

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ
УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
з дисципліни
“Моделювання та аналіз програмного забезпечення”
для студентів
спеціальності 6.050101
“Програмне забезпечення систем”
денної форми навчання
Частина 2

2012

Методичні вказівки до лабораторних робіт з дисципліни “Моделювання та аналіз програмного забезпечення” для студентів спеціальності 6.050101 “Програмне забезпечення систем” денної форми навчання. Частина 2/ Укл.: С.М.Сердюк, Ж.К.Камінська, Міронова Н.О. – Запоріжжя: ЗНТУ, 2012. – 78 с.

Укладачі: С. М. Сердюк, доцент, к.т.н.,
Ж.К. Камінська, асистент
Н.О. Міронова, асистент

Рецензент: С.К. Корнієнко, доцент, к.т.н.

Відповідальний
за випуск: С.М. Сердюк, доцент, к.т.н.,

Затверджено
на засіданні кафедри
“Програмних засобів”

Протокол №4
від “21 березня 2012

ЗМІСТ

ВСТУП.....	31
7 ЛАБОРАТОРНА РОБОТА №7	
Діаграми класів.....	32
7.1 Мета роботи.....	32
7.2 Основні теоретичні відомості.....	32
7.3 Завдання на лабораторну роботу.....	69
7.4 Зміст звіту.....	69
7.5 Приклади індивідуальних завдань.....	69
7.6 Контрольні запитання.....	69
8 ЛАБОРАТОРНА РОБОТА № 8	
Діаграми послідовностей.....	70
8.1 Мета роботи.....	70
8.2 Основні теоретичні відомості.....	70
8.3 Завдання на лабораторну роботу.....	84
8.4 Зміст звіту.....	87
8.5 Контрольні запитання.....	87
9 ЛАБОРАТОРНА РОБОТА № 9	
Діаграми кооперації.....	88
9.1 Мета роботи.....	88
9.2 Основні теоретичні відомості.....	88
9.3 Завдання на лабораторну роботу.....	105
9.4 Зміст звіту.....	105
9.5 Контрольні запитання.....	105
РЕКОМЕНДОВАНА ЛІТЕРАТУРА.....	105
ЧАСТИНА 3	

ВСТУП

Метою даних лабораторних робіт є опис особливостей процесу об'єктно-орієнтованого аналізу, проектування та розробки програмних додатків; визначення базових конструкцій мови UML версії 2.0 та нотації графічних елементів, які використовуються при побудові моделей програмних систем та бізнес-процесів; огляд усіх типів канонічних діаграм мови UML та практичні рекомендації по їх побудові у програмному пакеті Rational Rose версії 2003.

Мова UML (Unified Modeling Language) не залежить від об'єктно-орієнтованих мов програмування і може підтримувати будь-яку з них. Ця мова також не залежить від використаної методології розробки проекту. Діаграми, що створені на UML, виразні і зрозумілі для всіх розробників, залучених в проект, причому, що важливо, не тільки в момент розробки, а й багато місяців потому.

UML є відкритою мовою і має засоби розширення базового ядра. На UML можна змістовно описувати класи, об'єкти і компоненти в різних предметних областях, які досить часто сильно відрізняються один від одного.

Мова UML призначена для опису моделей, причому для роботи з цією мовою використовуються спеціальні редактори діаграм, такі наприклад, як Rational Rose.

Rational Rose – програмний пакет для візуального об'єктно-орієнтованого моделювання систем на основі класів та їх взаємодії. Більш спрощено, це візуальний редактор, що дозволяє моделювати програмні системи будь-якої складності на основі графічних діаграм мови UML.

У роботі №1 розглядається проектування діаграм класів. У роботі №2 розглядається проектування діаграм послідовностей. На наступному етапі, в роботі №3, розглядається проектування діаграм кооперації.

Варіанти завдань для проектування діаграм студенти отримують від викладача. Допускається розробка діаграм за темою обраною самим студентом, після узгодження предметної області з викладачем.

7 ЛАБОРАТОРНА РОБОТА № 7

Діаграми класів

7.1 Мета роботи

Навчитися проектувати діаграми класів з використанням програмного середовища Rational Rose, зокрема, створювати класи, атрибути та операції класів, накладати зв'язки між класами, генерувати коди класів на мові програмування C++.

7.2 Основні теоретичні відомості

7.2.1 Призначення діаграми класів

Class diagram (діаграма класів) основна діаграма для створення коду програми. За допомогою діаграми класів створюється внутрішня структура системи, описується успадкування та взаємне положення класів один щодо іншого. Тут описується логічне представлення системи, так як класи - це лише заготовки, на основі яких потім буде визначено фізичні об'єкти. Таким чином, діаграма класів описує загальне уявлення системи.

Зазвичай діаграма класів створюється для всіх класів системи, на відміну від діаграми об'єктів, яку проектувальники створюють для окремих об'єктів зі складною поведінкою і взаємодією.

Діаграма класів містить значки, що представляють класи, інтерфейси і їх зв'язки. Класи можуть представляти будь-які C++ класи: прості, параметризовані або метакласи. Інтерфейси - це деякий набір дій або операцій, який обслуговує взаємодію реалізацій класів.

Можливе створення однієї або декількох діаграм класів, які описують класи верхнього рівня в поточній моделі. Також можливе створення однієї (або більше) діаграм класів, які описують класи, що містяться в пакетах. Так, діаграма класів сама по собі є пакетом для класів моделі, але можна виділити додаткові пакети для логічного угруповання класів.

За допомогою діаграми класів можлива зміна в будь-який момент властивостей будь-якого класу або його зв'язків, і при цьому діаграми або специфікації, пов'язані із зміненням класом, будуть автоматично оновлені. Діаграма класів може бути використана як при аналізі готової системи, так і при розробці нової.

7.2.2 Концепція об'єкту та класу

7.2.2.1 Об'єкти

Модель класів описує статичну структуру системи: об'єкти і відносини між ними, атрибути та операції для кожного класу об'єктів. Модель класів - найважливіша з трьох основних моделей (моделі класів, моделі станів, моделі взаємодій). В основі системи повинні бути об'єкти, а не необхідна функціональність, тому що об'єктно-орієнтована система краще відповідає реальному світу і виявляється більш життєздатною при можливих змінах.

Мета моделювання класів полягає в описі об'єктів. Об'єкт (object) - це концепція, абстракція чи сутність, що володіє індивідуальністю і має сенс в рамках програми. Об'єкти часто бувають іменами власними чи конкретними посиланнями, які використовуються в описі завдань або при спілкуванні з користувачами. Деякі об'єкти існують в реальному світі (наприклад, Альберт Ейнштейн чи компанія General Electric), тоді як інші є суто концептуальними сутностями (наприклад, тестовий прогін № 1234 або формула коренів квадратного рівняння). Об'єкти третього типу додаються в модель в процесі реалізації і не мають ніякого відношення до фізичної реальності. Вибір об'єктів залежить від природи завдання і від уподобань розробника. Всі об'єкти мають індивідуальність і тому відрізняються один від одного. Індивідуальність означає, що об'єкти відрізняються один від одного внутрішньо, а не за зовнішніми властивостями. Два процесора Athlon II X2 240 2,8 GHz все одно є різними процесорами. Об'єкт є екземпляром класу.

7.2.2.2 Класи

Клас (class) описує групу об'єктів з однаковими властивостями (атрибутами), однаковою поведінкою (операціями), типами відносин і семантикою. Приклади класів: людина, компанія, процес і вікно. Кожна людина має ім'я і дату народження, а також може де-небудь працювати. Кожен процес має власника, пріоритет і список необхідних ресурсів. Класи часто бувають іменами загальними і іменними групами, які використовуються в описі завдань або при спілкуванні з користувачами.

Об'єкти одного класу мають однакові атрибути і форми поведінки. Більшість об'єктів відрізняються один від одного значеннями своїх атрибутів і відносинами з іншими об'єктами. Можливе існування різних об'єктів з однаковими значеннями атрибутів, що знаходяться в однакових відносинах з іншими об'єктами.

Операції можуть бути написані один раз для цілого класу, завдяки чому всі об'єкти класу отримують можливість повторно використовувати написаний код. Наприклад, всі еліпси мають спільні процедури промальовування, обчислення площі та перевірки на наявність перетину з деякою прямою.

Графічне зображення. Клас графічно зображується у вигляді прямокутника, який може бути розділений горизонтальними лініями на розділи або секції (рис. 7.2.1), в яких можуть вказуватися ім'я класу (рис. 7.2.1, а), атрибути (змінні) (рис. 7.2.1, б) і операції (методи) (рис. 7.2.1, в).

Ім'я класу. Ім'я класу є обов'язковим елементом позначення класу і повинно бути унікальним в рамках пакету. У кожного класу має бути ім'я (текстовий рядок), який унікально відрізняє його від усіх інших класів. При формуванні імен класів в UML допускається використання довільної комбінації букв, цифр і навіть розділових знаків. Однак на практиці рекомендується використовувати як імена класів короткі і осмислені прикметники та іменники, кожен з яких починається з великої літери.

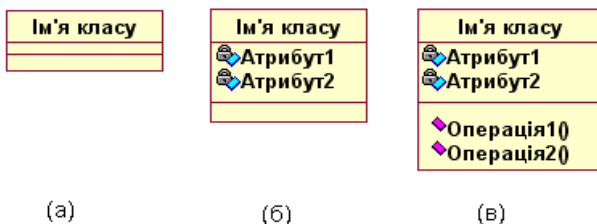


Рисунок 7.2.1 – Графічне зображення класу на діаграмі класів

7.2.2.2.3 Атрибути класу

Атрибути класу записуються у другій зверху секції прямокутника класу. Обов'язковим елементом синтаксичного

значення атрибуту є *ім'я атрибуту*. Атрибут класу може мати квантор видимості, який визначає можливість доступу до атрибуту:

- Public (відкритий доступ);
- Protected (доступ за спадкуванням);
- Private (закритий доступ – встановлюється за замовчуванням);
- Implementation (доступ за реалізацією).

Атрибут класу може мати *кратність атрибуту* - кількість атрибутів, що входить до складу певного класу. Для позначення кратності часто використовується діапазон з двох додатніх цілих чисел, перше з яких позначає нижню границю кількості атрибутів, а друге – верхню границю кількості атрибутів. Щоб вказати будь-яке додатнє ціле число, використовується спеціальний символ «*».

7.2.2.2.4 Операції класу

Операції класу – це певні функціональні можливості, які має кожен екземпляр класу. Операції класу позначаються у третій зверху секції прямокутника класу.

Операція класу може мати квантор видимості (значення аналогічні розглянутим для класів), який визначає можливість доступу до операції. На відміну від класів, для операцій за замовчуванням встановлюється значення Public.

Операція класу може містити рядок-параметр, який вивначає паралельність виконання операції та записується у вигляді «{concurrency=значення}», де значення може бути наступним:

- *Sequential* (послідовне виконання – встановлюється за замовчуванням) – необхідно, щоб одночасно у системі не виконувалися інші операції поряд з даною, бо це може призвести до помилки, тобто дана операція повинна бути єдиною виконуваною операцією у системі на даний момент;
- *Guarded* (захищене виконання) – необхідно, щоб виклики операції були впорядковані за часом для збереження цілостності об'єктів класу;
- *Synchronous* (синхронізоване виконання) – допускає можливість одночасного виконання даної операції з іншими операціями.

7.2.3 Концепція зв'язку та асоціації

7.2.3.1 Зв'язки та асоціації

Зв'язки і асоціації дозволяють встановлювати відносини між об'єктами та класами.

Зв'язок (link) - це фізичне або концептуальне з'єднання між об'єктами. У більшості випадків зв'язок з'єднує рівно два об'єкти, але бувають зв'язки, які з'єднують більшу кількість об'єктів.

Асоціація (association) - це опис групи зв'язків, що мають загальну структуру і загальну семантику. Зв'язки, які є екземплярами деякої асоціації, з'єднують об'єкти тих класів, які з'єднані між собою цією асоціацією. Асоціація описує множину потенційних зв'язків так само, як клас описує множину потенційних об'єктів. Зв'язки і асоціації зазвичай присутні в постановці завдання у вигляді дієслів.

Система графічних позначень UML вказує зображати зв'язок як лінію між двома об'єктами. Лінія може складатися з кількох прямолінійних сегментів. Якщо у зв'язку є ім'я, воно підкреслюється. Асоціація поєднує між собою класи і теж позначається лінією (яка теж може мати кілька прямолінійних сегментів). Назви зв'язків та асоціацій виділяються курсивом, а сегменти ліній прив'язуються до прямокутної сітки. Рекомендується розставляти класи так, щоб асоціація читалася зліва направо як наведено на рис. 7.2.2.

Назву асоціації вказувати не обов'язково, якщо в моделі не виникає двозначності. Неоднозначність виникає в тих випадках, коли між одними і тими ж класами існує кілька асоціацій.

Асоціації за своєю сутністю є двосторонніми. Назва бінарної асоціації зазвичай читається в конкретному напрямку, але сама асоціація може бути простежена в будь-якому напрямку.

Розробники часто реалізують асоціації у вигляді посилань з одного об'єкта на інший. Посилання (reference) - це атрибут об'єкта, який посилається на інший об'єкт. Наприклад, структура даних класу Person (Людина) може містити атрибут employer (роботодавець), який посилається на об'єкт класу Company (Компанія), а об'єкт класу Company може містити атрибут employees (співробітники), який посилається на множину об'єктів класу Person. Реалізація асоціацій у вигляді посилань цілком допустима, але моделювати асоціації таким чином не слід.



Рисунок 7.2.2 – Приклад зв'язку "Односпрямована асоціація"

7.2.3.2 Види асоціацій

Класи рідко бувають ізольовані, найчастіше вони взаємодіють один з одним. Взаємодії показуються за допомогою різного виду зв'язків. Типи зв'язків впливають на вихідний код, який генерується на основі діаграм. Використовуються наступні види зв'язків:

- Unidirectional Association (односпрямована асоціація);
- Dependency (залежність);
- Association Class (асоційований клас);
- Generalization (спадкування);
- Realization (реалізація).

Unidirectional Association (односпрямована асоціація). Це один з важливих і складних типів зв'язку, який використовується в діаграмі класів. Даний зв'язок показує, що один клас включається в інший як атрибут за посиланням або за значенням (рис.7.2.2).

Для включення в клас змінної простого типу треба використовувати завдання атрибутів, а при необхідності включення в клас змінної зі складним типом (наприклад, класа або структури), треба використовувати односпрямовану асоціацію.

Наведений далі лістинг показує код C++, який був створений для приклада зв'язку зображеного на рис.7.2.2.

```

class Class2;
class Class1
{
public:
    Class2 * theClass2;
};
  
```

Rational Rose створює код класу залежно від встановлених специфікацій зв'язку, тому розглянемо вплив різних установок специфікацій на одержуваний код. При натисканні правої кнопки миші на зв'язку активізується контекстне меню, яке надає швидкий доступ до установок зв'язку. Проте специфікації зв'язку дозволяють виконати те ж саме за допомогою вкладок діалогових вікон.

Активізуйте вікно специфікацій за допомогою контекстного меню або подвійного натискання миші на стрілці асоціації, при цьому відкривається вкладка General специфікацій (рис. 7.2.3).

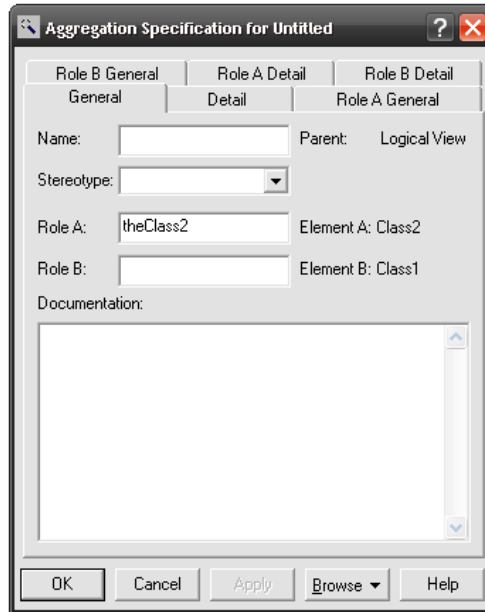


Рисунок 7.2.3 – Вкладка General специфікацій Unidirectional Association

Вкладка General - показує інформацію про ім'я, стереотипи, батьківський клас і іншу основну інформацію про зв'язок.

Name (ім'я) – задає ім'я зв'язку. Для кожного зв'язку може бути, хоча і необов'язково, задано ім'я, яке одним словом або цілою фразою вказує мету або семантику зв'язку;

Parent (батьківський) – вказує ім'я пакету, якому належить зв'язок;

Stereotype (стереотип) – вказує стереотип. Стереотип дозволяє вказувати додаткові особливості для розроблюваної моделі, які не підтримуються мовою UML. Поняття стереотипу дозволяє легше додавати інформацію в нові елементи моделі шляхом вибору стереотипу для цих елементів із вже заданих і являє собою додаткову класифікацію елементів. Деякі стереотипи вже визначені в Rational

Rose, але завжди можна додати нові стереотипи користувача, опис яких зберігається у файлі стереотипів DefaultStereotypes.ini. Стереотип може бути показаний для класу або прихований за допомогою пункту Options => Stereotype Display контекстного меню класу. Ось список деяких стереотипів, доступних в Rational Rose (рис 7.2.4):

- Actor (виконавець);
- boundary (кордон);
- Business Actor (бізнес-виконавець);
- Business Document (бізнес-документ);
- Business Entity (бізнес-сутність);
- control (управління);
- interface (інтерфейс);
- Physical Worker (робітник).

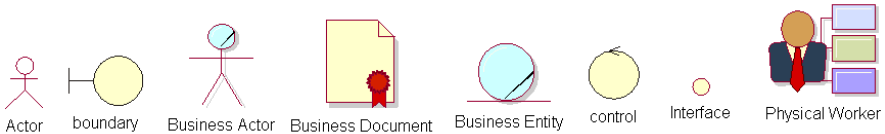


Рисунок 7.2.4 – Стереотипи класів

Role A / Role B – вказує ім'я ролі, з якою один клас асоціюється з іншим;

Element A/ Element B – вказує ім'я класу, який асоційований з даною роллю.

Вкладка Detail - вказує додаткові властивості зв'язку (рис. 7.2.5).

Link Element – вказує асоційований клас, якщо зв'язок з'єднаний з ним асоціацією;

NameDirection – вказує ім'я пов'язаного класу;

Constraints – вказує вираз деякої семантичної умови, що має бути виконана, в той час як система перебуває в стійкому стані. Це поле застосовується до зв'язку в цілому, в той час як обмеження, зазначене у цій вкладці, застосовується безпосередньо до даної ролі.

Вкладка Role General - відображає налаштування змінної, яка буде включена в клас (рис. 7.2.6).

