




Lesson 22

30.09.2024

```
public class ex1 {  
    public static void main(String[] args) {  
        {  
            for (;;)   
                System.out.println("Java");  
        }  
    }  
}
```

```
public class ex2 {  
    public static void main(String[] args) {  
        try {  
            foo();  
        } catch (Exception ex) {  
            System.out.println("exMain");  
            ex.printStackTrace();  
        }  
    }  
  
    public static void foo() {  
        try {  
            throw new IllegalArgumentException("catch");  
        } finally {  
            try {  
                throw new RuntimeException("finally");  
            } catch (IllegalArgumentException ex) {  
                System.out.println("exFoo");  
                ex.printStackTrace();  
            }  
        }  
    }  
}
```

```
public class ex3 {  
    public static void main(String[] args) {  
        int i = 10 ;  
        System.out.println(i > 3 != false );  
    }  
}
```



```
public class ex4 {  
    public static void main(String[] args) {  
        byte var = 100 ;  
        switch (var) {  
            case 100 :  
                System. out .println( "var is 100" );  
                break ;  
            case 200 :  
                System. out .println( "var is 200" );  
                break ;  
            default :  
                System. out .println( "In default" );  
        }  
    }  
}
```

```
public class ex5 {  
    public static void main(String[] args) {  
        StringBuilder sb = new StringBuilder("TOMATO");  
        System.out.println(sb.reverse()  
            .replace('O', 'A'));  
    }  
}
```

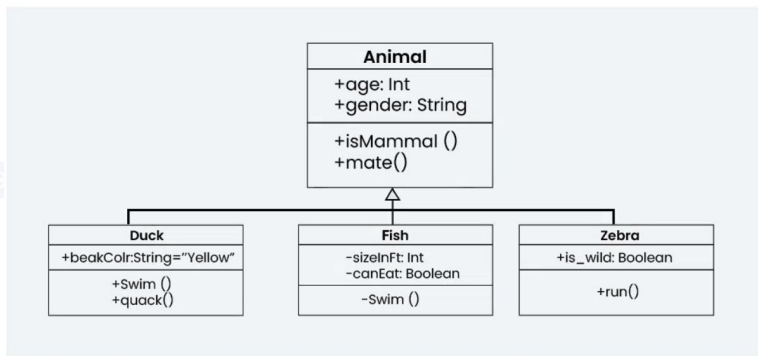


Що таке діаграми класів?

Діаграми класів — це тип діаграми UML (Unified Modeling Language), яка використовується в розробці програмного забезпечення для візуального представлення структури та зв'язків класів у системі, тобто для побудови та візуалізації об'єктно-орієнтованих систем.

На цих діаграмах класи зображені у вигляді коробок, кожна з яких містить три відділення для імені класу, атрибутів і методів. Лінії, що з'єднують класи, ілюструють асоціації, показуючи такі відносини, як «один до одного» або «один до багатьох».

Діаграми класів надають загальний огляд дизайну системи, допомагаючи комунікувати та документувати структуру програмного забезпечення. Вони є фундаментальним інструментом об'єктно-орієнтованого проектування та відіграють вирішальну роль у життєвому циклі розробки програмного забезпечення.



Нотація класу UML

нотація класу - це графічне представлення, яке використовується для зображення класів та їхніх зв'язків в об'єктно-орієнтованому моделюванні.

Позначення класу

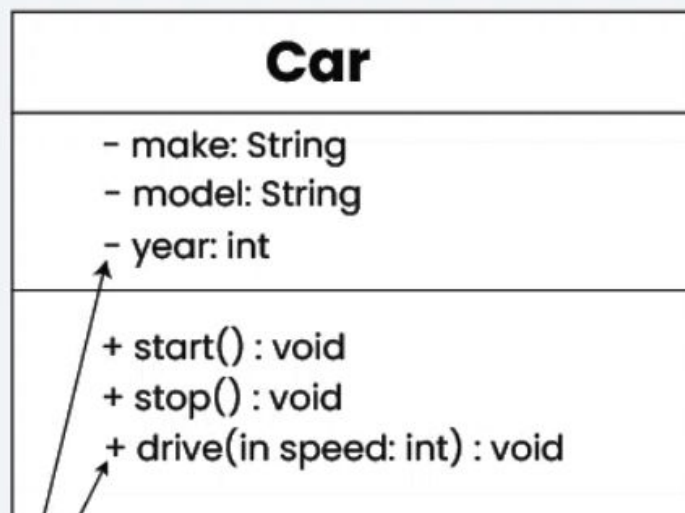
Назва класу - зазвичай пишеться у верхньому відділенні коробки класу, виділяється по центру та жирним шрифтом.

