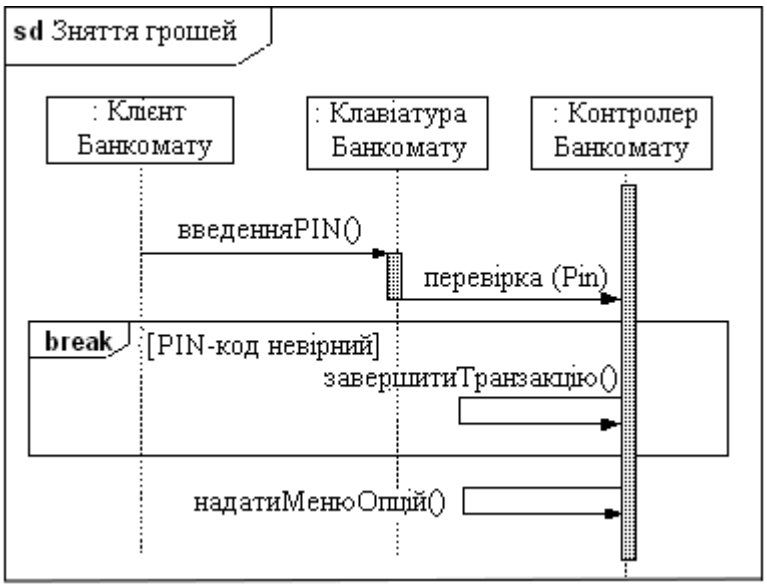

Рисунок 4.6 – Приклад використання оператора взаємодії break

Якщо ця сторожова умова набуває значення “істина”, то виконується комбінований фрагмент «Завершення», а частина фрагмента взаємодії, що залишилася, ігнорується.

Одна і та ж послідовність повідомлень може багато разів зустрічатися в різних діаграмах, а повтор одного і того ж фрагмента взаємодії приводить до появи помилок. Тому в цих випадках застосовуються включення взаємодій (використання взаємодії, interaction use). Включення взаємодій – елемент моделі, який представляє посилання, що параметризується, на деяку взаємодію в контексті іншої взаємодії. Включення означає, що в даному місці у взаємодію включається цілий потік взаємодії, на який посилається дане включення. Використання взаємодії зображується у формі фрейма комбінованого фрагмента з оператором ref, за яким слідує повне ім'я використання взаємодії.

Наприклад, фрагмент взаємодії для входу Registrar в систему матиме місце на початку багатьох діаграм послідовностей. Має сенс виділити цю загальну поведінку в окрему діаграму послідовностей і потім посилатися на неї у разі потреби (рис. 4.7).

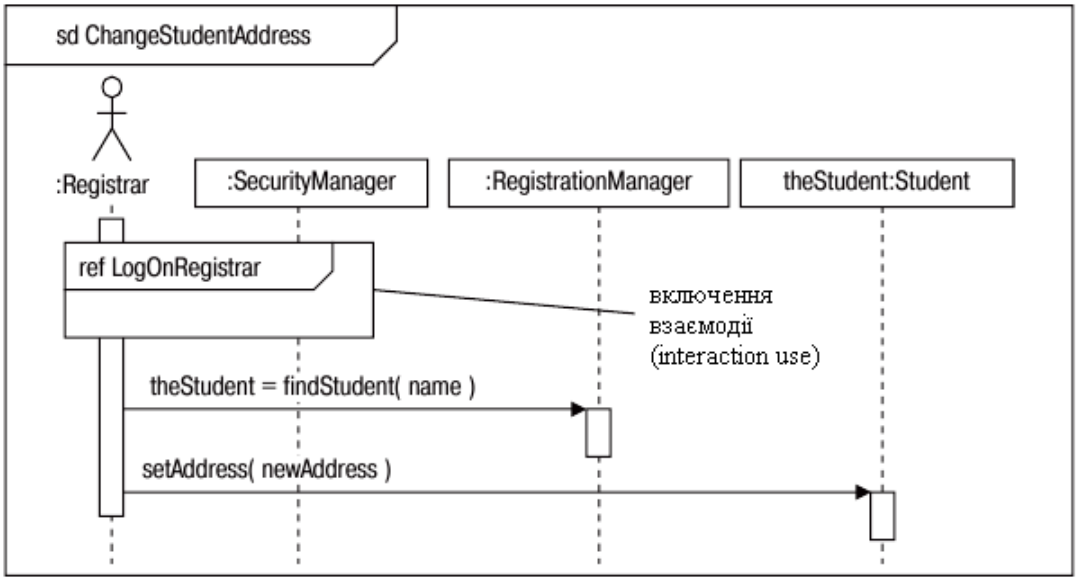

Рисунок 4.7 – Приклад використання interaction use

Приведемо декілька моментів, про які необхідно пам'ятати при використанні включень взаємодії:

– Взаємодія, на яку посилається включення, вставляється у взаємодію, що включає, в точці, де вперше з'являється включення взаємодії.