Вкладка Role A Detail - деталізує установки для зв'язку (рис. 7.2.7). Вона має наступні поля:

Role – дозволяє задавати ім'я змінної класу;

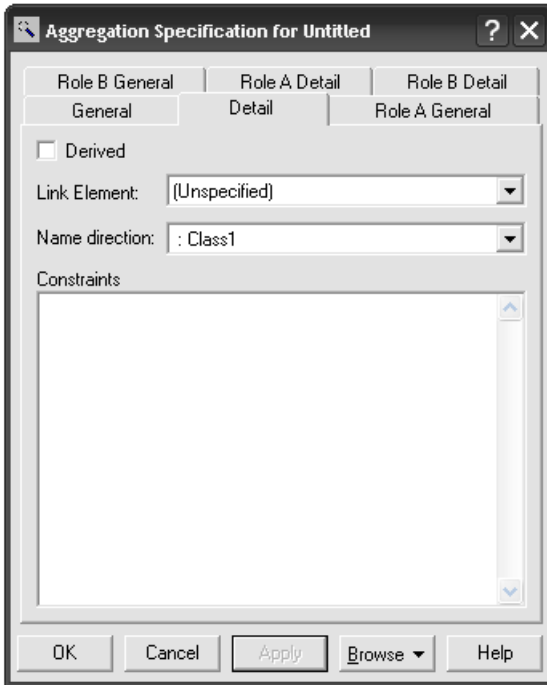


Рисунок 7.2.5 – Вкладка Detail специфікацій Unidirectional Association

Element – показує ім'я класу, для якого створюється змінна;

Constraints – вказує вираз деякої семантичної умови, що має бути виконана в той час, як система перебуває в стійкому стані. При завданні обмеження воно буде показано на діаграмі у фігурних дужках. Причому на сгенерованому коді це обмеження не буде відображатися.

Multiplicity показує, скільки об'єктів даного класу очікується створити. Може бути задано числом або буквою «n». Значення «n» вказує, що кількість нелімітовано. Можна задати різні варіанти очікуваної кількості або діапазон. Причому, вказане число відображається поруч зі стрілкою зв'язку.

Navigable показує напрямок, в якому діє зв'язок. При установці цього прапорця зв'язок набуває вигляду стрілки, що вказує напрям зв'язку. Це поле безпосередньо впливає на створюваний код класу, так як на який клас буде направлена стрілка зв'язку, той і буде включений в інший. Для того щоб змінити напрям зв'язку, досить зняти

прапорець з вкладки Role A Detail і встановити його у вкладці Role B Detail. При цьому у випадку, коли зняті прапорці на обох вкладках жоден елемент не буде включений в інший. При цьому на діаграмі буде показана просто лінія (рис. 7.2.8).

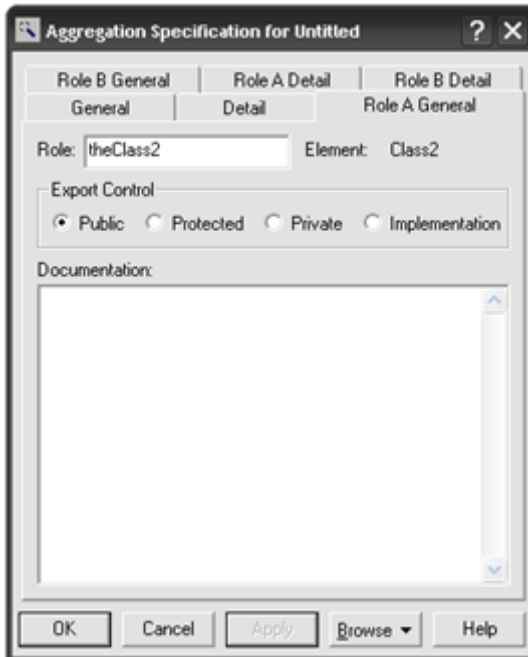


Рисунок 7.2.6 – Вкладка Role A General специфікацій Unidirectional Association

Aggregate показує, що один клас не просто використовує, а містить інший. Для того щоб показати, що клас Class2 входить до класу Class 1, необхідно встановити цей прапорець у вкладці Role B Detail. При цьому стрілка зв'язку на діаграмі набуває ромб зі зворотного боку стрілки (рис. 7.2.9).

Агрегація означає фізичне включення пов'язаного класу в інший клас. При цьому генератор коду мови C++ Rational Rose створює код, наведений на лістингу.

```
class Class2;
class Class1
{
```

```
public:
Class2 theClass2;
};
```

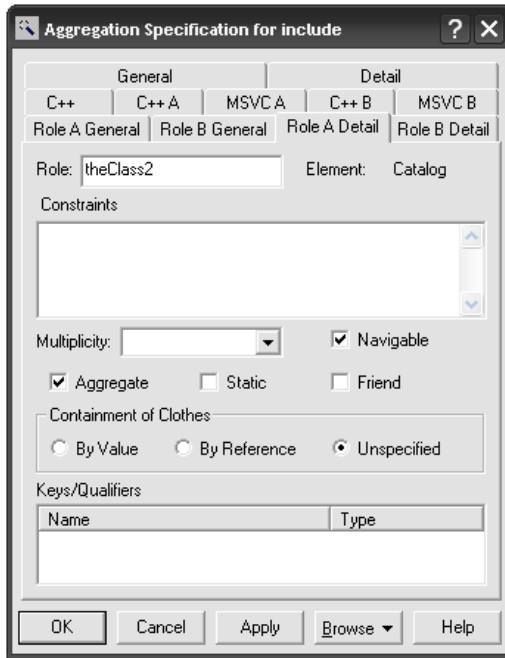


Рисунок 7.2.7 – Вкладка Role A Detail специфікацій Unidirectional Association

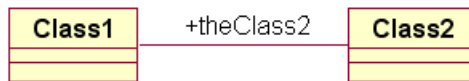


Рисунок 7.2.8 – Приклад не спрямованого зв'язку

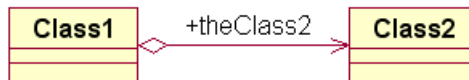


Рисунок 7.2.9 – Приклад агрегування класу

Зауваження. Агрегація може бути включена тільки для однієї вкладки або для Role A, або для Role B.

Static позначає реквізит, спільний для всіх об'єктів даного класу. Після ініціалізації до нього можна звертатися, навіть якщо ще не було створено жодного об'єкта класу. *Static* застосовується для того, щоб змінні такого типу не тиражувалися при створенні нового об'єкта класу. Наприклад, якщо необхідно знати, скільки об'єктів класу створено, можна доручити самому класу слідкувати за цим числом, створивши в ньому змінну типу `static int iCnt`, видиму в усіх об'єктах класу;

Friend визначає, що вказаний клас є дружнім класом, тобто має доступ до захищених методів і атрибутів класу.

Зауваження. Якщо прапорець встановити, а потім зняти, а між цими операціями оновити код за моделлю, то потім необхідно вручну видалити з коду класу рядок `friend`.

Key / qualifier – атрибут, який унікальним чином ідентифікує одиничний об'єкт. На генерацію коду впливу не має.

Association Class (асоційований клас). Цей тип зв'язку використовується для відображення властивості асоціації. Властивості зберігаються в класі і з'єднуються зв'язком *Association* (рис. 7.2.10). Цей тип зв'язку не має своїх специфікацій. У загальному вигляді асоціація призначена для завдання додаткових атрибутів у зв'язку. Фактично даний тип зв'язку означає, що певний клас зі своїми атрибутами включається як елемент в два інших, хоча, як не дивно, при генерації коду це не відображається.

Dependency of instances (залежність реалізації). Цей тип зв'язку дозволяє показати, що один клас використовує об'єкти іншого. Використання може здійснюватися при передачі параметрів або виклику операцій класу. У цьому випадку генератор коду C++ Rational Rose включає заголовки в клас, який використовує оператори або об'єкти іншого класу. Графічно цей вид зв'язку відображається пунктирною стрілкою (рис. 7.2.11).

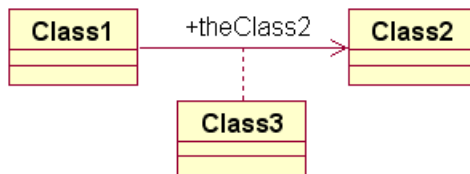


Рисунок 7.2.10 – Приклад зв'язку "Асоційований клас"



Рисунок 7.2.11 – Приклад зв'язку "Залежність реалізації"

Generalization (спадкування). Даний тип зв'язку дозволяє вказати, що один клас є батьківським по відношенню до іншого, при цьому буде створено код успадкування класу. Приклад такого зв'язку зображений на рис. 7.2.12.



Рисунок 7.2.12 – Приклад зв'язку "Спадкування"

Лістинг, отриманий при генерації коду для показаного прикладу

```
# include "Class1.h";
class Class2: public Class1
{
};
```

Realization (реалізація)

Даний тип зв'язку дозволяє показати, що один клас є реалізацією, тобто створений на основі шаблону іншого (рис. 7.2.16).



Рисунок 7.2.16 – Приклад зв'язку Realization

У C++ широко використовується поняття шаблон (template), що означає клас, на основі якого створюється ціле сімейство класів, поведінка яких визначається поведінкою шаблону, проте тип даних, з яким буде працювати створений на його основі клас, визначається параметром шаблону. У Rational Rose для позначення класу шаблону використовується поняття Parameterized Class (параметризований клас). Його можна створити за допомогою спеціального значка, який необхідно включити до рядка інструментів, або у вікні специфікацій класу на вкладці General змінити його тип на ParameterizedClass.

Графічно шаблон зображається прямокутником, до верхнього правого кута якого приєднаний маленький прямокутник з пунктирних ліній (рис. 7.2.17). У верхньому прямокутнику вказується список формальних параметрів для тих класів, які можуть бути одержані на

основі даного шаблону. Завдання формальних параметрів виконується у вікні специфікацій класу на вкладці Detail у полі Formal Arguments.

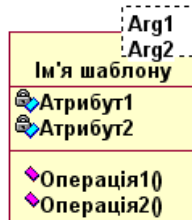


Рисунок 7.2.17 – Графічне зображення шаблону на діаграмі класів

7.2.3.3 Кратність

Кратність (multiplicity) - це кількість примірників одного класу, які можуть бути пов'язані з одним примірником іншого класу через одну асоціацію. Кратність обмежує кількість пов'язаних між собою об'єктів. У літературі найчастіше розглядаються два значення кратності: «1» і «багато», але в загальному випадку кратність може бути потенційно нескінченною підмножиною невід'ємних цілих чисел. На діаграмах UML кратність вказується явно біля кінця лінії, якою позначається асоціація. Значення кратності вказується у вигляді діапазону, наприклад «1» (рівно один), «1 ..*» (один і більше) або «3 .. 5» (від трьох до п'яти включно). Спеціальний символ «*» позначає слово «багато» (many) - нуль і більше. На рис. 7.2.18 наведено приклад, коли один примірник класу Class1 може бути пов'язаний з одним або більше примірниками класу Class2.



Рисунок 7.2.18 – Приклад використання кратності примірників різних класів

7.2.3.4 Ім'я полюсів асоціації

Кратність неявно має на увазі наявність полюсів асоціації (association end). Наприклад, асоціація один-до-багатьох має два

полюси, в одного з яких зазначена кратність «один», а у іншого - «багато». Концепція полюса асоціації - одна з найважливіших в UML. Полюс асоціації може мати не тільки кратність, але й своє власне ім'я. Імена полюсів асоціацій часто присутні в описах завдань у вигляді іменників. Ім'я полюса вказується близько до кінця асоціації. Використання імен полюсів асоціацій не є обов'язковим, але найчастіше виявляється простіше вказувати імена полюсів замість імен асоціацій або, принаймні, разом з ними (рис. 7.2.19).



Рисунок 7.2.19 – Приклад використання полюсів асоціації

Імена полюсів асоціацій особливо зручні для простежування асоціацій, тому що кожен з них може розглядатися як псевдоатрибут. Кожен полюс бінарної асоціації посиляється на об'єкт або множину об'єктів, пов'язаних з вихідним об'єктом. З точки зору вихідного об'єкта, простежування асоціації - це операція, яка повертає пов'язані з ним об'єкти. Ім'я полюса асоціації - це засіб для простежування асоціації без явної її вказівки.

Імена полюсів асоціацій обов'язкові для встановлення асоціацій між двома об'єктами одного і того ж класу. Імена полюсів асоціацій дозволяють розрізнити між собою різні асоціації між одними і тими ж класами. Якщо між парою класів існує тільки одна асоціація, імені класів може бути цілком достатньо, тому імена полюсів можна не вказувати.

Імена полюсів дозволяють уніфікувати кілька посилянь на один і той же клас. При побудові діаграм класів слід коректно використовувати імена полюсів асоціацій і не вводити окремий клас для кожного посилення.

Оскільки імена полюсів асоціації дозволяють відрізнити об'єкти один від одного, всі імена на дальньому полюсі асоціації, прикріпленої до певного класу, повинні бути унікальними. Хоча ім'я ставиться близько цільового об'єкта асоціації, фактично воно є псевдоатрибутом вихідного класу, а тому повинно бути унікальним всередині нього. З тієї ж причини ім'я полюса асоціації не має збігатися з ім'ям будь-якого атрибуту вихідного класу.

7.2.3.5 Упорядкування

Досить часто об'єкти у полюса асоціації з позначенням кратності «багато» не мають ніякого вираженого порядку. У цьому випадку їх можна розглядати як множину. Упорядкування є внутрішньою властивістю асоціації. Упорядкованість множини об'єктів можна вказати за допомогою слова {ordered}, яке ставиться близько до відповідного полюса асоціації (рис. 7.2.20).

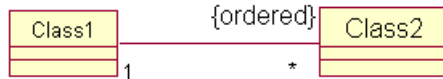


Рисунок 7.2.20 – Приклад упорядкування

7.2.3.6 Мультимножини та послідовності

Бінарна асоціація зазвичай дозволяє створити між парою об'єктів не більше одного зв'язку. Вказавши близько полюса асоціації слова {bag} або {sequence}, можна дозволити створення множини зв'язків між двома об'єктами. Мультимножина (bag) - це сукупність елементів, в якій допускається наявність дублікатів. Послідовність (sequence) – це впорядкована сукупність елементів, в якій також допускається наявність дублікатів (рис. 7.2.21). Позначення {bag} і {sequence}, як і {ordered}, застосовні лише до бінарних асоціацій.

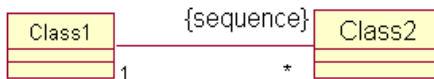


Рисунок 7.2.21 – Приклад послідовності

У UML1 не допускалося створення множини зв'язків між двома об'єктами. Деякі розробники моделей неправильно розуміли цю вимогу у разі упорядкованих полюсів асоціацій і створювали некоректні моделі, допускаючи можливість існування декількох зв'язків між двома об'єктами. У UML2, якщо біля полюса зазначено слово {bag} або {sequence}, зв'язків між двома об'єктами може бути декілька. Якщо ж цих вказівок немає, зв'язок може бути тільки один. Різниця між {ordered} і {sequence} полягає в тому, що перше позначення забороняє наявність дублікатів, а друге - дозволяє.

Послідовність - це впорядкована мультимножина, тоді як упорядкованість мається на увазі по відношенню до звичайної множини.

7.2.3.7 Кваліфіковані асоціації

Кваліфікованою називається асоціація, у якій є спеціальний атрибут (кваліфікатор), який використовується для того, щоб відрізнити об'єкти один від іншого, що знаходяться на полюсі асоціації з кратністю «багато» (рис. 7.2.22). Кваліфікація зменшує ефективну кратність асоціації до одиниці.

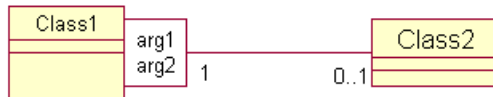


Рисунок 7.2.22 – Приклад кваліфікованої асоціації

Кваліфікатори можуть бути визначені для асоціацій типу один-до-багатьох і багато-до-багатьох. Кваліфікатор дозволяє вибрати окремих об'єктів з множини цільових об'єктів, зменшуючи таким чином ефективну кратність до значення «один». Кваліфікована асоціація з цільовою кратністю «один» або «не більше одного» утворює чіткий маршрут для пошуку цільового об'єкта по вихідному.

Обидві моделі (кваліфікована і некваліфікована) цілком коректні, але модель з кваліфікованою асоціацією повідомляє додаткову інформацію. У кваліфікованій моделі додається обмеження на кратність.

Для позначення кваліфікатора використовується невеликий прямокутник, який пристикується до вихідного класу близько кінця лінії, що позначає асоціацію. Кваліфікатор може бути пристикований до будь-якої сторони прямокутника вихідного класу. Оригінальний клас разом з кваліфікатором визначають цільовий клас.

7.2.4 Створення діаграми класів

Головна діаграма класів (Main) вже присутня у новоствореній порожній моделі, але можливе створення додаткових діаграм за допомогою контекстного меню Logical View у вікні Browse, за

допомогою пункту Browse в головному меню або за допомогою кнопки Class diagram.

7.2.4.1 Строка інструментів

При активізації діаграми рядок інструментів набуває такий вигляд (рис. 7.2.23).



Рисунок 7.2.23 – Строка інструментів для діаграми класів

Selection Tool (інструмент вибору)



Основний інструмент, який дозволяє вибирати елементи діаграми, для того щоб проводити з ними подальші дії. Якщо проектувальник не створює новий елемент, то цей інструмент активний. При створенні нового елемента діаграми необхідно вибрати потрібний інструмент в рядку інструментів, кнопка «залипає», а після створення необхідно знову перейти в режим Selection Tool.

Text Box (текст)



Даний інструмент дозволяє створити довільний напис на діаграмі, не прив'язаний до якого-небудь елементу.

Зауваження. Цей напис не є повноцінним елементом моделі і не відображається у вікні Browse, а використовується як коментар в конкретній діаграмі.

Для створення напису необхідно натиснути кнопку Text Box, при цьому курсор набуде вигляду вертикальної стрілки, і клацнути на тому місці діаграми, де необхідно створити напис. У позначеному квадратними точками в кутах вікні можна вводити напис.

Порада. Для того щоб змінити введений напис, потрібно активізувати редагування подвійним клацанням миші. Для того щоб змінити шрифт (або його розмір) використовуйте контекстне меню.

Note (зауваження)



Даний інструмент створює елемент зауваження, що дозволяє вписати в нього прийняті під час аналізу рішення. Нотатки можуть

містити простий текст, фрагменти коду або посилання на інші документи. Зазвичай вікно Note з'єднують з іншими елементами діаграми за допомогою інструменту Anchor Note, для того щоб показати до якого елемента діаграми відноситься зауваження. У цьому відмінність від елемента Text Box, який розташовується на діаграмі без приєднання до інших елементів.

Цей елемент не має обмеження на кількість символів, і вікно Note може бути розтягнуто, для того щоб вмістити необхідний текст. При активізації цього інструменту курсор приймає форму хреста при створенні елементів діаграми, які є об'єктами, і форму стрілки при створенні зв'язків. Контекстне меню для значка Note дозволяє окрім установки шрифту встановлювати колір ліній і заливки.