Атрибути - також відомі як властивості або поля, представляють члени даних класу. Вони перераховані у другому відділенні вікна класу та часто включають видимість (наприклад, загальнодоступний, приватний) і тип даних кожного атрибута.

Методи - також відомі як функції або операції, представляють поведінку або функціональність класу. Вони перераховані в третьому відділі вікна класу та включають видимість (наприклад, публічний, приватний), тип повернення та параметри кожного методу.

Позначення видимості- вказують на рівень доступу до атрибутів і методів. Загальні позначення видимості включають:

- + для всіх (видимо для всіх класів)
- для приватних (видно лише в межах класу)
- # для захищених (видимих для підкласів)
- ~ для видимості пакета або за замовчуванням (видимий для класів у тому самому пакеті)



← **Name**

← **Attributes**

← **Operations**

Visibility notation

Відносини між класами

У діаграмах класів зв'язки між класами описують, як класи пов'язані або взаємодіють один з одним у системі. В об'єктно-орієнтованому моделюванні існує кілька типів зв'язків, кожен з яких служить певній меті. Ось кілька поширених типів зв'язків у діаграмах класів:



Composition



Directed Association



Usage(Dependency)



Generalization



Aggregation



Association



Dependency

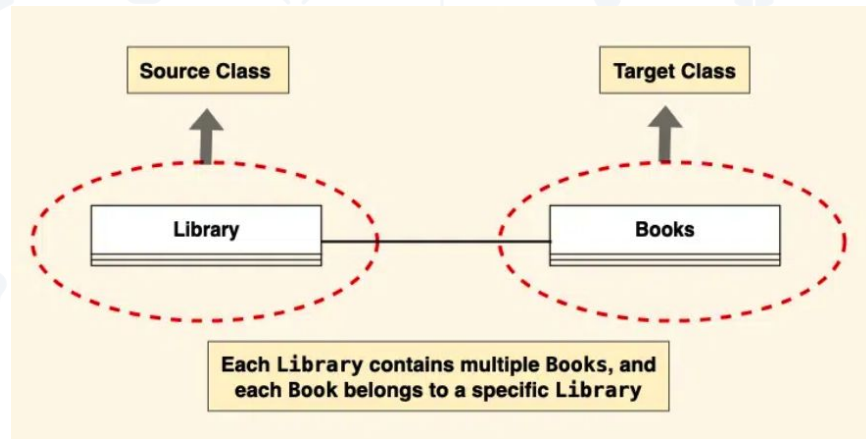


Realization

Асоціація (Association) представляє двонаправлений зв'язок між двома класами. Він вказує на те, що екземпляри одного класу підключені до екземплярів іншого класу. Асоціації зазвичай зображуються суцільною лінією, що з'єднує класи, із необов'язковими стрілками, які вказують напрямок зв'язку.

Розберемо асоціацію на прикладі:

Розглянемо просту систему управління бібліотекою. У цій системі ми маємо дві основні сутності: книгу та бібліотеку. Кожна бібліотека містить декілька книг, і кожна книга належить до певної бібліотеки. Цей зв'язок між бібліотекою та книгою є асоціацією.