– Після закінчення взаємодії, що включається, необхідно бути уважним до того, де опиняється фокус управління. Наступне повідомлення, що відправляється у взаємодії, що включає, має бути погоджене з положенням фокусу.

– Всі лінії життя, що беруть участь у включенні, мають бути присутніми і у взаємодії, що включає.

– Щоб позначити зону дії включення взаємодії, воно повинне відображатися поверх всіх використовуваних ним ліній життя.

Взаємодії можуть мати параметри. Це забезпечує можливість передавати взаємодії різні значення в кожному випадку її включення. Параметри можуть бути задані за допомогою звичайного синтаксису операцій.

Шлюзи (gates) – це входи і виходи взаємодій. Шлюзи використовуються, коли взаємодію необхідно ініціювати лінією життя, яка не є частиною взаємодії. Шлюз – це точка на рамці діаграми послідовностей. Шлюз сполучає повідомлення, яке знаходиться поза рамкою, з повідомленням усередині рамки. Сигнатури обох повідомлень мають бути однаковими.

І шлюзи, і параметри забезпечують гнучкість в багатократному використанні взаємодій. Параметри використовуються, коли відомі вихідні і цільові лінії життя всіх повідомлень взаємодії. Шлюзи використовуються, коли деякі повідомлення приходять із-за меж рамки взаємодії і заздалегідь невідомо, звідки вони можуть поступити.

На sequence-діаграмі є можливість додавати «сценарії» шляхом розміщення приміток в нижній лівій частині діаграм. Сценарій може складатися з фактичних кроків прецеденту або короткого огляду того, що відбувається на діаграмі, представленого в текстовій формі. В будь-якому разі сценарій може бути корисним доповненням, особливо якщо діаграма складна.

4.3 Контрольні запитання і завдання

4.3.1 Контрольні запитання

1. Які діаграми використовуються в рамках етапу «Реалізація прецеденту» RUP?
2. На яких етапах різних моделей ЖЦ можливе вживання sequence-діаграм?
3. Які види повідомлень використовуються в sequence-діаграмах?
4. Які оператори використовуються в sequence-діаграмах в процесі аналізу?
5. Для чого використовується включення взаємодій (interaction use)?
6. Проаналізуйте взаємозв'язок sequence diagram з іншими діаграмами мови UML в рамках розроблених моделей
7. Які етапи в рамках структурних моделей ЖЦ не накриті об'єктними моделями?

4.3.2 Завдання

Розробити sequence-діаграм, що розширюють моделі ПрО за допомогою опису взаємодії об'єктів і класів ПрО.

ПЕРЕЛІК ПОСИЛАНЬ

1. Основы проектирования АСУ / Под. Ред. А.А. Павлова, - К.: Вища школа, 1991. – 352.
2. Смирнова Г.П., Сорокин А.А., Тельнов Ю.Т. Проектирование экономических информационных систем. - М.: ФиС, 2001. – 512 с.
3. Петров Э.Г. и др. Методология структурного системного анализа и проектирования ИУС. – Рубикон, Харьков, 1997. – 140с.
4. Маклаков С.В. BPWin и ERWin. CASE-средства разработки информационных систем. - М.: ДИАЛОГ-МИФИ, 1999. – 256 с.
5. Ларман К. Применение UML и шаблонов проектирования.: Пер. с англ.: Уч. пос. –М.: Издательский дом «Вильямс». 2001. – 496с.: ил.
6. Мацяшек Л. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML.: Пер. с англ.. –М.: Издательский дом «Вильямс». 2002. – 432с.: ил.
7. Проектирование экономических экспертных систем. - М.: ЮНИТИ, 1996. – 166 с.
8. Курицкий Б.Я. Поиск оптимальных решений. - СПб.:ВНУ, 1997. – 384 с.
9. Буч Г., Рамбо Д., Джекобсон А. Язык UML: Руководство пользователя: Пер. с англ. – М.: ДМК, 2000. – 432с.: ил.