Note Anchor (якір для зауваження)



Даний інструмент дозволяє з'єднати елемент Note з будь-яким елементом на діаграмі, у тому числі і з іншим елементом Note.

Зауваження. Не можна з'єднати між собою два елементи Note Anchor. Для приєднання зауваження до елемента діаграми, необхідно вибрати інструмент Note Anchor, при цьому курсор набуває форми вертикальної стрілки, клацнути по значку Note і, не відпускаючи кнопки миші, «тягнути» лінію до потрібного значка, по досягненню якого кнопку миші відпустити. Аналогічно відбувається з'єднання за допомогою інших інструментів для установки зв'язків.

Class (клас)



Даний інструмент дозволяє створити новий клас у діаграмі і моделі. Поняття класу в Rational Rose аналогічно поняттю класів в C++. Клас – це установки структури і шаблону поведінки для деякої множини реальних об'єктів, які в подальшому будуть визначені в програмі на основі даного шаблону. Клас – це деяка абстракція реального світу. Коли ця абстракція приймає конкретне втілення, вона називається об'єктом.

Interface (інтерфейс)



Значок Interface дозволяє створити об'єкт Interface, який вказує на видимі ззовні операції класу або компонента. Зазвичай інтерфейс створюється тільки для певних класів або компонентів і призначений скоріше для логічного відображення системи, але може бути

присутнім як на діаграмі класів, так і на діаграмі компонентів. У діаграмі класів Interface зазвичай відображається як значок класу зі стереотипом «interface».

Unidirectional Association (односпрямована асоціація)

Значок Unidirectional Association дозволяє створити односпрямовану асоціацію класу з класом чи класу з інтерфейсом. Це загальний і найслабший вид зв'язку.

Association Class (асоційований клас)

Значок Association Class дозволяє зв'язати класи асоціативним зв'язком. Ця властивість зберігається в класі і для того, щоб її встановити, необхідно створити клас і зв'язати клас реляцією з іншим за допомогою цього значка.

Package (пакет)

Значок Package дозволяє створити елемент Package, який використовується для групування елементів. Може бути використаний для фізичного або логічного угруповання.

Dependency of instances (залежність реалізації)

Значок Dependency of instantiates дозволяє створити зв'язок Dependency of instantiates, при цьому генератор коду створює код класу, що включає визначення залежного класу шляхом генерації директиви #include. Установка цього типу зв'язків показує, що клас використовує інший клас як параметр в одному з методів.

Generalization (спадкування)

Значок Generalization дозволяє створити зв'язок Generalization, для якого Rational Rose створює код спадкування, тобто створюється підклас для сполученого цим зв'язком класу, успадкованого з батьківського класу.

Realization (реалізація)

Значок Realize дозволяє створити зв'язок Realize між класом і інтерфейсом або між компонентом і інтерфейсом. Цей тип зв'язку використовується для того, щоб показати, що клас виконує операції, що надаються інтерфейсом.

7.2.4.2 Контекстне меню класу

Після додавання класу в діаграму стає доступно контекстне меню класу. Зміст меню може змінюватися при асоціації класу з різними мовами програмування. Ознайомимось з можливостями меню для класу, не асоційованого з якою-небудь мовою програмування (рис. 7.2.24).

Перерахуємо призначення окремих пунктів:

Open Specifications – відкриття діалогового вікна заповнення специфікацій;

Sub Diagrams – дозволяє створювати для поточного класу діаграми активності і станів або перейти на піддіаграми класу;

New Attribute – дозволяє додавати новий атрибут класу;

New Operation – дозволяє додавати нову операцію для класу;

Select in Browser – дозволяє виділити клас у вікні Browser;

Relocate – дозволяє перемістити клас в новий пакет або на нове місце положення;

Options – виклик підменю налаштування значка класу;

Format – виклик підменю налаштування шрифту, кольору, заливки діаграми.

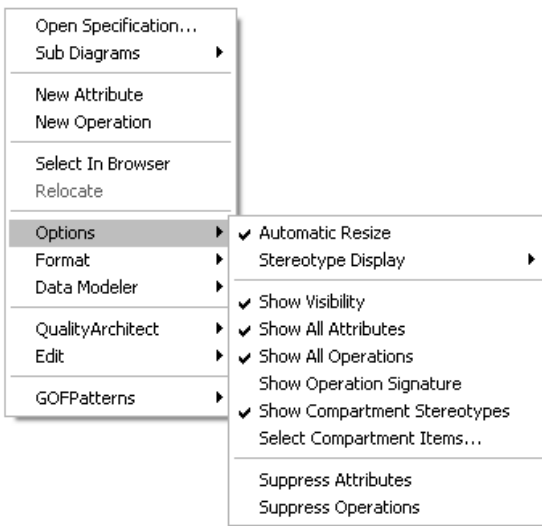


Рисунок 7.2.24 – Контекстне меню класу

Меню *Options* дозволяє керувати відображенням класу в діаграмі класів і складається з наступних пунктів:

Automatic Resize – автоматичне налаштування розміру значка, для того щоб вмістити весь введений текст назви, атрибута або операції. Дана функція зручна для початкового заповнення назв атрибутів і операцій і включена за замовчанням. Надалі, коли даний клас вже пов'язаний з іншими і займає своє місце на діаграмі класів, її можна вимкнути;

Stereotype Display – дозволяє показати або приховати стереотип для даного класу;

Show Visibility – дозволяє показати тип доступу для операторів і атрибутів, таких як *public*, *protected*, *private*, *implementation*. Причому показані ці типи доступу будуть за допомогою графічних значків;

- *Public (default)* – ;
- *Protected* – ;
- *Private* – ;
- *Implementation* – .

Show All Attributes – показує або приховує атрибути класу;

Show All Operations – показує або приховує всі операції класу;

Show Operation Signature – показує або приховує так звану сигнатуру операції, тобто параметри і значення, що повертаються;

Show Compartment Stereotypes – дозволяє показувати чи приховувати ім'я стереотипу для операції або атрибута класу;

Select Compartment items – дозволяє активізувати вікно вибору пунктів операцій або атрибутів для показу, в тому випадку якщо потрібно приховати не всі атрибути або операції, а тільки деякі. Для цього необхідно активізувати вікно *Select Compartment items* і вибрати необхідні для показу атрибути та операції (рис. 7.2.25). При цьому з лівої частини вікна, де присутні всі реквізити, необхідно перемістити в праву тільки ті, які необхідні для показу, після чого натиснути ОК.

Зуваження. Для того, щоб внесені зміни вступили в силу, необхідно зняти галочку з пунктів меню *Show All Attributes* і *Show All Operations*, і тільки після цього все, окрім обраних пунктів, буде приховано.

Suppress Attributes – дозволяє приховати всі атрибути, навіть якщо вони були обрані за допомогою вікна *Select Compartment items*. Цей пункт цікавий тим, що при його виборі не тільки приховуються

атрибути, але і закривається пункт меню *Attributes*, що не дозволяє ввести нові;

Suppress Operations – дозволяє приховати всі операції аналогічно атрибутам у попередньому пункті.

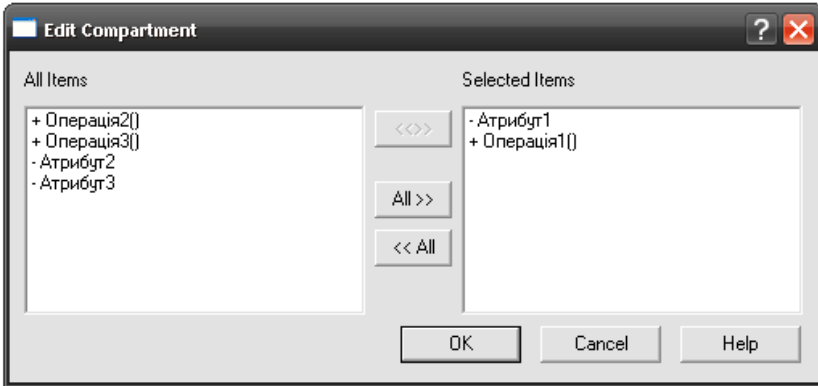


Рисунок 7.2.25 – Встановлення необхідних для показу атрибутів та операцій

7.2.4.3 Специфікації класу

Rational Rose дозволяє встановлювати значну кількість властивостей класу, які впливають на генерацію коду класу. Розглянемо вкладки вікна специфікацій.

Вкладка *General* (головна). При виборі з контекстного меню пункту *Open Specification* відкривається діалогове вікно, наведене на рис. 7.2.26. Специфікації класу мають кілька вкладок, і першою активізується вкладка *General* (головна).

Це вікно дозволяє задати основні характеристики класу, такі як його ім'я, тип, визначити стереотип класу і доступ до нього, коли клас знаходиться в пакеті. Так само як і у всіх інших діаграмах, тут можна задати документацію до класу.

Розглянемо поля, які знаходяться на цій вкладці:

Name – призначено для завдання імені класу;

Type – призначено для завдання типу класу. У нашому випадку це «клас», але може бути вибрано значення «метаклас», «параметризований клас» і т.д.;

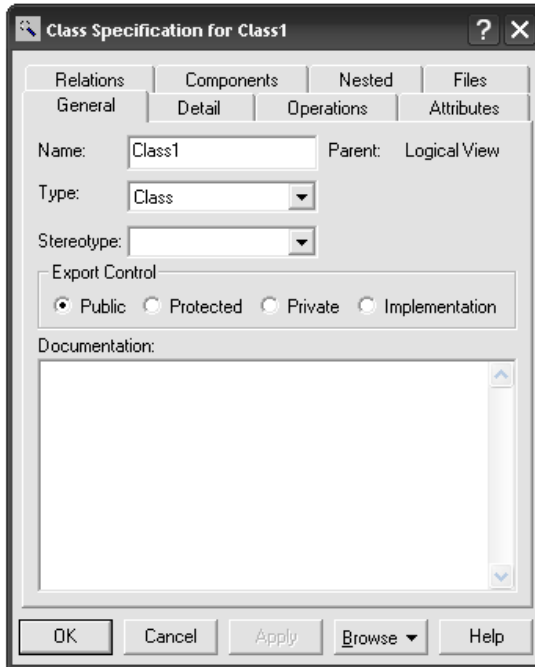


Рисунок 7.2.26 – Вкладка General специфікацій

Stereotype – задає стереотип класу;

Export Control – призначений для визначення доступу до класу, коли він розташований в пакеті. *Public* визначає, що елемент видно поза пакетом, в якому він визначений і його можна імпортувати в інші частини створюваної моделі; *Protected* елемент доступний тільки для вкладених класів, класів з типом *friends* і власне всередині класу; *Private* позначає захищений елемент класу; *Implementation* елемент видно тільки в тому пакеті, в якому він визначений.

Вкладка Detail (деталізація) - дозволяє вказувати додаткові установки класу, такі як очікувана кількість створюваних об'єктів класу, очікувані витрати оперативної пам'яті і т.д. (рис. 7.2.27).

Розглянемо поля, які знаходяться на цій вкладці:

Multiplicity (множинність) – дозволяє задати очікувану кількість об'єктів, які будуть створені на основі даного класу. Зазвичай цей параметр зручно задавати для пов'язаних класів;

Space – показує кількість оперативної пам'яті, необхідної для

створення об'єкта даного класу. Поле може бути задано безпосередньо або формулою, яка описує вимоги по пам'яті, і значення має враховувати накладні витрати на створення об'єкта плюс розмір усіх об'єктів, що входять в даний;

Persistence – визначає час життя об'єкта класу. Встановлення прапорця *persistent* означає, що об'єкт буде доступний протягом всієї роботи програми або для доступний іншим потокам або процесам.

Зауваження. Установка *persistence* для класу і для об'єкта повинна бути однаковою. Дана установка не діє для утиліт класів, параметризованих утиліт класів і реалізацій утиліт класів.

Concurrency – позначає поведінку елемента у многопоточковому середовищі. Установка такого поля в операції не повинна суперечити установці в самому класі. Дана установка може приймати наступні варіанти:

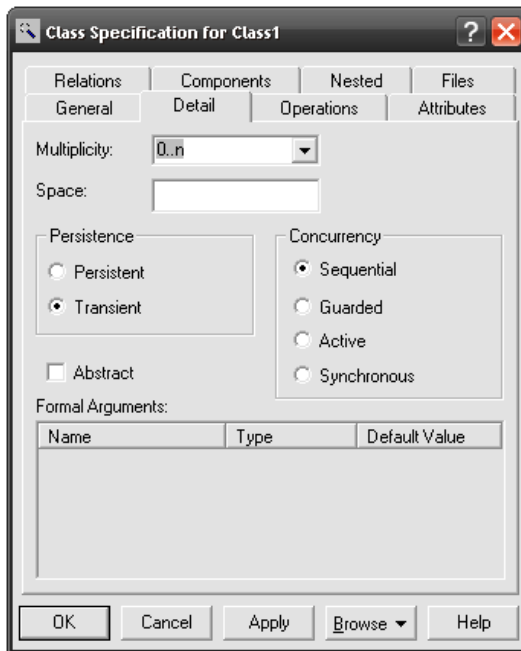


Рисунок 7.2.27 – Вкладка Detail специфікацій

➤ *Sequential* (за замовчуванням) – робота класу забезпечується тільки для одного потоку. Тільки один потік може бути

запущений за допомогою методів класу в один момент часу;

➤ *Guarded* – клас забезпечує роботу з декількома потоками. Такий клас забезпечує взаємодію між потоками клієнтів для досягнення несуперечливої роботи потоків, є арбітром потоків, надаючи роботу конкретному потоку в конкретний момент часу;

➤ *Active* – клас, є класом окремого потоку;

➤ *Synchronous* – клас забезпечує роботу декількох потоків, синхронізуючи їх.

Abstract adornment – означає, що клас є абстрактним, тобто базовим класом, який повинен бути успадкованим підкласами і без такого наслідування не має сенсу. Також в класі можуть бути визначені абстрактні операції, які являють собою шаблон для створення операцій в класах, успадкованих з цього, й самі не виконують ніяких дій.

На основі абстрактного класу не можна створювати об'єкти, але можна створювати підкласи. На діаграмі класів назва абстрактного класу представляється курсивом.

Formal Arguments – заповнюється тільки для параметризованих класів і утиліт класів. Для звичайних класів дане поле недоступно.

Вкладка Components (компоненти) - відображає компоненти, з якими асоційований клас (рис. 7.2.28).

Вона схожа на вкладку *Realized* в специфікаціях компонента. Тут можлива установка зв'язку поточного класу і компонента, а в *Realize* установка зв'язку поточного компонента і класу. На вкладці червоним маркером позначені компоненти, які включені в поточну модель і можуть бути показані інші компоненти моделі. Якщо у вас не показаний компонент MFC 6.0, то необхідно увімкнути прапорець *Show All Components* або ще не була імпортована бібліотека MFC.

Вкладка Attributes (атрибути) - дозволяє додавати, видаляти, редагувати атрибути класу (рис. 7.2.29).

На вкладці наведено список атрибутів класу, який можна редагувати за допомогою контекстного меню. Прапорець *Show inherited* дозволяє приховати або показати доступні атрибути батьківських класів.

Для того щоб додати атрибут, необхідно з контекстного меню вибрати пункт *Insert*. По подвійному натисненню миші на атрибуті або з контекстного меню *Rational Rose* надає доступ до діалогового вікна специфікацій атрибутів (рис. 7.2.30).

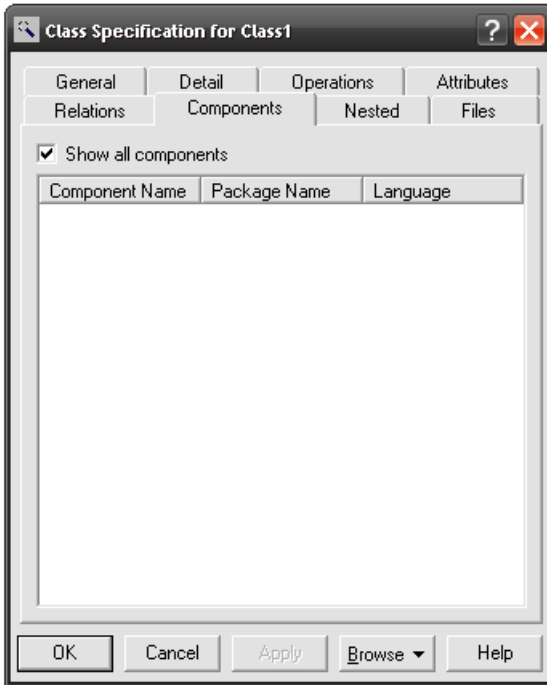


Рисунок 7.2.28 – Вкладка Components специфікацій

Тут ви можете змінити назву атрибута (Name), його тип (Type) і стереотип (Stereotype), задати початкове значення (Initial value) і тип доступу до атрибуту (Export Control).

Додаткова вкладка Detail специфікацій атрибутів класу (рис. 7.2.31) дозволяє задати тип зберігання атрибута в класі:

- By Value – за значенням;
- By Reference – за посиланням;
- Unspecified – не вказано.

Також користувач може вказати, що атрибут є *Static* (статичним) або *Derived* (похідним).

Вкладка Operations (операції) - дозволяє додавати, видаляти, редагувати операції класу (рис. 7.2.32).

На цій вкладці наведено список операцій класу, який можна редагувати за допомогою контекстного меню. Для того щоб додати операцію, необхідно з контекстного меню вибрати пункт *Insert*.

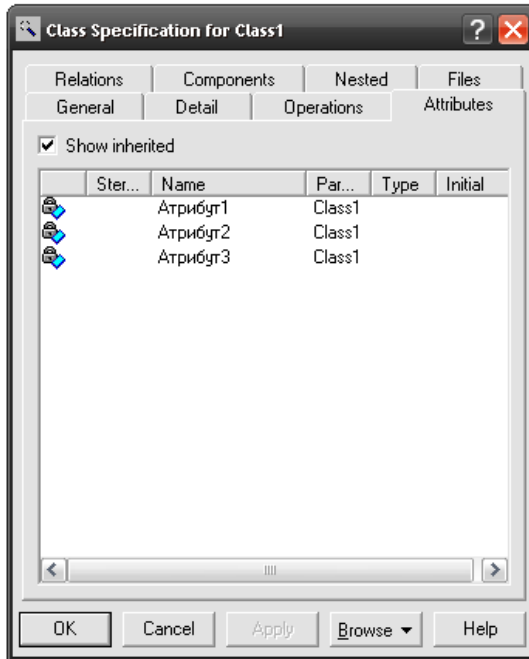


Рисунок 7.2.29 – Вкладка Attributes специфікацій

По подвійному натисненню миші на операції або з контекстного меню Rational Rose надає доступ до діалогового вікна специфікацій операції. Вкладка *General* специфікацій операції аналогічна вкладці *General* атрибутів.