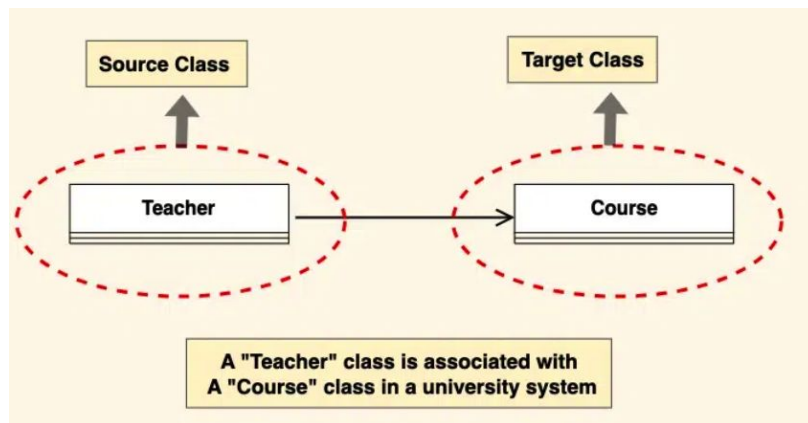


Клас «Library» можна вважати вихідним класом, оскільки він містить посилання на кілька екземплярів класу «Book». Клас «Book» вважатиметься цільовим класом, оскільки він належить до певної бібліотеки.

Спрямована асоціація (directed association) на діаграмі класів UML представляє зв'язок між двома класами, де асоціація має напрямок, вказуючи, що один клас асоціюється з іншим певним чином.

У спрямованій асоціації до лінії асоціації додається стрілка, яка вказує напрямок зв'язку. Стрілка вказує від класу, який ініціює асоціацію, до класу, на який асоціація впливає або на яку впливає. Спрямовані асоціації використовуються, коли асоціація має певний потік або спрямованість, наприклад, вказує, який клас відповідає за ініціювання асоціації або який клас залежить від іншого.

Розглянемо сценарій, коли клас «Вчитель» пов'язаний з класом «Курс» в системі університету. Спрямована стрілка асоціації може вказувати від класу «Вчитель» до класу «Курс», вказуючи, що викладач пов'язаний із певним курсом або викладає його.

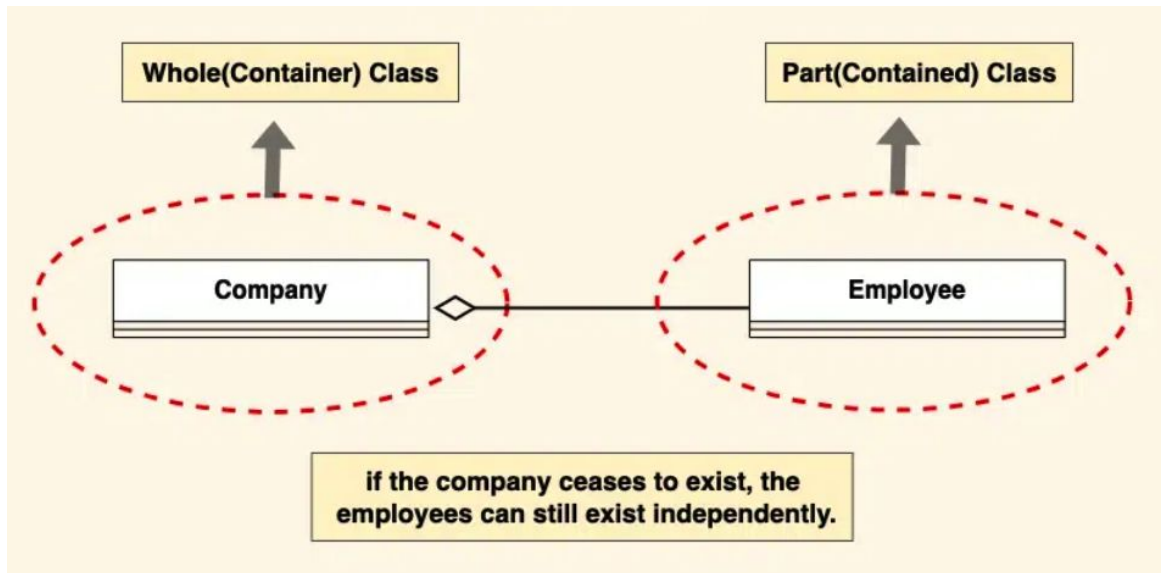


Вихідним класом є клас «Вчитель». Клас «Вчитель» ініціює об'єднання, викладаючи певний курс. Цільовий клас – клас «Курс». На клас «Курс» впливає асоціація, оскільки його викладає певний викладач.

Агрегація (Aggregation) — це спеціалізована форма асоціації, яка представляє зв'язок «ціле-частина». Це означає сильніший зв'язок, коли один клас (ціле) містить або складається з іншого класу (частини). Агрегація представлена ромбовидною формою збоку від усього класу. У такому вигляді відносин дочірній клас може існувати незалежно від свого батьківського класу.

Давайте розберемо агрегацію на прикладі:

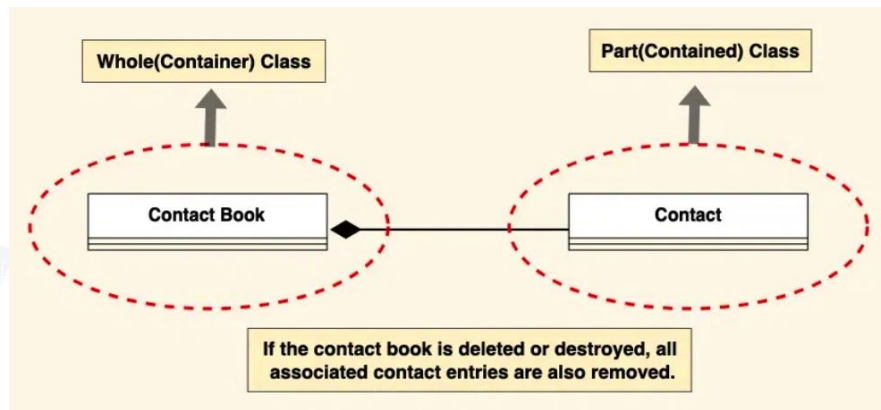
Компанію можна розглядати як єдине ціле, а співробітників – її частини. Співробітники належать компанії, і компанія може мати кількох працівників. Однак, якщо компанія припиняє своє існування, працівники можуть існувати самостійно.



Композиція (Composition) є сильнішою формою агрегації, що вказує на більш значущі відносини власності чи залежності. У складі клас частини не може існувати незалежно від цілого класу. Композиція представлена заповненим ромбом збоку від усього класу.

Розберемо композицію на прикладі:

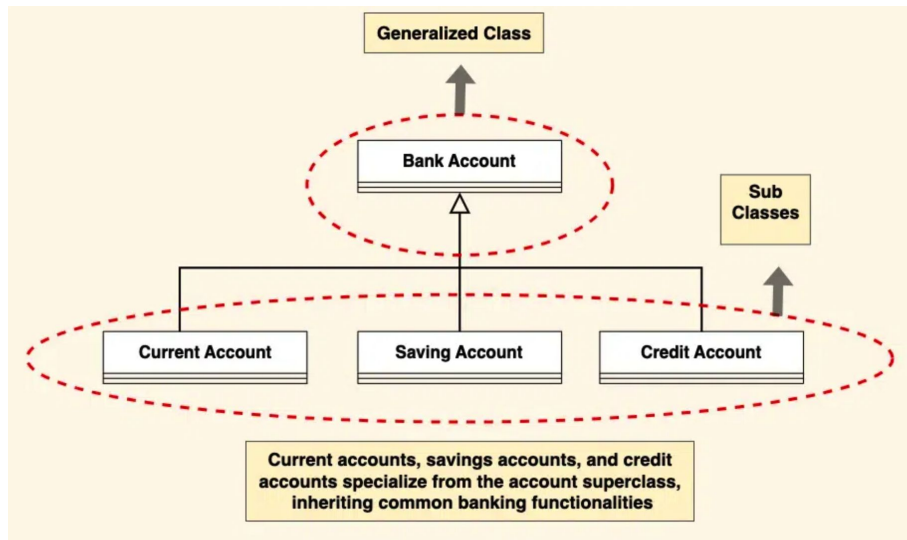
Уявіть собі програму цифрової контактної книги. Книга контактів - це ціле, і кожен запис контакту є частиною. Кожен запис контакту повністю належить і керується книгою контактів. Якщо книгу контактів видалено або знищено, усі пов'язані записи контактів також видаляються.



Це ілюструє композицію, оскільки існування контактних записів повністю залежить від наявності контактної книги. Без контактної книги окремі контактні записи втрачають сенс і не можуть існувати самі по собі.