Вкладка *Detail* специфікацій операцій дозволяє встановлювати додаткові властивості операції на цій вкладці:

Arguments (аргументи) – дозволяє встановлювати список аргументів для операції з їх типами і значеннями за умовчанням;

Protocol (протокол) – дозволяє задавати список установок, який надається клієнту для виклику;

Qualification (кваліфікація) – дозволяє ідентифікувати залежні від мови можливості, які дозволяють кваліфікувати метод. Синтаксис запису наступний: <ім'я операції>.Qualification. Типом властивості Qualification буде тип string;

Exceptions (виключення) – дозволяє задавати список винятків, які можуть бути викликані операцією. Тут необхідно ввести ім'я

одного або декількох класів, що обробляють виняткові стани;

Size (розмір) – дозволяє задати розмір пам'яті, необхідної для виконання операції;

Time (час) – дозволяє задати час виконання операції;

Concurrency (конкуренція) – відображає для многопоточної програми тип виконання операції:

- *Sequential* (default) – тільки один потік повинен виконуватися в один і той же час;
- *Guarded* – можливе виконання декількох потоків, керованих деяким класом;
- *Synchronous* – операції можуть йти паралельно.

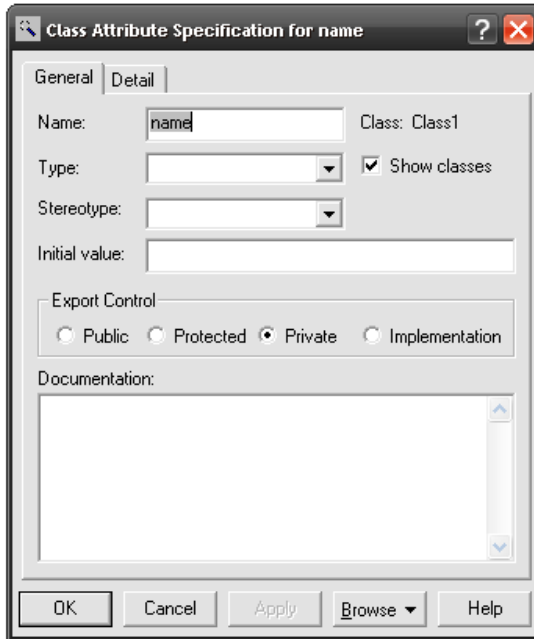


Рисунок 7.2.30 – Вкладка General специфікацій атрибутів класу

Користувач може встановлювати конкуренцію виконання для класу в специфікаціях класу. Це поле недоступно для редагування утиліт класів, параметризованих класів і реалізацій утиліт класів. Вкладки *Preconditions*, *Postconditions*, *Semantics* дозволяють задавати додаткові описи процесів підготовки і завершення операції, а також

опис алгоритму операції. Крім текстового опису тут же можна задати ім'я *Interaction* діаграми, яка описує зазначені дії.

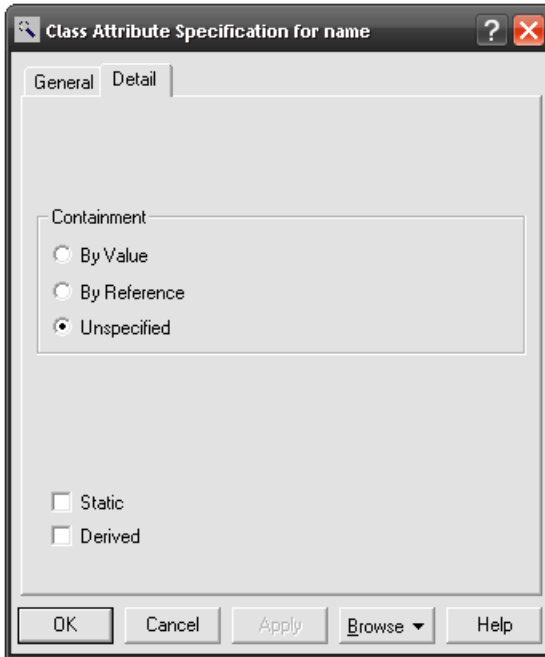


Рисунок 7.2.31 – Вкладка Detail специфікацій атрибутів класу

Вкладка Relations (зв'язки) - дозволяє додавати, видаляти, редагувати зв'язку класу (рис. 7.2.33). На вкладці наведено список зв'язків класу, який можна редагувати за допомогою контекстного меню. Щоб додати зв'язок треба скористатися відповідним інструментом з рядка інструментів, а для видалення – контекстним меню.

Зауваження. Після видалення зв'язків з діаграми повністю видалити їх можна тільки за допомогою вкладки Relations.

7.2.5 Приклад виконання

Необхідно розробити діаграму класів інтернет-магазину одягу. Необхідно змодельовати роботу користувачів інтернет-магазину з каталогом одягу. До користувачів інтернет-магазину будуть

відноситися: потенційні покупці та адміністратор інтернет-магазину. Каталог одягу інтернет-магазину буде містити інформацію про одяг.

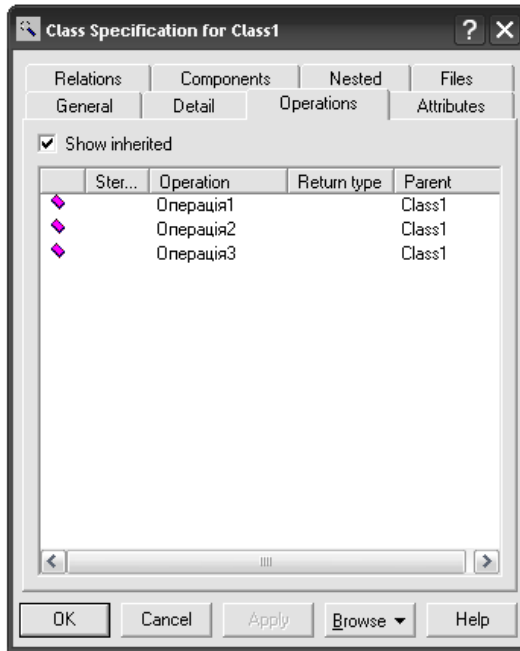


Рисунок 7.2.32 – Вкладка Operations специфікації

Рекомендації. Для спрощення моделювання необхідно створити клас, який буде містити загальну спільну інформацію класів потенційних покупців та адміністратора.

Реалізація.

1. Спочатку треба створити діаграму класів. В браузері необхідно виконати наступну послідовність дій: Logical view => New => Class Diagram (рис 7.2.34).

2. На створеній діаграмі класів необхідно розмістити класи за допомогою значка  на панелі інструментів. На діаграмі будуть знаходитися наступні класи: «Clothes» («Одяг»), «Catalog» («Каталог»), «User» («Користувач»), «Customer» («Покупець»), «Administartor» («Адміністратор») та «Person» («Людина»). Розташування класів на діаграмі зображено на рис. 7.2.35.

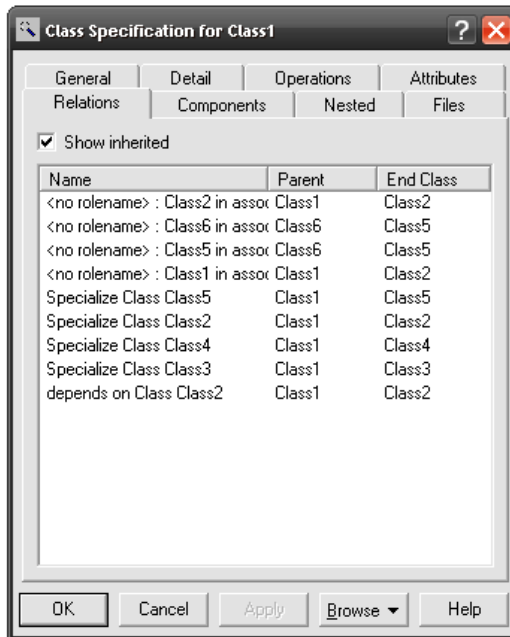


Рисунок 7.2.33 – Вкладка Relations специфікації

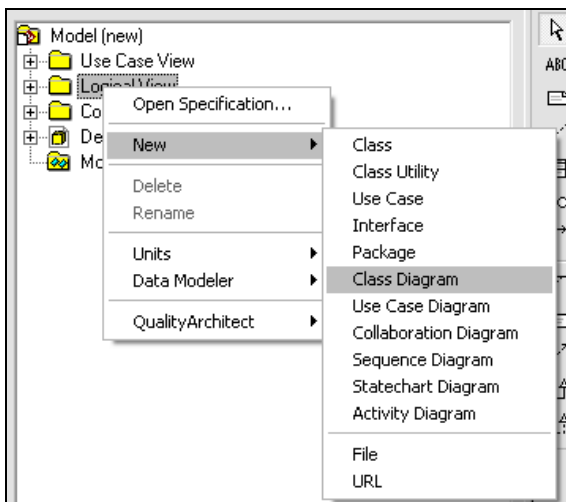


Рисунок 7.2.34 – Створення діаграми класів

Рекомендації. Ім'я класів, атрибутів та операцій краще записувати латиницею для спрощення в подальшому генерації коду класу.

3. Кожному класу необхідно присвоїти атрибути. Для створення атрибутів необхідно клікнути правою кнопкою миші по класу та у контекстному меню обрати опцію *New Attribute*. Потім ввести необхідне ім'я атрибуту (рис. 7.2.36). Для редагування інформації про атрибут необхідно клікнути правою кнопкою миші по класу та у контекстному меню обрати опцію *Open Specification...* Далі перейти на вкладку *Attributes* та клікнути правою кнопкою миші по необхідному атрибуту. У контекстному меню необхідно обрати опцію *Specification...* . На вкладці *General* необхідно проставити тип атрибуту у полі *Type*.

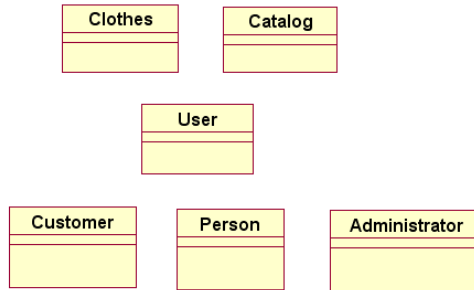


Рисунок 7.2.35 – Класи на діаграмі

4. Кожному класу необхідно присвоїти операції. Для створення операцій необхідно клікнути правою кнопкою миші по класу та у контекстному меню обрати опцію *New Operation* і ввести необхідне ім'я операції (рис. 7.2.37).

Для редагування інформації про операції необхідно клікнути правою кнопкою миші по класу та у контекстному меню обрати опцію *Open Specification...* Далі перейти на вкладку *Operations* та клікнути правою кнопкою миші по необхідній операції. У контекстному меню необхідно обрати опцію *Specification...* . На вкладці *General* необхідно проставити тип повертаємого значення операції у полі *Return Type* та доступ до методу у полі *Export Control*.

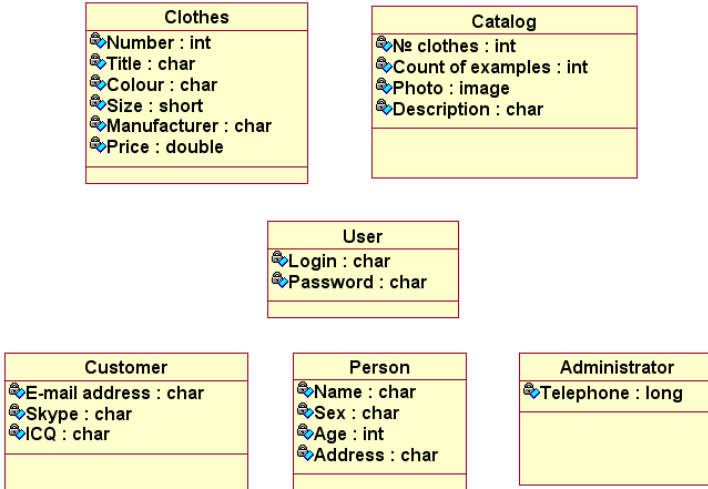


Рисунок 7.2.36 – Класи з атрибутами

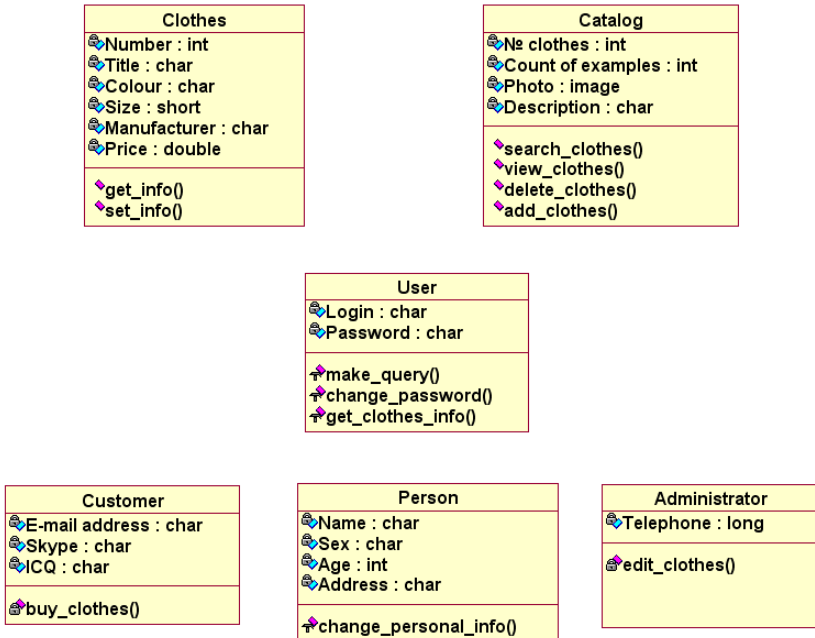


Рисунок 7.2.37 – Класи з атрибутами та операціями

5. Між усіма класами необхідно розмістити зв'язки (рис. 7.2.38). «Clothes» входить до «Catalog», але може існувати і без нього, тому їх зв'яже відношення агрегації. Для цього необхідно розмістити зв'язок асоціації за допомогою інструмента  між класами «Clothes» та «Catalog», у його властивостях на вкладці *Role A Detail* зняти позначку «Navigable» та відмітити «Aggregate». На вкладці *General* вказати ім'я зв'язку «include». Далі зробимо існуючу асоціацію кваліфікованою, щоб відрізнити об'єкти класу «Clothes» один від одного. Для цього на вкладці *Role B Detail* у полі *Keys/Qualifiers* натискаємо правою кнопкою миші та вводимо ім'я кваліфікатора – «clothesID». На цій же вкладці у полі *Multiplicity* виставляємо значення «0...1». На вкладці *Role A Detail* у полі *Multiplicity* виставляємо значення «1».

Класи «User» і «Catalog» є незалежними, тому їх необхідно зв'язати відношенням асоціації. Оскільки конкретний запит до каталогу може бути зроблений лише одним користувачем, а сам користувач може зробити від 1 до декількох запитів до каталогу, тому кратність відношення «User»-«Catalog» 1 – 1..N. Для цього необхідно розмістити зв'язок асоціації за допомогою інструмента  між класами «User» та «Catalog», на вкладці *Role A Detail* зняти відмітку «Navigable» та вибрати «Multiplicity» 1..n. На вкладці *Role B Detail* вибрати «Multiplicity» 1. На вкладці *General* вказати ім'я зв'язку «make queries». Для того, щоб упорядкувати всі запити до каталогу одягу створимо упорядкування запитів додаванням рядка-параметра {ordered} до кінця асоціації з боку класу «Catalog».

Класи «Customer» та «Administartor» є потомками класів «User» та «Person», тому вони зв'язані відношенням узагальнення, тобто ми маємо множинне спадкування класів «Customer» та «Administartor» від несумісних класів «User» та «Person». Для зображення множинного спадкування від несумісних класів необхідно розмістити зв'язок узагальнення за допомогою інструмента  між класами «Customer» та «User», «Administrator» та «User», «Customer» та «Person», «Administrator» та «Person» (від класу-потомка до класу-предка).

6. Необхідно сгенерувати коди класів на C++. На основі діаграми класів Rational Rose дозволяє створювати код класу на обраній мові. Для того, щоб скористатися даною можливістю, необхідно переконатися, що вибрана мова програмування встановлена

за допомогою Menu: Add-Ins менеджера (Add-Ins => Add-In Manager... => Rose C++ (встановити галочку).

Перед початком генерації коду необхідно перевірити виконання наступних умов на діаграмі класів:

- всі імена класів, атрибутів, операцій повинні бути записані латиницею;
- всі атрибути повинні мати визначений тип;
- всі операції повинні мати визначений тип повертаємого значення.

Далі Menu: Tools => C++ => Code Generation. З'явиться діалогове вікно, показане на рис. 7.2.39.

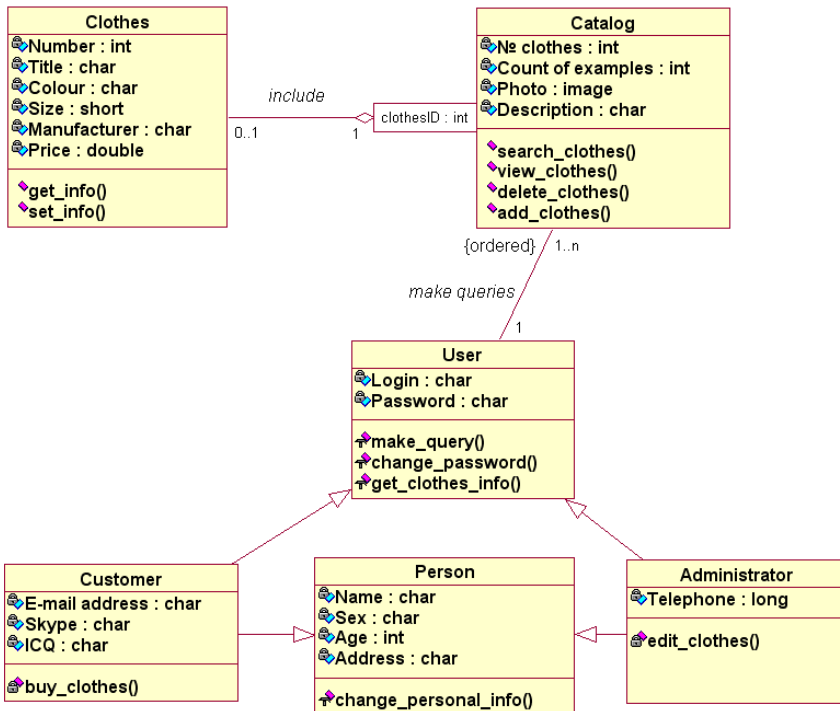


Рисунок 7.2.38 – Діаграма класів і зв'язки між класами

Після генерації коду в контекстному меню стане доступний додатковий пункт C++, в якому можна переглянути заголовки (Header).h і файл тіла класу (Body).cpp, як показано на рис. 7.2.40.