Спадкування(Inheritance) представляє зв'язок "is-a" між класами, де один клас (підклас або дочірній) успадковує властивості та поведінку іншого класу (суперкласу або батьківського). Спадкування зображується суцільною лінією із закритою порожнистою стрілкою, що вказує від підкласу до суперкласу.

У прикладі банківських рахунків ми можемо використовувати узагальнення для представлення різних типів рахунків, таких як поточні рахунки, ощадні рахунки та кредитні рахунки.



Клас Bank Account служить узагальненим представленням усіх типів банківських рахунків, тоді як підкласи (Current Account, Savings Account, Credit Account) представляють спеціалізовані версії, які успадковують і розширюють функціональні можливості базового класу.

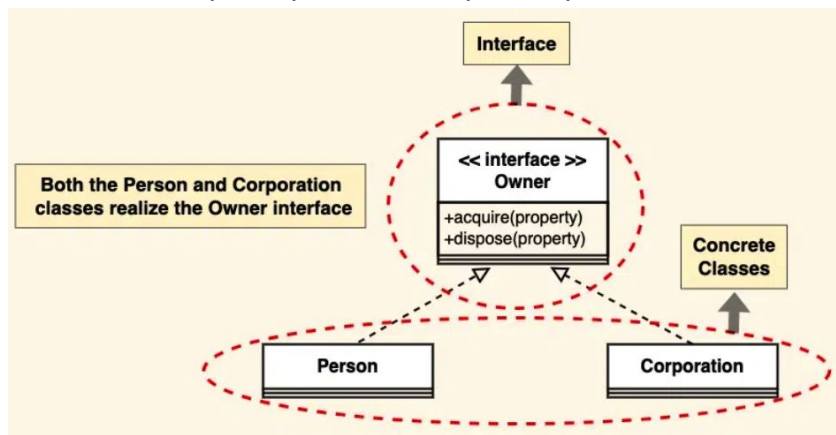
Реалізація (realization) вказує на те, що клас реалізує функції інтерфейсу. Він часто використовується у випадках, коли клас реалізує операції, визначені інтерфейсом. Реалізація зображена пунктирною лінією з відкритою стрілкою, що вказує від класу реалізації до інтерфейсу.

Давайте розглянемо сценарій, де «Особа» і «Корпорація» реалізують інтерфейс «Власник».

Інтерфейс власника: цей інтерфейс тепер включає такі методи, як «acquire(property)» і «dispose(property)», щоб представляти дії, пов'язані з придбанням і розпорядженням власністю.

Клас Person (реалізація): клас Person реалізує інтерфейс Owner, надаючи конкретні реалізації для методів «acquire(property)» і «dispose(property)». Наприклад, особа може придбати у власність будинок або розпорядитися автомобілем.

Клас Corporation (Реалізація): Подібним чином клас Corporation також реалізує інтерфейс власника, пропонуючи конкретні реалізації для методів «acquire(property)» і «dispose(property)». Наприклад, корпорація може придбати право власності на нерухоме майно або розпоряджатися транспортними засобами компанії.

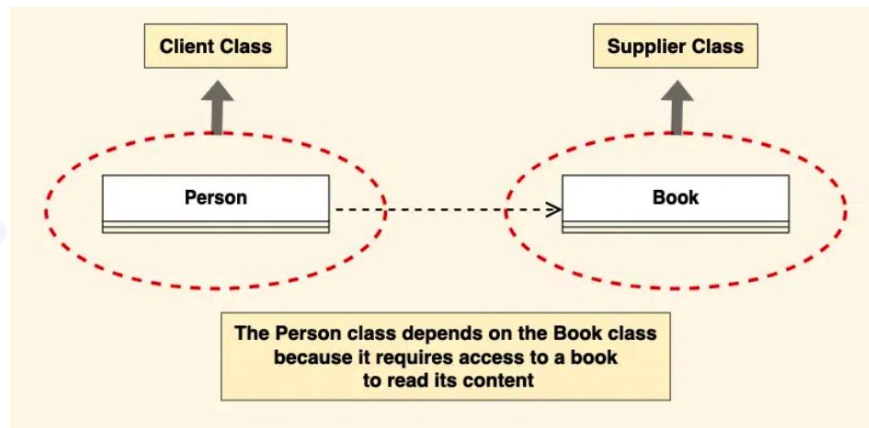


Залежність(dependency) існує між двома класами, коли один клас покладається на інший, але зв'язок не такий міцний, як асоціація чи спадкування. Він представляє більш слабкий зв'язок між класами. Залежності часто зображуються пунктирною стрілкою.

Давайте розглянемо сценарій, коли Людина залежить від Книги.

Клас Person: представляє особу, яка читає книгу. Клас Person залежить від класу Book для доступу та читання вмісту.

Клас книги: представляє книгу, яка містить вміст для читання Person. Клас Book є незалежним і може існувати без класу Person.



Клас Person залежить від класу Book, оскільки йому потрібен доступ до книги, щоб прочитати її вміст. Однак клас Book не залежить від класу Person; він може існувати незалежно і не покладається на клас Person для своєї функціональності.

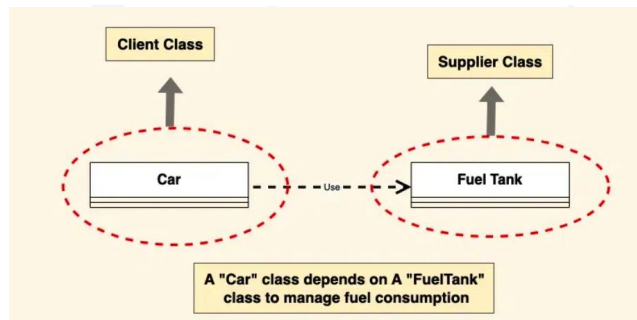
Відношення залежності (usage dependency) використання на діаграмі класів UML вказує на те, що один клас (клієнт) використовує або залежить від іншого класу (постачальника) для виконання певних завдань або доступу до певної функціональності. Клас клієнта покладається на послуги, що надаються класом постачальника, але не володіє та не створює його екземпляри.

Залежності використання представляють форму залежності, коли один клас залежить від іншого класу для задоволення певної потреби чи вимоги.

Клас клієнта потребує доступу до певних функцій або послуг, що надаються класом постачальника.

У діаграмах класів UML залежності використання зазвичай представлені пунктирною лінією зі стрілкою, що вказує від класу клієнта до класу постачальника.

Стрілка вказує напрямком залежності, показуючи, що клас клієнта залежить від послуг, які надає клас постачальника.



Класу «Автомобіль» може знадобитися доступ до методів або атрибутів класу «FuelTank», щоб перевірити рівень палива, долити паливо або контролювати споживання палива. У цьому випадку клас «Автомобіль» має залежність використання від класу «FuelTank», оскільки він використовує його служби для виконання певних завдань, пов'язаних із керуванням паливом.

Sequence Diagrams

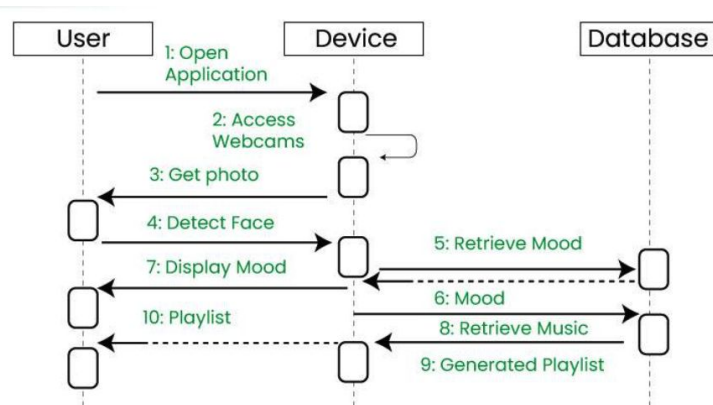




Діаграма взаємодії

Діаграма взаємодії використовується, щоб показати інтерактивну поведінку системи. Оскільки візуалізація взаємодії в системі може бути складною, ми використовуємо різні типи діаграм взаємодії, щоб відобразити різні особливості та аспекти взаємодії в системі.