Тепер можна переглядати файли тіла класу і заголовний файл після кожної внесеної зміни, щоб перевірити, як ці зміни позначаться на сгенерованому коді.

Також можна переглядати коди створених файлів безпосередньо, зайшовши у директорію ...\\Rational\\Rose\\C++\\source\\.

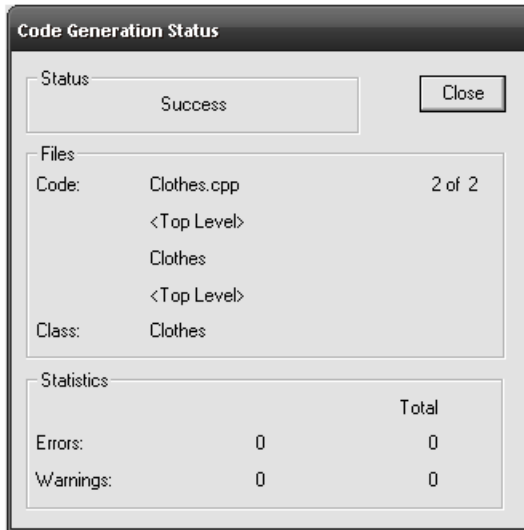


Рисунок 7.2.39 – Діалогове вікно після генерації коду

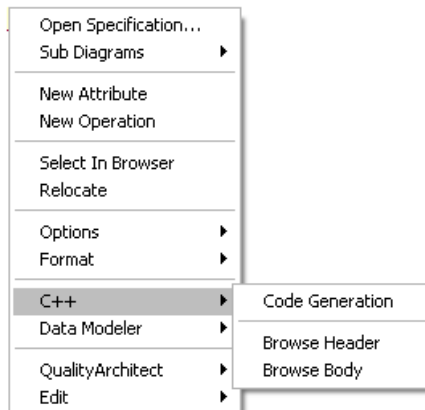


Рисунок 7.2.40 – Доступне меню C++

7.3 Завдання на лабораторну роботу

7.3.1 Ознайомитися з пунктом 1.2 і конспектом лекцій.

7.3.2 Обрати та узгодити з викладачем тему для створення діаграми класів.

7.3.3 Побудувати діаграму класів, зв'язки між класами (зокрема, створити множинне спадкування між класами та визначити кваліфіковані асоціації), занести до кожного класу атрибути та операції, використати можливість упорядкування об'єктів, сгенерувати код класів на мові C++.

7.3.4 Оформити звіт по роботі.

1.3.5 Відповісти на контрольні питання.

7.4 Зміст звіту

7.4.1 Тема та мета роботи.

7.4.2 Завдання до роботи.

7.4.3 Побудована діаграма класів та сгенеровані коди класів на C++.

7.4.4 Висновки, за результатами виконання роботи.

7.5 Приклади індивідуальних завдань

1. Інформаційна система банківського відділу.
2. Інформаційна система платіжного терміналу.
3. Інформаційна система відділу кадрів ВНЗ (вищого навчального закладу).
4. Інформаційна система ресторану.
5. Інформаційна система інтернет-магазину.
6. Інформаційна система проектування складних програмних додатків.
7. Інформаційна система автомобільного салону.
8. Інформаційна система АЗС.
9. Інформаційна система гідропонної установки.
10. Інформаційна система реєстрації поліклініки.

7.6 Контрольні запитання

7.5.1 Яке призначення діаграми класів?

7.5.2 З яких секцій складається визначення класу?

7.5.3 Що таке зв'язки та асоціації?

7.5.4 Які види зв'язків доступні на діаграмі класів?

7.5.5 Для чого використовується зв'язок Association Class?

7.5.6 Яке призначення зв'язків Dependency та Generalization?

7.5.7 Яке призначення зв'язку Realization?

7.5.8 Що таке мультимножини та послідовності?

7.5.9 Що таке Parameterized Class та як звичайний клас змінити на параметризований?

7.5.10 Що таке множинне спадкування та які можливі види обмежень при множинному спадкуванні?

8 ЛАБОРАТОРНА РОБОТА № 8

Діаграми послідовностей

8.1 Мета роботи

Навчитися проектувати діаграми послідовностей з використанням програмного середовища Rational Rose, зокрема, створювати об'єкти, налаштовувати час життя об'єктів, створювати повідомлення між об'єктами та встановлювати властивості повідомлень.

8.2 Основні теоретичні відомості

8.2.1 Призначення діаграми послідовностей

Протягом роботи складної системи об'єкти, що є клієнтами, посилають один одному різні повідомлення, а об'єкти, що є серверами, обробляють їх. У простому випадку можна розглядати повідомлення як виклик методу будь-якого класу, в складніших випадках сервер має обробник черги повідомлень, і повідомлення їм обробляються асинхронно, тобто сервер накопичує кілька повідомлень в черзі, якщо не може обробити їх відразу.

Обмін повідомленнями відбувається в певній послідовності, і діаграми послідовності (*Sequence diagram*) дозволяють отримати відбиток цього обміну у часі. Необхідно точно представляти взаємодію об'єктів між собою, визначення клієнтів і серверів та порядку обміну повідомлень між ними.

На основі прийому-передачі повідомлень заснована багатозадачність Windows. Для простоти демонстрації створення програми будемо вважати, що повідомлення обробляються негайно в тій послідовності, в якій вони видаються клієнтами.

8.2.2 Об'єкти

Діаграма послідовності показує учасників взаємодії системи з дійовими особами. Кожна дійова особа, як і система, зображується прямокутником з вертикальною лінією, яка називається лінією життя, а кожне повідомлення – горизонтальною стрілкою, спрямованою від відправника до одержувача. Час спрямований зверху вниз, однак масштаб на діаграмі не дотримується. Діаграма показує тільки послідовність повідомлень, а не точний їх порядок у часі. Для зображення об'єкта використовується прямокутник, який знаходиться у верхній частині своєї лінії життя (рис. 8.1). У прямокутнику записується ім'я об'єкта та класу в наступному вигляді: <ім'я об'єкта>: <ім'я класу>. Підкреслення всього запису є ознакою об'єкта.

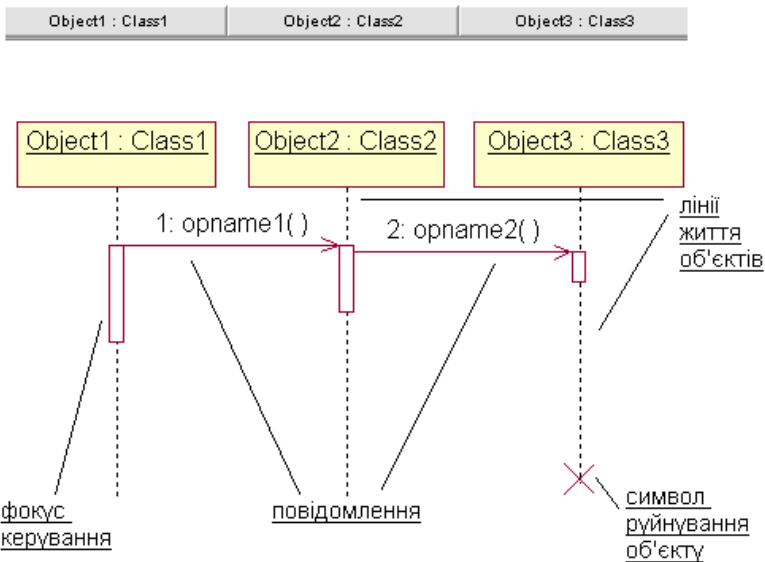


Рисунок 8.1 – Різні графічні примітиви діаграми послідовності

Зауваження. Не виключається ситуація, коли ім'я об'єкта може бути відсутнім на діаграмі послідовності. У цьому випадку вказується тільки ім'я класу, а сам об'єкт вважається анонімним.

Крайнім зліва на діаграмі зображується об'єкт-ініціатор взаємодії (Object1 на рис. 8.1). Праворуч зображується інший об'єкт, який безпосередньо взаємодіє з першим. Таким чином, всі об'єкти на діаграмі послідовності утворюють деякий порядок, який визначається ступенем активності цих об'єктів при взаємодії один з одним.

Другий вимір діаграми послідовності - вертикальна часова вісь, спрямована зверху вниз. Початковому моменту часу відповідає сама верхня частина діаграми. При цьому взаємодії об'єктів реалізуються за допомогою повідомлень, які надсилаються одними об'єктами іншим.

Повідомлення зображуються у вигляді горизонтальних стрілок з ім'ям повідомлення і також утворюють порядок за часом свого виникнення. Таким чином, повідомлення, розташовані на діаграмі послідовності вище, ініціюються раніше тих, що розташовані нижче. При цьому масштаб на осі часу не вказується, оскільки діаграма послідовності моделює лише часову впорядкованість взаємодій типу "раніше-пізніше".

8.2.2.1 Лінія життя об'єкта

Лінія життя об'єкта (object lifeline) зображується пунктирною вертикальною лінією, асоційованою з єдиним об'єктом на діаграмі послідовності. Вона служить для позначення періоду часу, протягом якого об'єкт існує в системі і може потенційно брати участь у всіх її взаємодіях. Якщо об'єкт існує в системі постійно, то і його лінія життя повинна тривати по всій площині діаграми послідовності від самої верхньої її частини до самої нижньої (Object1 та Object2 на рис. 8.1).

Окремі об'єкти, виконавши свою роль в системі, можуть бути зруйновані, щоб звільнити займані ними ресурси. Для таких об'єктів лінія життя обривається в момент його знищення. Для позначення моменту знищення об'єкта в мові UML використовується символ у формі латинської букви "X" (Object3 на рис. 8.1). Нижче цього символу пунктирна лінія не зображується, оскільки відповідного об'єкта в системі вже немає. Не обов'язково створювати всі об'єкти в початковий момент часу. Окремі об'єкти в системі можуть створюватися по мірі необхідності, економлячи ресурси системи і

підвищуючи її продуктивність. У цьому випадку прямокутник об'єкта зображується не у верхній частині діаграми послідовності, а в тій її частині, яка відповідає моменту створення об'єкта (Object6 на рис. 8.2). При цьому прямокутник об'єкта вертикально розташовується в тому місці діаграми, яке по осі часу збігається з моментом його виникнення в системі. Очевидно, об'єкт обов'язково створюється зі своєю лінією життя і, можливо, з фокусом керування (див. п. 8.3.3).

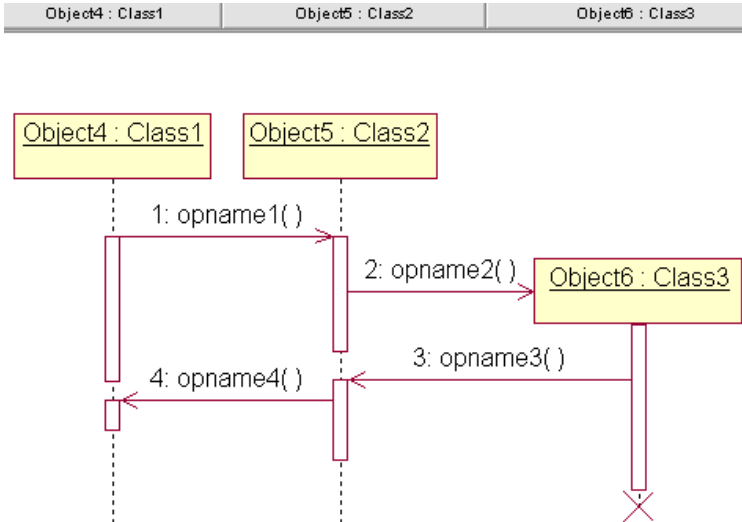


Рисунок 8.2 – Графічне зображення різних варіантів ліній життя об'єктів

8.2.2.2 Налаштування часу життя об'єкту

По подвійному натисненню миші на значку об'єкту або через контекстне меню та опцію *Open Specification...* активізується діалогове вікно параметрів об'єкта (рис. 8.3) в якому можна ввести назву об'єкта і клас, реалізацією якого є даний об'єкт.

Для реалізації об'єктів у вікні параметрів доступна опція *Persistence*, що відображає час життя об'єкта (від його створення до знищення). Зазвичай об'єкти існують в межах їх області видимості і автоматично знищуються системою, коли виходять за цю область. Це відбувається в тих випадках, коли, наприклад, в підпрограмі визначається об'єкт класу, який автоматично знищується при

завершенні підпрограми.

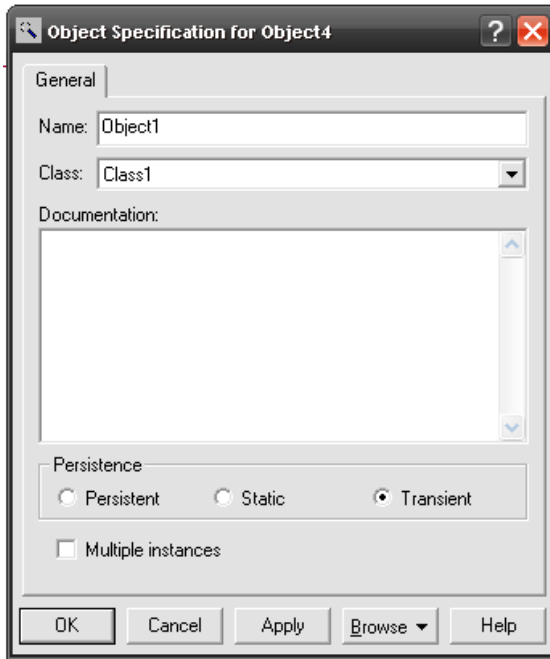


Рисунок 8.3 – Налаштування параметрів об'єкта

У мові C++ можна створити і пересувати об'єкт, не чекаючи, коли це зробить система, за допомогою команд `new` і `delete`. Зазвичай до такого варіанту вдаються й для зменшення файлу, що виконується. Відомо, що при створенні програми на Visual C++ визначення:

```
char s [10000];
```

приведе до збільшення виконуваного файлу на 10000 байт, тому що для цієї змінної буде зарезервовано місце в області даних програми.

Проте створення масиву за допомогою операторів:

```
char * s;  
s = new char [10000]; // робота з масивом
```

практично не приведе до збільшення розміру виконуваного файлу, хоча мається на увазі, що програміст звільнить виділену пам'ять до завершення програми за допомогою оператора:

delete s [];

інакше вона буде недоступна для системи в цілому.

Для того щоб не забути звільнити виділену пам'ять, оптимальним буде виділення пам'яті в конструкторі класу та звільнення її в деструкторі. Таким чином, можна настроїти наступні параметри життя об'єкту:

- *Persistent* – означає, що область видимості об'єкта перевищує час життя;

- *Static* – означає, що даний елемент існує на всьому протязі роботи програми;

- *Transient* – означає, що час життя об'єкту і область видимості збігаються.

Для того щоб показати, що ці властивості описують стан всіх об'єктів даного класу, можна вказати *Multiple Instance*.

Для безпосередньої вказівки моменту знищення об'єкта використовується значок *Destruction Marker*.

8.2.3 Фокус керування

Фокус керування – спеціальне поняття, яке застосовується для виділення активних станів об'єктів, коли вони виконують певні дії. Для позначення фокуса керування використовується вертикальний прямокутник (Object6 на рис. 8.2), верхнє ребро якого означає початок активності, а нижнє ребро – кінець активності. Якщо об'єкт стає активним декілька разів протягом свого існування, то формується декілька фокусів керування (Object5 на рис. 8.2), щоб виділити активні та пасивні стани об'єкта.

Важливо розуміти, що отримати фокус керування може тільки існуючий об'єкт, у якого в цей момент є лінія життя. Якщо ж деякий об'єкт був знищений, то знов виникнути в системі він вже не може. Замість нього може бути створений інший екземпляр цього ж класу, який, строго кажучи, буде іншим об'єктом.

8.2.4 Створення повідомлень

Для створення повідомлень необхідно вибрати стрілку *Object message* і з'єднати пунктирні лінії двох об'єктів в одному напрямку.

Потім ще раз з'єднати пунктирні лінії двох об'єктів, але в

іншому напрямку для передачі зворотного повідомлення (рис. 8.4).

З повідомленням можливо зв'язати операцію класу. Для цього необхідно натиснути правою кнопкою миші на повідомленні та у контекстному меню вибрати доступні операції класу (рис. 8.5).

Також можливо зв'язати з повідомленням знову створену операцію у класі. Для цього необхідно натиснути правою кнопкою миші на повідомленні та у контекстному меню вибрати опцію *<new operation>*. У діалоговому вікні, що з'явилося потрібно ввести назву операції (рис. 8.6). Таким чином, можливо не відкриваючи діаграму класів, додати до певного класу новий метод.

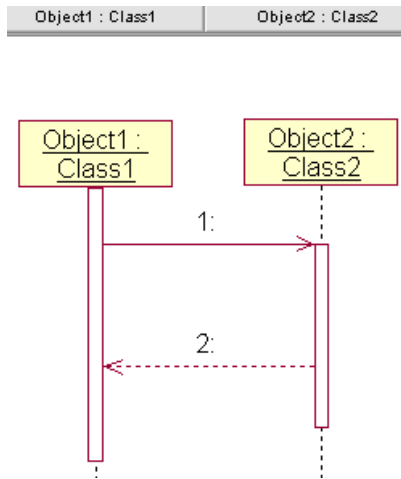


Рисунок 8.4 – Створення повідомлень між об'єктами

8.2.4.1 Властивості повідомлень

Для зміни властивостей повідомлень необхідно натиснути правою кнопкою миші на повідомлення, вибрати опцію *Open Specification...* або двічі натиснути лівою кнопкою миші на повідомленні. У діалоговому вікні, що з'явилося потрібно перейти на вкладку *Detail* (рис. 8.7).

Тут знаходяться дві групи радіо-кнопок:

1. *Synchronization* – визначає порядок обміну повідомленнями і може бути обрана з наступних варіантів:

➤ *Simple* – проста посилка повідомлення;

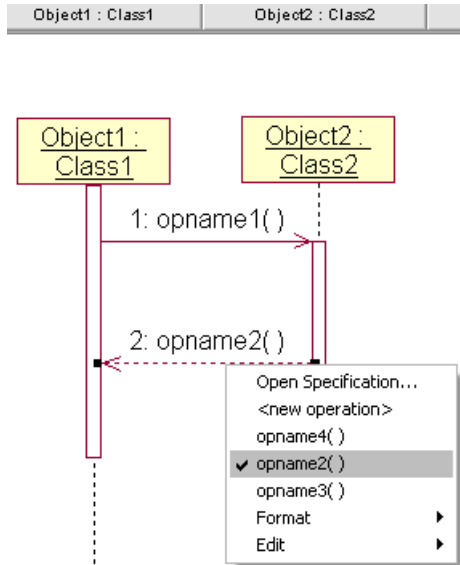


Рисунок 8.5 – Зв’язування операції класу з повідомленням

- *Synchronous* – операція відбувається лише в тому випадку, коли клієнт посилає повідомлення, а сервер може прийняти повідомлення клієнта;
- *Balking* – операція відбувається лише в тому випадку, коли сервер готовий негайно прийняти повідомлення, якщо сервер не готовий до прийому, клієнт не видає повідомлення;
- *Timeout* – клієнт відмовляється від видачі повідомлення, якщо сервер протягом певного часу не може його прийняти;
- *Procedure Call* – клієнт викликає процедуру сервера і повністю передає йому управління;
- *Asynchronous* – клієнт видає повідомлення, і, не чекаючи відповіді сервера, продовжує виконання свого програмного коду;
- *Return* - визначає, що відбувається повернення з процедури.

Для кожного виду операцій стрілка повідомлення змінюється відповідно до рис. 8.8.

2. *Frequency* – визначає частоту обміну повідомленнями:

- *Periodic* – повідомлення надходять від клієнта із заданою періодичністю;

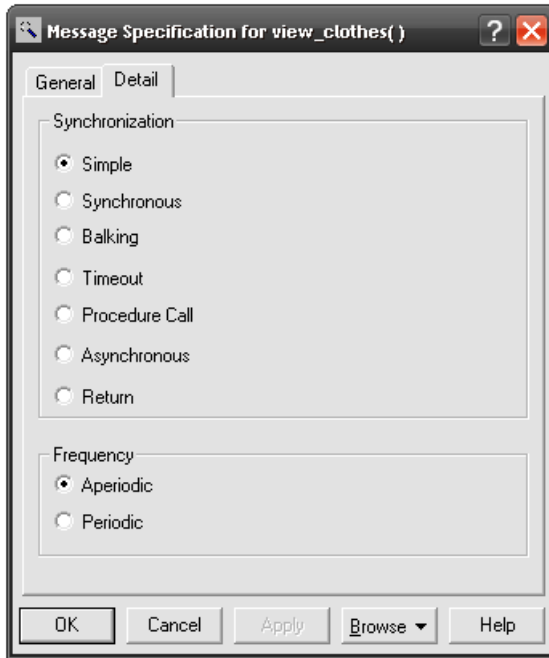


Рисунок 8.7 – Встановлення властивостей повідомлення

➤ Aperiodic – повідомлення надходять нерегулярно.

Для того щоб видалити неправильно введене повідомлення з діаграми, потрібно натиснути Ctrl + D.

8.2.5 Створення діаграми послідовностей

Створимо діаграму послідовностей за допомогою Menu: Browse => Logical View => New => Sequence Diagram (рис. 8.9). Діаграма послідовностей являє собою розворот взаємодії у часі і відображає послідовність видачі повідомлень клієнтами.

8.2.5.1 Строка інструментів

При переході на цю діаграму панель інструментів змінилася на нову, в якій тепер доступні значки *Text Box*, *Note*, *Anchor to Item*, *Object*, *Message*, *Message to Self* (рис. 8.10). Перші три були розглянуті для попередньої діаграми і виконують ті ж функції.

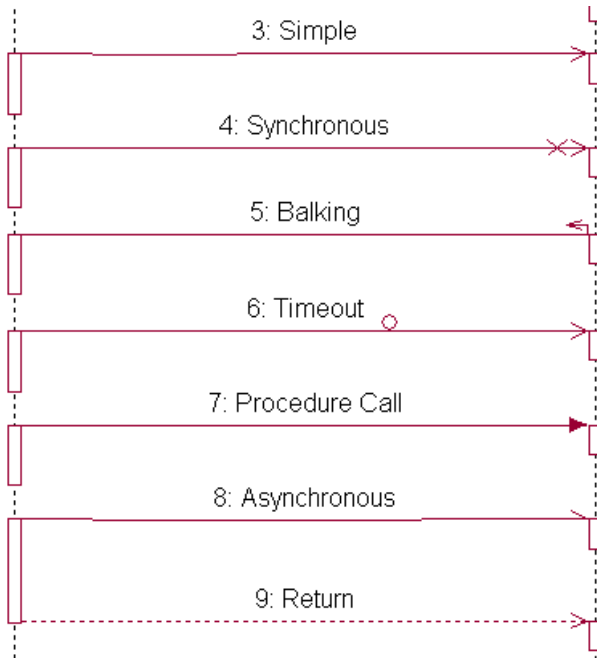


Рисунок 8.8 – Різні види повідомлень

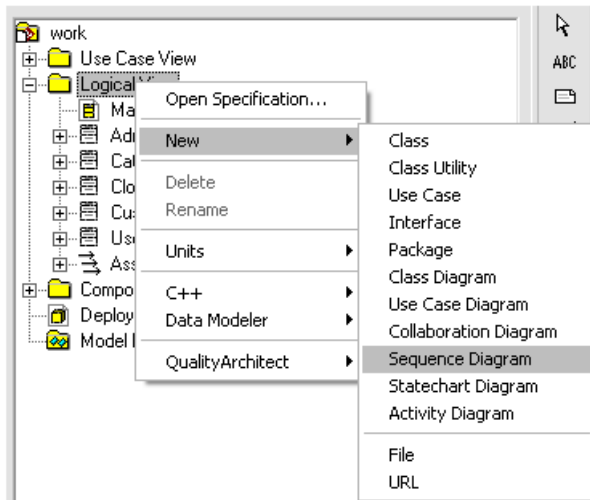


Рисунок 8.9 – Створення діаграми послідовностей



Рисунок 8.10 – Строка інструментів Sequence Diagram

Object (об'єкт) . Значок *Object* дозволяє включити новий об'єкт в діаграму. Кожен об'єкт є реалізацією класу, тому в ньому можна вказати клас, на основі якого він створений.

Message (повідомлення) . Значок *Message* дозволяє створити повідомлення, передане від одного об'єкта до іншого. Так як вся взаємодія в об'єктно-орієнтованих системах здійснюється за допомогою повідомлень між об'єктами, то класи повинні дозволяти відправку або отримання повідомлень.

Message to Self (повідомлення самому собі) . Даний значок дозволяє показати, що відправник є одночасно і його одержувачем.

При обміні повідомленнями одні класи є клієнтами і приймають повідомлення, а інші - серверами і відправляють повідомлення. У тому випадку, коли об'єкт відправляє повідомлення самому собі, він одночасно є і сервером і клієнтом, що й відображає даний значок.

Return Message (повернення повідомлення) . Даний значок дозволяє показати, що відбувається повернення управління з викликаної підпрограми на сервері клієнта.

Destruction Marker (маркер знищення) . Даний значок дозволяє показати, що відбувається знищення програмного об'єкта.

8.2.6 Приклад виконання

Необхідно розробити діаграму послідовностей для опису взаємодії між об'єктами класів «User» («Користувач») та «Catalog» («Каталог»). Зокрема, необхідно змоделювати процес перегляду користувачем усієї інформації, наявної у каталозі одягу та пошуку інформації про певний товар у каталозі одягу інтернет-магазину.

Реалізація.

1. Для створення діаграми послідовностей у браузері необхідно

виконати наступну послідовність дій: Logical view => New => Sequence Diagram (рис 8.11).

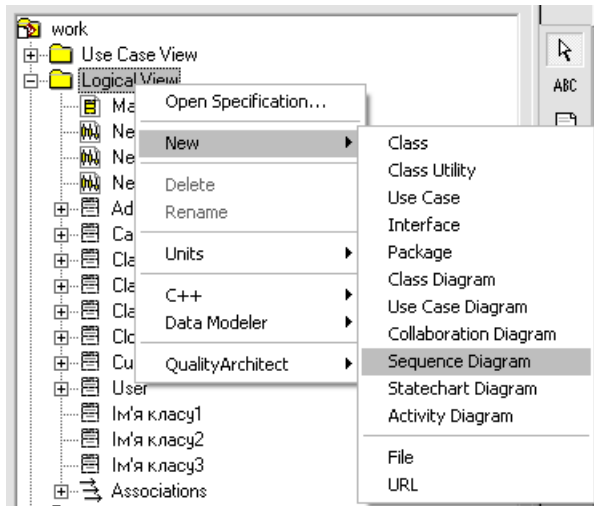


Рисунок 8.11 – Створення діаграми послідовностей

2. На створеній діаграмі послідовностей необхідно розмістити об'єкти за допомогою значка  на панелі інструментів. На діаграмі послідовностей будуть знаходитися наступні об'єкти: Vadim (клас «User») та Catalog_clothes (клас «Catalog»). Розташування об'єктів на діаграмі послідовностей зображено на рис. 8.12.

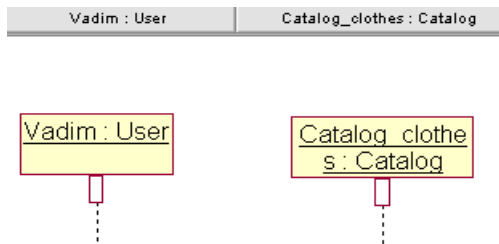


Рисунок 8.12 – Об'єкти на діаграмі послідовностей

3. За допомогою інструмента  створюємо повідомлення від фокусу управління класу «User» до фокусу управління класу «Catalog» (рис. 8.13).

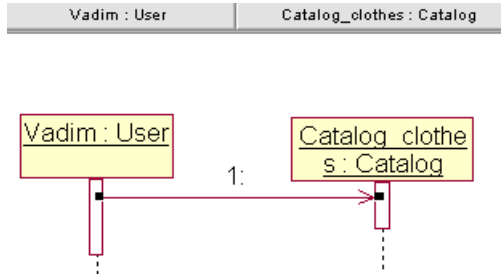


Рисунок 8.13 – Створення повідомлення

4. Зв'яжемо зі створеним повідомленням існуючу операцію view_clothes(). Для цього необхідно клікнути правою кнопкою миші на лінії повідомлення та обрати view_clothes() (рис. 8.14).

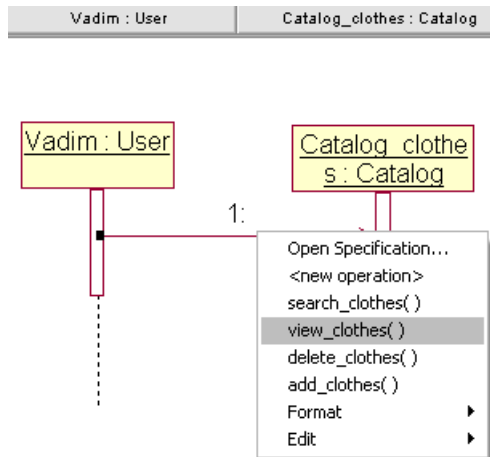


Рисунок 8.14 – Зв'язування операції з повідомленням

5. За допомогою інструмента  створюємо повідомлення від фокусу управління класу «Catalog» до фокусу управління класу «User» (рис. 8.15).

6. Зв'яжемо зі створеним повідомленням нову операцію. Для цього необхідно клікнути правою кнопкою миші на лінії

повідомлення, обрати <new operation> (рис. 8.16) та ввести назву операції get_clothes_info() (рис. 8.17).

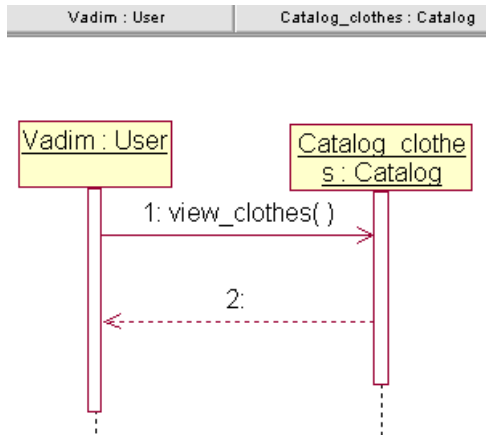


Рисунок 8.15 – Створення зворотного повідомлення

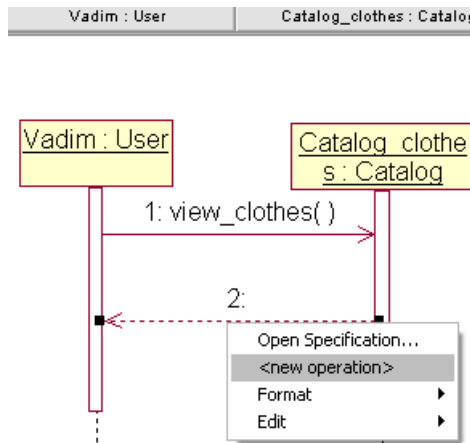


Рисунок 8.16 – Створення нової операції

7. За допомогою інструмента  створюємо повідомлення від фокусу управління класу «User» до фокусу управління класу «Catalog» (рис. 8.18).

8. Зв'яжемо зі створеним повідомленням існуючу операцію search_clothes(). Для цього необхідно клікнути правою кнопкою миші

на лінії повідомлення та обрати `search_clothes()` (рис. 8.19).

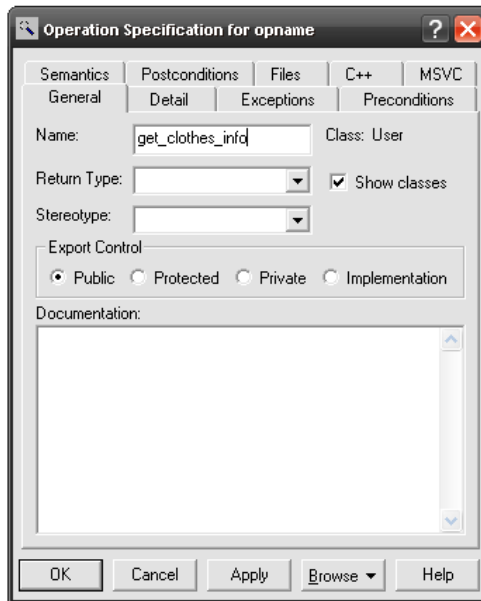


Рисунок 8.17 – Вікно специфікацій нової операції

9. За допомогою інструмента  створюємо повідомлення від фокусу управління класу «Catalog» до фокусу управління класу «User» (рис. 8.20).

10. Зв'яжемо зі створеним повідомленням існуючу операцію `get_clothes_info()`. Для цього необхідно клікнути правою кнопкою миші на лінії повідомлення та обрати `get_clothes_info()` (рис. 8.21). Результат побудови діаграми послідовностей наведено на рис. 8.22.

8.3 Завдання на лабораторну роботу

8.3.1 Ознайомитися з пунктом 8.2 і конспектом лекцій.

8.3.2 За обраною у лабораторній роботі №1 темою створити діаграми послідовностей.

8.3.3 Оформити звіт по роботі.

8.3.4 Відповісти на контрольні питання.

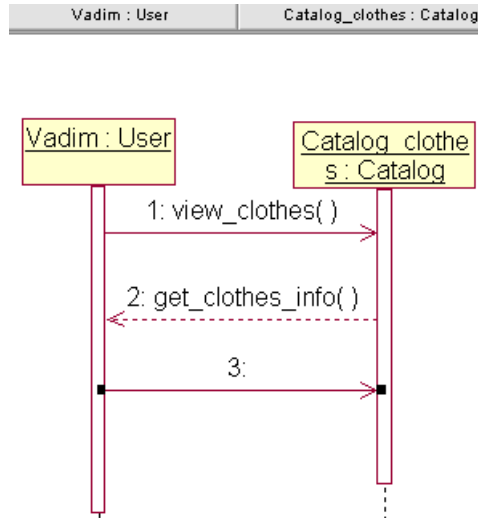


Рисунок 8.18 – Створення повідомлення

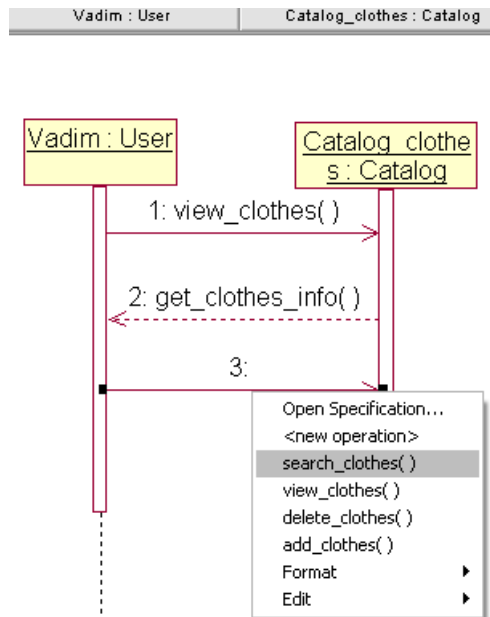


Рисунок 8.19 – Зв'язування операції з повідомленням

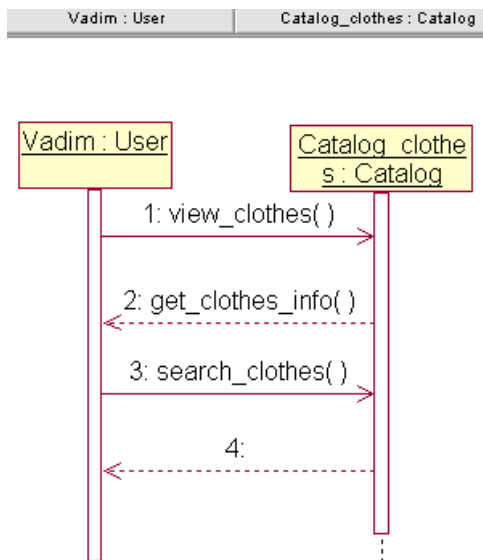


Рисунок 8.20 – Створення зворотного повідомлення

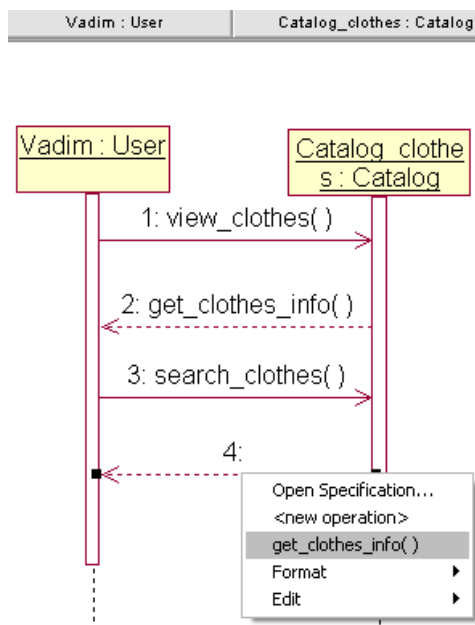


Рисунок 8.21 – Зв'язування операції з повідомленням

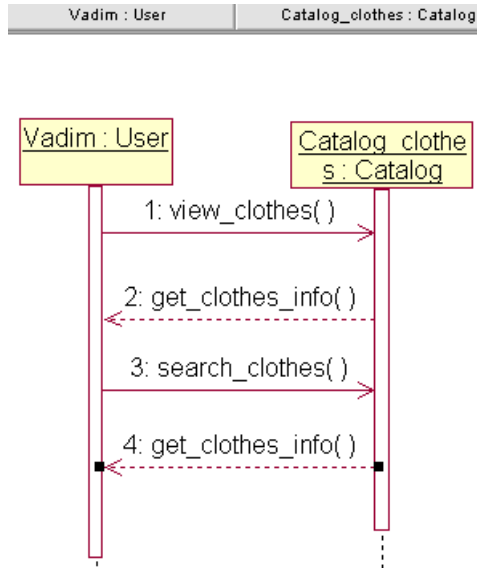


Рисунок 8.22 – Результиуюча діаграма послідовностей

8.4 Зміст звіту

- 8.4.1 Тема та мета роботи.
- 8.4.2 Завдання до роботи.
- 8.4.3 Побудована діаграма послідовностей.
- 8.4.4 Висновки, за результатами виконання роботи.