- Діаграма послідовності просто зображує взаємодію між об'єктами в послідовному порядку, тобто порядку, в якому відбуваються ці взаємодії.
- Ми також можемо використовувати терміни діаграми подій або сценарії подій для позначення діаграми послідовності.
- Діаграми послідовності описують, як і в якому порядку функціонують об'єкти в системі.
- Ці діаграми широко використовуються бізнесменами та розробниками програмного забезпечення для документування та розуміння вимог до нових та існуючих систем.

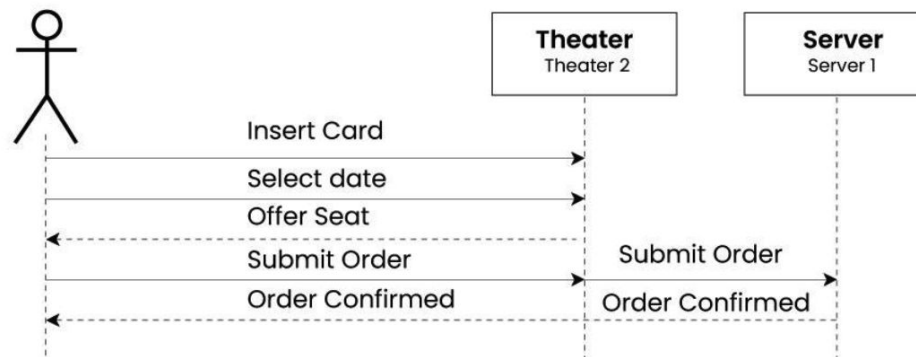


Актор (actor) на діаграмі UML представляє тип ролі, де він взаємодіє з системою та її об'єктами. Тут важливо зазначити, що актор завжди знаходиться поза межами системи, яку ми прагнемо змодельювати за допомогою діаграми UML.

Ми використовуємо акторів для зображення різних ролей, включаючи користувачів та інших зовнішніх суб'єктів. Ми представляємо актора на діаграмі UML, використовуючи нотацію особи. На діаграмі послідовності може бути кілька акторів.

Наприклад:

User interacting with seat reservation system



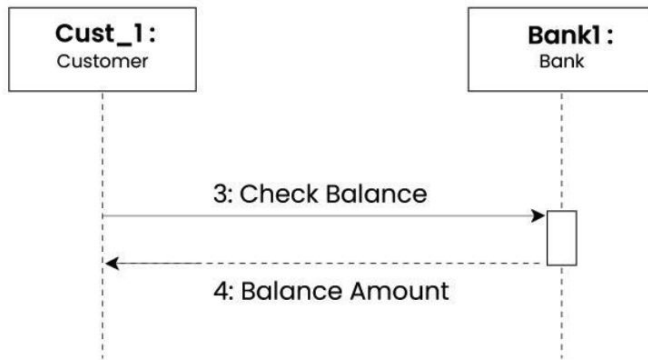
Тут користувач у системі бронювання місць показаний як актор, якщо він існує поза системою і не є частиною системи.


Лінія життя (lifeline) — це іменований елемент, який зображує окремого учасника на діаграмі послідовності. Отже, в основному кожен екземпляр на діаграмі послідовності представлений лінією життя. Елементи лінії життя розташовані у верхній частині діаграми послідовності. Стандарт UML для іменування лінії життя має наступний формат

Ми покажемо лінію життя в прямокутнику під назвою голова з її назвою та типом. Голова розташована на верхній частині вертикальної пунктирної лінії (називається ніжкою), як показано вище.

Якщо ми хочемо змодельювати екземпляр без назви, ми дотримуємося того самого шаблону, за винятком того, що тепер частина назви lifeline залишається порожньою.

Sequence Diagram





Повідомлення (messages) зв'язок між об'єктами зображується за допомогою повідомлень. Повідомлення з'являються в послідовному порядку на лінії життя.

Зображуємо повідомлення за допомогою стрілок.

Лінії життя та повідомлення складають ядро діаграми послідовності.