8.5 Контрольні запитання

- 8.5.1 Для чого призначені діаграми послідовностей?
- 8.5.2 В якому напрямку спрямований час?
- 8.5.3 В якому напрямку спрямовані повідомлення?
- 8.5.4 В яких станах може знаходитися об'єкт?
- 8.5.5 Для чого призначений фокус керування?
- 8.5.6 Які види повідомлень дозволяє відобразити діаграма?
- 8.5.7 Як налаштувати відображення часу життя об'єкту?
- 8.5.8 Як створити класи, не виходячи з діаграми?
- 8.5.9 Який порядок обміну повідомленнями може бути заданий?
- 8.5.10 Як може бути задана частота обміну повідомленнями?

9 ЛАБОРАТОРНА РОБОТА № 9

Діаграми кооперації

9.1 Мета роботи

Навчитися проектувати діаграми кооперації (ДК) з використанням програмного середовища Rational Rose: створювати об'єкти, накладати зв'язки між об'єктами, створювати повідомлення між об'єктами та встановлювати властивості повідомлень.

9.2 Основні теоретичні відомості

9.2.1 Призначення діаграми кооперації

Діаграма кооперації (Collaboration Diagram), на відміну від попередньої, не акцентує увагу на послідовності передачі повідомлень, а відображає наявність взаємозв'язків взагалі, тобто відображає наявність повідомлень від клієнтів до серверів. Так як шкала часу не бере участь в демонстрації повідомлень, то ця діаграма є оптимальною для того, з'ясувати взаємодію всіх об'єктів.

Крім того, ця діаграма показує взаємодію між об'єктами, а не класами, тобто є миттєвим знімком об'єктів системи в певному стані.

Діаграма кооперації призначена для специфікації структурних аспектів взаємодії. Головна її особливість полягає в можливості графічно представити не тільки послідовність взаємодії, але і всі структурні відносини між об'єктами, які беруть участь у цій взаємодії.

На ДК у вигляді прямокутників зображуються об'єкти, що беруть участь у взаємодії, які містять ім'я об'єкта, його клас і, можливо, значення атрибутів. Далі вказуються асоціації між об'єктами у вигляді різних сполучних ліній. При цьому можна явно вказати імена асоціації і ролей, які відіграють об'єкти в даній асоціації. Додатково можуть бути зображені динамічні зв'язки - потоки повідомлень. Вони представляються у вигляді сполучних ліній між об'єктами, над якими розташовується стрілка з вказівкою напрямку, імені повідомлення і порядкового номера в загальній послідовності ініціалізації повідомлень.

У мові UML *кооперація* використовується для зображення множини взаємодіючих з певною метою об'єктів та позначає певні особливості реалізації окремих операцій.

Для опису кооперації використовуються два рівня:

- специфікації – для позначення ролі класифікаторів і асоціацій;

- прикладів – для позначення зв'язків між екземплярами.

На ДК рівня специфікації зображуються ролі елементів взаємодії, тобто класів і асоціацій.

На ДК рівня прикладів зображуються множини екземплярів класів (об'єктів), що відносяться до реалізації операції, та екземплярів асоціації (зв'язків), до яких додаються стрілки повідомлень.

В кооперації рівня прикладів визначаються властивості, які повинні мати екземпляри для того, щоб брати участь в кооперації. На ДК також указуються асоціації, які повинні мати місце між об'єктами кооперації. При цьому не обов'язково зображувати всі властивості або всі асоціації, оскільки на ДК присутні тільки ролі класифікаторів, але не самі класифікатори. Таким чином, у той час як класифікатор вимагає повного опису всіх своїх екземплярів, роль класифікатора вимагає опису тільки тих властивостей і асоціацій, які необхідні для участі в окремій кооперації.

Одна і та ж сукупність об'єктів може брати участь у різних коопераціях. При цьому, залежно від розглянутої кооперації, можуть змінюватися як властивості окремих об'єктів, так і зв'язки між ними. Це відрізняє ДК від діаграми класів, на якій повинні бути вказані всі властивості і асоціації між елементами діаграми.

9.2.2 Об'єкти

Об'єкти є основними елементами або графічними примітивами, з яких будується діаграма кооперації на рівні прикладів. Для графічного зображення об'єктів використовується такий же символ прямокутника, що і для класів.

Як зазначалося вище, об'єкт (object) є окремим екземпляром класу, який створюється на етапі виконання програми. Він може мати своє власне ім'я і конкретні значення атрибутів. Стосовно об'єктів формат рядка класифікатора доповнюється ім'ям об'єкту і набуває наступний вигляд (при цьому весь запис підкреслюється):

```
<Ім'я об'єкта> '/' <Ім'я ролі класифікатора> ':' <Ім'я  
класифікатора>  
[':' <Ім'я класифікатора>] *
```

Тут *Ім'я ролі класифікатора* може не вказуватися. У цьому випадку воно виключається з рядка тексту разом з наступною двокрапкою. Ім'я ролі може бути опущено в тому випадку, якщо існує тільки одна роль в кооперації, яку можуть відігравати об'єкти, створені на базі цього класу.

Таким чином, для позначення ролі класифікатора достатньо вказати або ім'я класу (разом з двокрапкою), або ім'я ролі (разом з похилою рисою). В іншому випадку прямокутник відповідатиме звичайному класу. Якщо роль, яку має відігравати об'єкт, успадковується від декількох класів, то всі вони повинні бути вказані явно і розділятися комою і двокрапкою.

Зауваження. У прямокутнику об'єкта ім'я об'єкта, ім'я ролі з символом Т або ім'я класу можуть бути відсутні. Однак двокрапка завжди має стояти перед ім'ям класу, а похила риска - перед ім'ям ролі. Стосовно до об'єктів весь запис має бути підкреслено, а ім'я об'єкта має бути записано з малої літери.

Нижче наводяться можливі варіанти запису рядка тексту в прямокутнику об'єкта.

- :C – анонімний об'єкт, що утворюється на основі класу C;
- /R – анонімний об'єкт, що грає роль R;
- /R:C – анонімний об'єкт, що утворюється на основі класу C, та що грає роль R;
- O/R – об'єкт з ім'ям O, що грає роль R;
- O:C – об'єкт з ім'ям O, утворюваний на основі класу C;
- O/R:C – об'єкт з ім'ям O, утворюваний на основі класу C, та що грає роль R;
- O – об'єкт з ім'ям O;
- O – «об'єкт-сирота» з ім'ям O;
- /R – роль з ім'ям R;
- :C – анонімна роль на базі класу C;
- /R:C – роль з ім'ям R на основі класу C.

9.2.2.1 Мультиоб'єкт

Мультиоб'єкт (multiobject) є сукупністю об'єктів та призначений для зображення передачі інформації усій сукупності об'єктів. Для створення мультиоб'єкта необхідно клікнути правою кнопкою миші по об'єкту та у контекстному меню обрати опцію *Open Specification...*

У діалоговому вікні, що з'явилося необхідно ввести його ім'я та встановити галочку у полі *Multiple instances* (рис. 9.1). Мультиоб'єкт зображується трьома прямокутниками (рис. 9.2). При цьому стрілка повідомлення відноситься до всієї множини об'єктів, які позначають даний мультиоб'єкт.

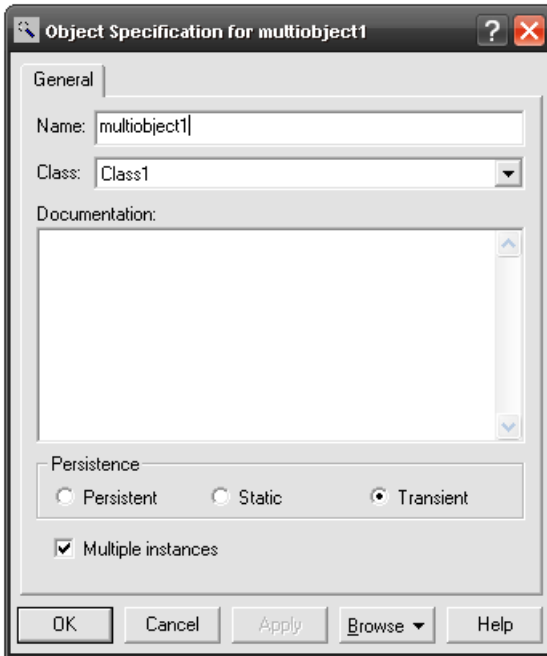


Рисунок 9.1 – Перетворення об'єкту у мультиоб'єкт

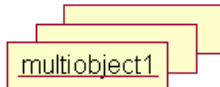


Рисунок 9.2 – Створений мультиоб'єкт

9.2.2.2 Активні та пасивні об'єкти

Згідно зі специфікацією мови UML усі об'єкти можна розділити на два типи: пасивні об'єкти та активні об'єкти.

Пасивний об'єкт має обмежену функціональність та може лише обробляти дані без можливості ініціювання керування іншими об'єктами.

Активний об'єкт, у порівнянні з пасивним об'єктом, має можливість не лише обробляти дані, а й ініціювати керування іншими об'єктами. Активний об'єкт має власну нитку керування, яка уявляє собою потік керування.

Відмінність між процесом і ниткою полягає в ступені використання ресурсів. Говорячи про процес, мають на увазі ресурсомісткий потік управління, тобто процес повністю монополізує ресурси системи. Нитка може використовувати лише невелику частину ресурсів системи. Прикладом може служити виконання деякої програми в своєму адресному просторі або у фоновому режимі.

Активні об'єкти позначаються прямокутником з більш широкими межами (рис. 9.3). Іноді може бути вказано ключове слово {active}, щоб виділити активний об'єкт на діаграмі. Кожен активний об'єкт може ініціювати єдину нитку або процес керування і представляти вихідну точку потоку керування.

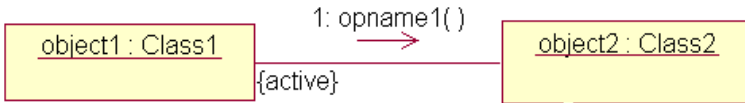


Рисунок 9.3 – Створення та взаємодія активного об'єкту

9.2.2.3 Контекстне меню об'єкту

Контекстне меню об'єкту має вигляд, наведений на рис. 9.4.

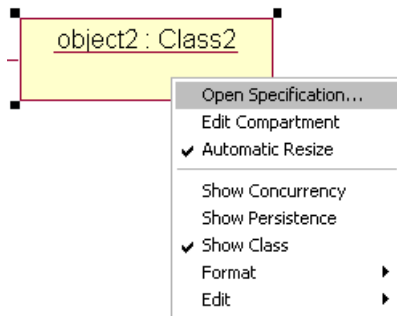


Рисунок 9.4 – Контекстне меню об'єкту

Можливості, що надаються за допомогою даного меню:

- *Open Specification* – редагування специфікацій об'єкта;

- *Edit Compartment* – активізує діалогове вікно показу додаткової інформації про об'єкт. Зміст такої інформації залежить від типу об'єкта;
- *Automatic Resize* – дозволяє встановлювати автоматичну настройку розміру об'єкта по довжині тексту, що міститься в ньому;
- *Show Concurrency* – дозволяє включити показ на даному значку типу узгодження при створенні багатопотокової програми. Даний тип визначений у класі;
- *Show Persistence* – дозволяє показати на діаграмі час життя об'єкта;
- *Show Class* – дозволяє показати на діаграмі ім'я класу.

9.2.2.4 Налаштування області видимості об'єктів

Для того щоб налаштувати область видимості об'єктів необхідно клікнути правою кнопкою миші по зв'язку та у контекстному меню обрати опцію *Open Specification...* В результаті з'явиться діалогове вікно, яке зображено на рис. 9.5.

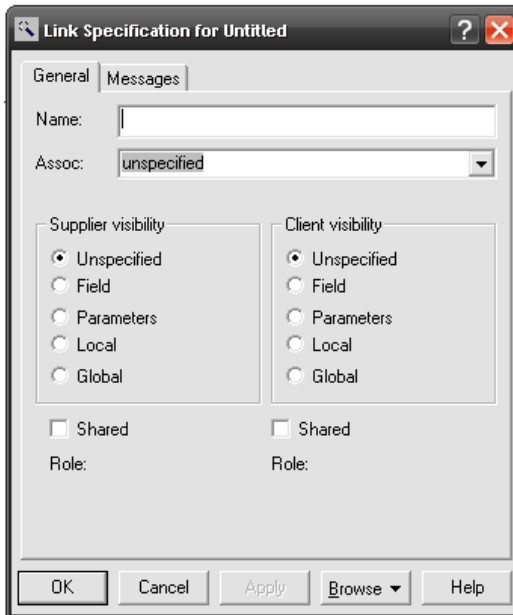


Рисунок 9.5 – Завдання області видимості об'єктів

Для завдання області видимості об'єкта сервера служить блок *Supplier visibility*, клієнта – *Client visibility*.

У цих блоках доступні значення для вибору:

- *Unspecified* – не визначено, це значення присвоюється за замовчанням;
- *Field* – об'єкт включено до іншого об'єкту;
- *Parameter* – об'єкт передається параметром в інший об'єкт;
- *Local* – об'єкт локально визначено у межах іншого об'єкта;
- *Global* – об'єкт глобальний по відношенню до іншого об'єкта.

При зміні області видимості на кінцях лінії сполучення з'являється квадратик з вказаною областю видимості. Тут же можна встановити прапорець, який показує, що об'єкт використовується спільно (Shared). Область видимості можна змінити з контекстного меню, причому, змінюється та сторона лінії сполучення, до якої був ближче курсор миші при активізації контекстного меню.

9.2.3 Зв'язки

Зв'язок (link) є екземпляром або прикладом довільної асоціації. Зв'язок як елемент мови UML може мати місце між двома і більше об'єктами. Бінарний зв'язок на діаграмі кооперації зображується відрізком прямої лінії, що з'єднує два прямокутники об'єктів (див. рис. 9.3). На кожному з кінців цієї лінії можуть бути явно вказані імена ролей даної асоціації. Поруч з лінією в її середній частині може записуватися ім'я відповідної асоціації.

Зв'язки не мають власних імен, оскільки повністю ідентичні як екземпляри асоціації. Тобто, всі зв'язки на діаграмі кооперації можуть бути тільки анонімними і записуються без двокрапки перед ім'ям асоціації. Для зв'язків не вказується також і кратність.

9.2.3.1 Стереотипи зв'язків

У мові UML використовуються наступні стереотипи для вказівки особливості даного зв'язку:

- "association" – асоціація (передбачається за замовчуванням, тому цей стереотип можна не вказувати);
- "parameter" – параметр методу. Відповідний об'єкт може бути тільки параметром деякого методу;

- "local" – локальна змінна методу. Її область видимості обмежена тільки сусіднім об'єктом;
- "global" – глобальна змінна. Її область видимості розповсюджується на всю діаграму кооперації;
- "self" - рефлексивний зв'язок об'єкта з самим собою, який допускає передачу об'єктом повідомлення самому собі. На діаграмі кооперації рефлексивний зв'язок зображується петлею у верхній частині прямокутника об'єкту.

Деякі приклади зв'язків з різними стереотипами зображені на рис. 9.6.

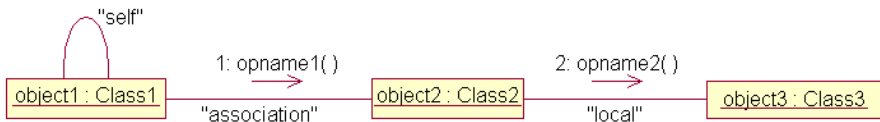


Рисунок 9.6 – Стереотипи зв'язків

9.2.4 Повідомлення

На діаграмі кооперації повідомлення мають певний сенс, бо вони визначають комунікацію між двома об'єктами, один з яких є відправником інформації, а інший – одержувачем інформації. В даному контексті повідомлення є стимулом для початку відправки інформації, виконання певних дій.

Згідно зі специфікацією мови UML повідомлення визначають ролі, які грають відправник та одержувач повідомлення. На діаграмах кооперації повідомлення зображаються певними типами стрілок поряд з відповідним зв'язком (рис. 9.7):

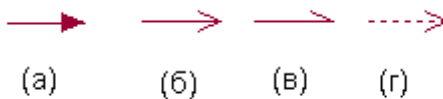


Рисунок 9.7 – Графічне зображення різних типів повідомлень

- суцільна лінія із замальованою трикутною стрілкою (рис. 9.7, а) використовується для позначення синхронних повідомлень виклику підпрограми або будь-якого потоку керування. Під синхронними повідомленнями розуміються такі, які ініціюються

об'єктом після завершення виконання певної дії. Відправник повідомлення чекає поки уся вкладена послідовність повідомлень не буде оброблена отримувачем повідомлення;

➤ суцільна лінія з V-подібною стрілкою (рис. 9.7, б) використовується для позначення асинхронних повідомлень певної стадії в послідовності потоку керування. Під асинхронними повідомленнями розуміються такі, які ініціюються об'єктом незалежно від того, чи прийняв попереднє повідомлення або виконав певну дію одержувач повідомлення чи ні;

➤ суцільна лінія з напівстрілкою зверху лінії (рис. 9.7, в) використовується для позначення повідомлень асинхронного потоку керування, які створюються активними об'єктами в довільні моменти часу;

➤ пунктирна лінія з V-подібною стрілкою (рис. 9.7, г) використовується для позначення повернення керування з виклику підпрограми.

Для встановлення типу стрілки повідомлення необхідно клікнути правою кнопкою миші по стрілці повідомлення та у контекстному меню обрати опцію *Open Specification...* У діалоговому вікні, що з'явилось необхідно перейти на вкладку *Detail* та обрати необхідний тип повідомлення (рис. 9.8).

9.2.5 Створення діаграми кооперації

Розробники Rational Rose заклали зручну можливість створення на основі діаграми послідовностей діаграми кооперацій і навпаки.

Для створення діаграми кооперацій на основі діаграми послідовностей, знаходячись в діаграмі послідовностей, зробимо наступне: Menu: Browse => Create Collaboration diagram. Буде створена діаграма, що представляє, на перший погляд, нагромадження значків. Але після її «розтягнення» мишкою, діаграма набуде цілком зрозумілий вигляд (рис. 9.9).

Також можна створити діаграму, натиснувши значок *Interaction diagram*  на панелі інструментів *Standard*. При цьому для створення необхідного типу діаграми в діалоговому вікні (рис. 9.10) необхідно вибрати тип діаграми *Collaboration*.

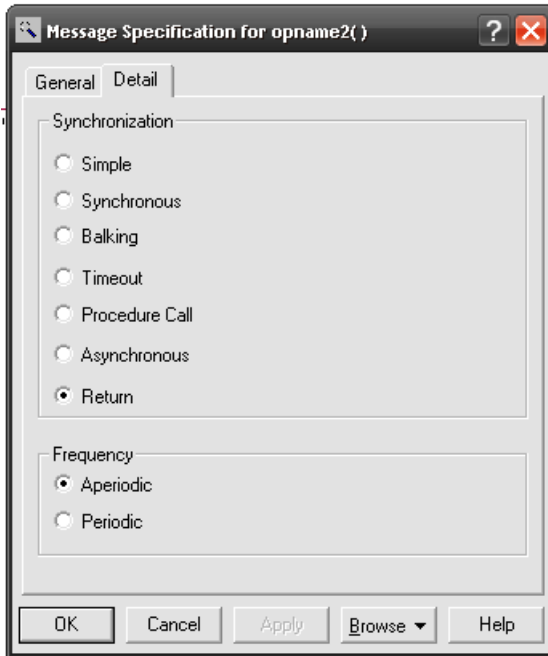


Рисунок 9.8 – Вкладка Detail специфікації повідомлення

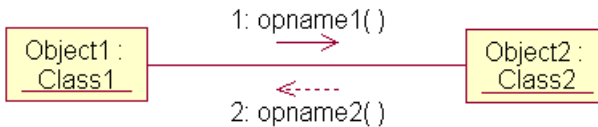


Рисунок 9.9 – Створена діаграма кооперації

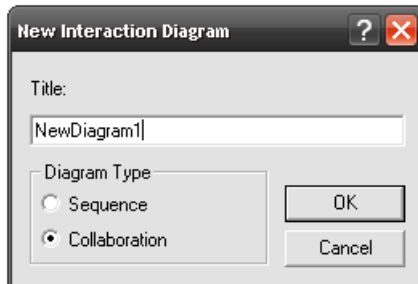


Рисунок 9.10 – Вікно вибору створення діаграми кооперації

Однак у цьому випадку буде створена порожня діаграма, і вже створені об'єкти не будуть туди перенесені.

9.2.5.1 Строка інструментів

При активізації діаграми рядок інструментів набуває вигляд як на рис. 9.11. Розглянемо нові інструменти.



Рисунок 9.11 – Строка інструментів діаграми кооперації

Object (об'єкт) . Дозволяє створювати об'єкти, які мають стани, поведінку та є індивідуальними. Кожен об'єкт на діаграмі показує реалізацію деякого класу.

Class Instance (реалізація класу) . Дозволяє додавати абстрактні реалізації класу в діаграму. У чому різниця між об'єктом і абстрактною реалізацією класу?

При додаванні значка на діаграму зовні ці значки не відрізняються. Відмінність в їх внутрішньому змісті. Об'єкт має на увазі налаштування його часу життя та інших властивостей цього конкретного об'єкту. Абстрактна реалізація класу не дозволяє змінювати ці властивості і призначена тільки для зображення взаємодії.

Object Link (зв'язок об'єкта) . Взаємодії об'єктів відображаються за допомогою зображення їх зв'язків. Існування зв'язків між двома класами символізує взаємодію між їх реалізаціями (об'єктами, створеними на основі цих класів). При цьому один об'єкт може посилати повідомлення іншому об'єкту.

Link To Self (зв'язок з самим собою) . Так як об'єкти можуть посилати повідомлення самим собі, то цей значок показує, що об'єкт має зворотний зв'язок з самим собою.

Link Message (передача повідомлення) . Дозволяє відобразити зв'язок, який передбачає обов'язкову передачу

повідомлення.

Reverse Link Message (зворотня передача повідомлення)

Дозволяє відобразити зв'язок, який передбачає обов'язкову передачу повідомлення аналогічно попередньому пункту, але в зворотному напрямку.

Data Token (потік даних)

Data Token дозволяє відобразити зв'язок, що показує передачу даних від одного об'єкта іншому.

Reverse Data Token (зворотній потік даних)

Як і попередній значок, дозволяє відобразити зв'язок, що показує передачу даних від одного об'єкта до іншого, але в зворотному напрямку.

9.2.6 Приклад виконання

Необхідно розробити діаграму кооперації для опису взаємодії між об'єктами класів «User» («Користувач») та «Catalog» («Каталог»). Зокрема, необхідно змодельовати процес перегляду усієї наявної у каталозі одягу інформації та пошуку необхідної інформації про певний товар користувачем у каталозі одягу інтернет-магазину.

Реалізація.

1. Створення діаграми кооперації можливе двома способами. Перший спосіб полягає у формуванні діаграми кооперації з існуючої діаграми послідовностей. Для цього необхідно виконати наступні команди Menu: Browse => Create Collaboration Diagram. Результат виконання зображений на рис. 9.12.

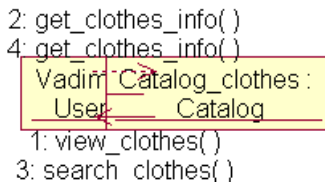


Рисунок 9.12 – Проміжний результат створення діаграми кооперації з діаграми послідовностей

У новому вікні програми необхідно «розтягнути» отримані об'єкти для отримання читабельної діаграми кооперації (рис. 9.13).

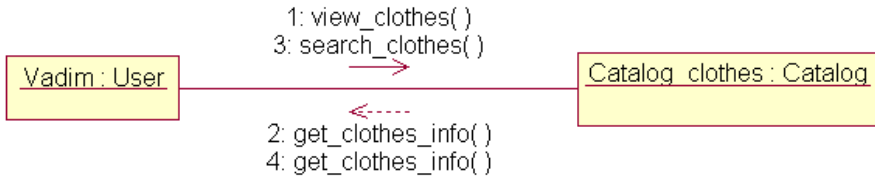


Рисунок 9.13 – Результуюча діаграма кооперації

2. Якщо ж діаграма послідовностей ще не створена, потрібно створювати діаграму кооперації вручну. Для цього в браузері необхідно виконати наступну послідовність дій: Logical view => New => Collaboration Diagram (рис. 9.14).

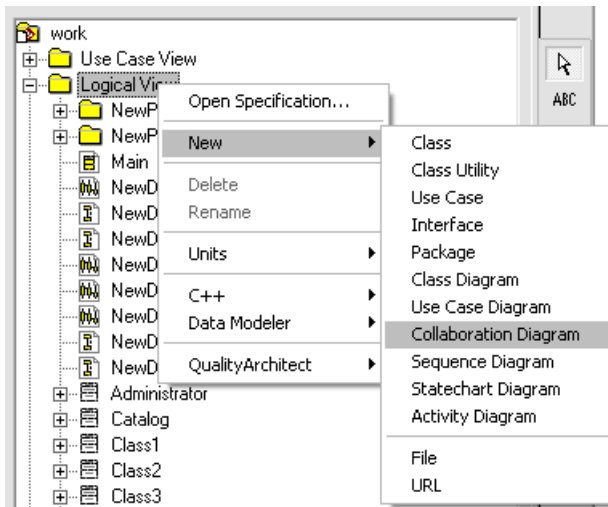


Рисунок 9.14 – Створення діаграми кооперації

3. На створеній діаграмі кооперації необхідно розмістити об'єкти за допомогою значка  на панелі інструментів. На діаграмі кооперації будуть знаходитися наступні об'єкти: Vadim (клас «User») та Catalog_clothes (клас «Catalog»). Розташування об'єктів на діаграмі кооперації зображено на рис. 9.15.

4. За допомогою інструмента  створюємо зв'язок між об'єктом «Vadim» класу «User» та об'єктом «Catalog_clothes» класу «Catalog» (рис. 9.16).



Рисунок 9.15 – Об'єкти на діаграмі кооперації



Рисунок 9.16 – Створення зв'язку між об'єктами

5. За допомогою інструмента  створюємо повідомлення для існуючого зв'язку між об'єктами шляхом натиснення лівої кнопки миші на зв'язку (рис. 9.17).



Рисунок 9.17 – Створення повідомлення

6. Зв'яжемо зі створеним повідомленням існуючу операцію view_clothes(). Для цього необхідно клікнути правою кнопкою миші на лінії повідомлення та обрати view_clothes() (рис. 9.18).

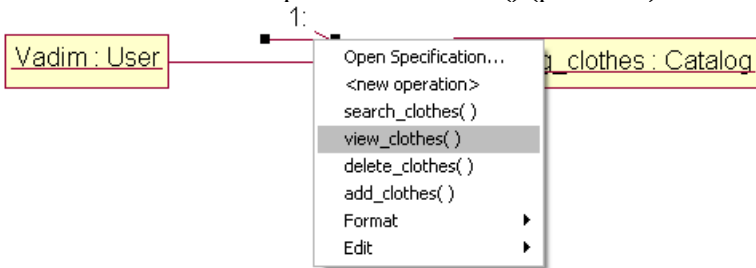


Рисунок 9.18 – Зв'язування операції з повідомленням

7. За допомогою інструмента  створюємо повідомлення для існуючого зв'язку між об'єктами шляхом натиснення лівої кнопки миші на зв'язку (рис. 9.19).

8. Зв'яжемо зі створеним повідомленням існуючу операцію get_clothes_info(). Для цього необхідно клікнути правою кнопкою миші на лінії повідомлення та обрати get_clothes_info() (рис. 9.20).

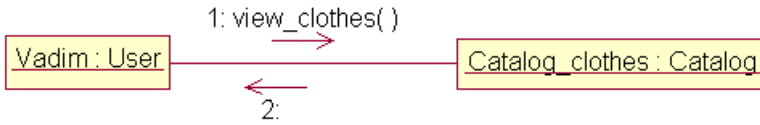


Рисунок 9.19 – Створення зворотного повідомлення

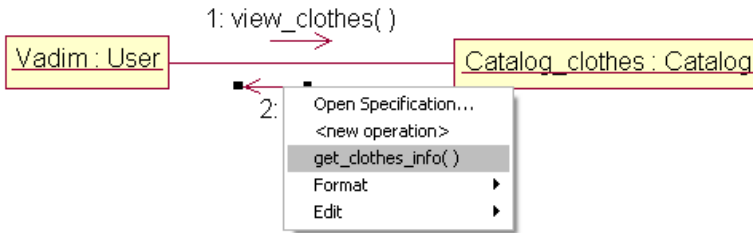


Рисунок 9.20 – Зв’язування операції з повідомленням

9. За допомогою інструмента  створюємо повідомлення для існуючого зв’язку між об’єктами шляхом натиснення лівої кнопки миші на зв’язку (рис. 9.21).

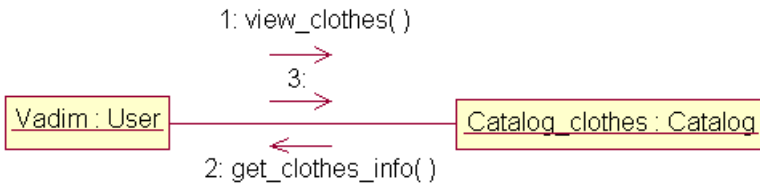


Рисунок 9.21 – Створення повідомлення

10. Зв’яжемо зі створеним повідомленням існуючу операцію `search_clothes()`. Для цього необхідно клікнути правою кнопкою миші на лінії повідомлення та обрати `search_clothes()` (рис. 9.22).

11. За допомогою інструмента  створюємо повідомлення для існуючого зв’язку між об’єктами шляхом натиснення лівої кнопки миші на зв’язку (рис. 9.23).

12. Зв’яжемо зі створеним повідомленням існуючу операцію `get_clothes_info()`. Для цього необхідно клікнути правою кнопкою миші на лінії повідомлення та обрати `get_clothes_info()` (рис. 9.24).

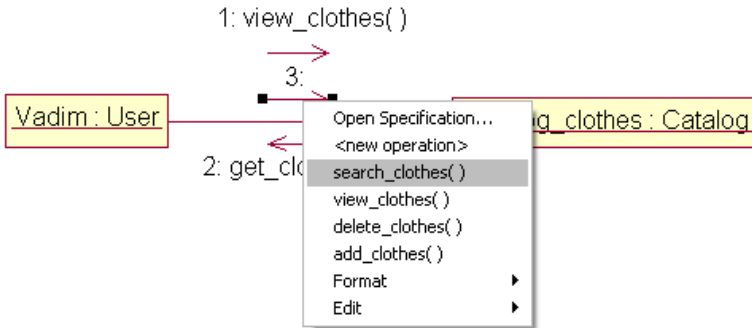


Рисунок 9.22 – Зв’язування операції з повідомленням

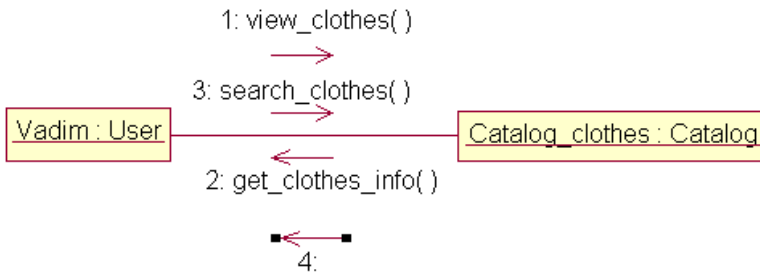


Рисунок 9.23 – Створення зворотного повідомлення

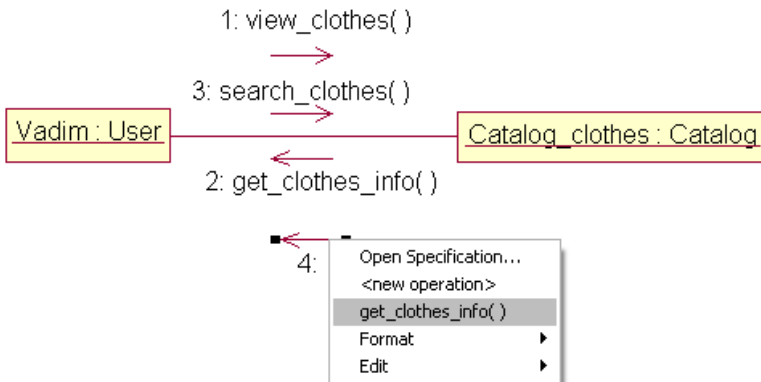


Рисунок 9.24 – Зв’язування операції з повідомленням

13. Змінемо типи стрілок, позначаючих зворотні повідомлення. Для цього оберемо по черзі кожну з двох стрілок та клінемо правою кнопкою по стрілці. У контекстному меню виберемо опцію *Open*

Specification... У діалоговому вікні, що з'явилося, перейдемо на вкладку *Detail*. На цій вкладці потрібно встановити радіо-кнопку біля поля *Return* (рис. 9.25).

14. Результат побудови діаграми кооперації наведено на рис. 9.26.

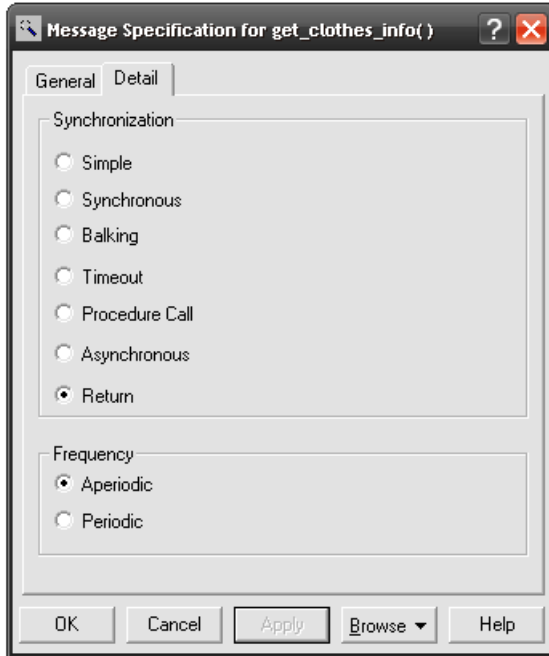


Рисунок 9.25 – Вкладка Detail специфікації повідомлення

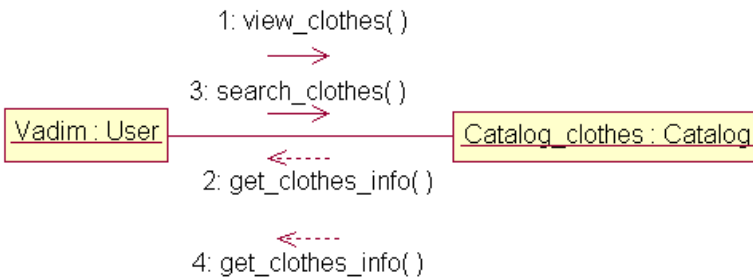


Рисунок 9.26 – Результуюча діаграма кооперації

9.3 Завдання на лабораторну роботу

- 9.3.1 Ознайомитися з пунктом 3.2 і конспектом лекцій.
- 9.3.2 За обраною у лабораторній роботі №1 темою створити ДК.
- 9.3.3 Оформити звіт по роботі.
- 9.3.4 Відповісти на контрольні питання.

9.4 Зміст звіту

- 9.4.1 Тема та мета роботи.
- 9.4.2 Завдання до роботи.
- 9.4.3 Побудована діаграма кооперації.
- 9.4.4 Висновки, за результатами виконання роботи.

9.5 Контрольні запитання

- 9.5.1 Для чого призначені діаграми кооперації?
- 9.5.2 Як перенести дані між діаграмами послідовностей та кооперації?
- 9.5.3 Які інструменти доступні у діаграмі кооперації?
- 9.5.4 Які команди доступні через контекстне меню об'єктів?
- 9.5.5 Які є можливості налаштування області видимості об'єктів?
- 9.5.6 Для чого призначені зв'язки на діаграмі кооперації?
- 9.5.7 Які є стереотипи зв'язків на діаграмі кооперації?
- 9.5.8 Для чого призначені повідомлення на діаграмі кооперації?
- 9.5.9 Які є види повідомлень на діаграмі кооперації?
- 9.5.10 Які є можливості налаштування властивостей повідомлень?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Трофимов С.А. CASE-технологии: практическая работа в Rational Rose. Изд. 2-е. -М.: Бином-Пресс, 2002 г. - 288 с: ил.
2. Дж. Рамбо, М. Блаха. UML 2.0. Объектно-ориентированное моделирование и разработка. 2-е изд. — СПб.: Питер, 2007. — 544 с: ил.
3. Леоненков А. Самоучитель UML [Текст]. — СПб.: БХВ-Петербург, 2004 – 432 с.