Різні-типи-повідомлень

Повідомлення можна розділити на такі категорії:

Синхронні повідомлення

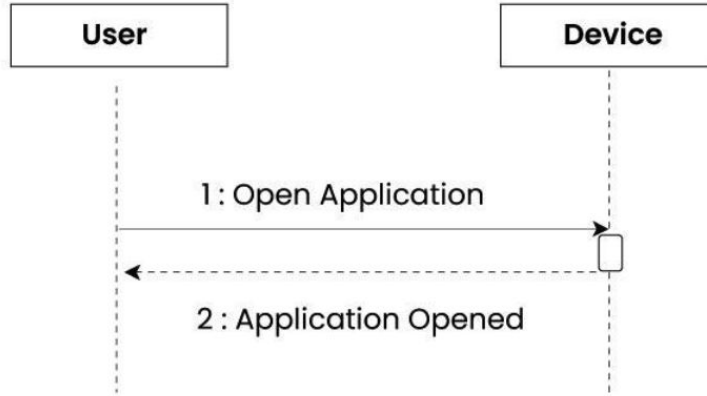
Синхронне повідомлення очікує на відповідь, перш ніж взаємодія зможе продовжитися. Відправник чекає, поки одержувач завершить обробку повідомлення. Абонент продовжує лише тоді, коли він знає, що одержувач обробив попереднє повідомлення, тобто він отримує повідомлення у відповідь.

Асинхронні повідомлення

Асинхронне повідомлення не чекає відповіді від отримувача. Взаємодія продовжується незалежно від того, обробив одержувач попереднє повідомлення чи ні. Ми використовуємо наконечник стрілки, щоб представити асинхронне повідомлення.



Synchronous Message



Asynchronous Message



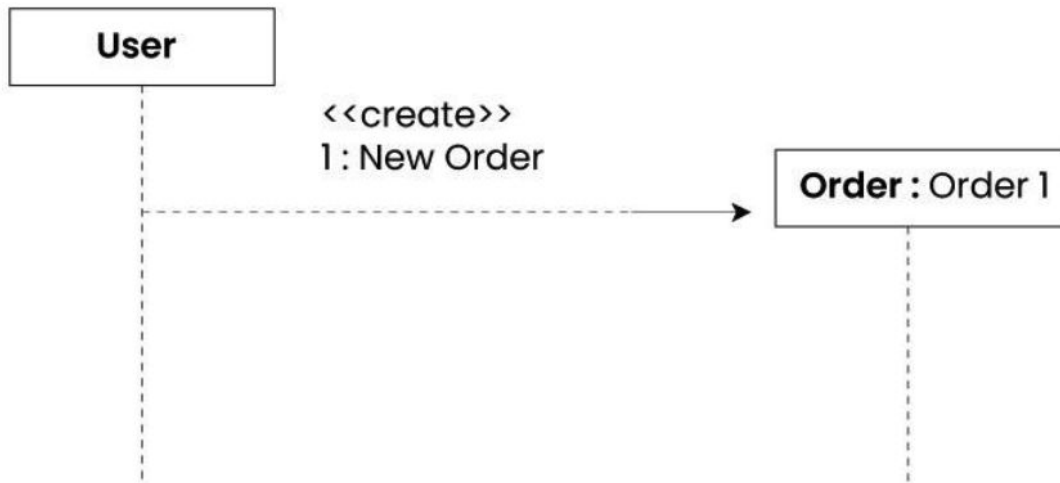
Створити повідомлення (Create message)

Ми використовуємо повідомлення Create для створення нового об'єкта на діаграмі послідовності. Бувають ситуації, коли певний виклик повідомлення вимагає створення об'єкта. Він представлений пунктирною стрілкою та словом створення, позначеним на ньому, щоб вказати, що це символ створення повідомлення.

Наприклад:

Створення нового замовлення на веб-сайті електронної комерції вимагатиме створення нового об'єкта класу Order.

Create Message



Видалити повідомлення (Delete Message)

Ми використовуємо Delete Message, щоб видалити об'єкт. Коли об'єкт звільняється від пам'яті або знищується в системі, ми використовуємо символ «Видалити повідомлення». Він знищує наявність об'єкта в системі. Він представлений стрілкою, що закінчується х.

Наприклад:

У наведеному нижче сценарії, коли користувач отримує замовлення, об'єкт класу замовлення може бути знищено.

Delete Image





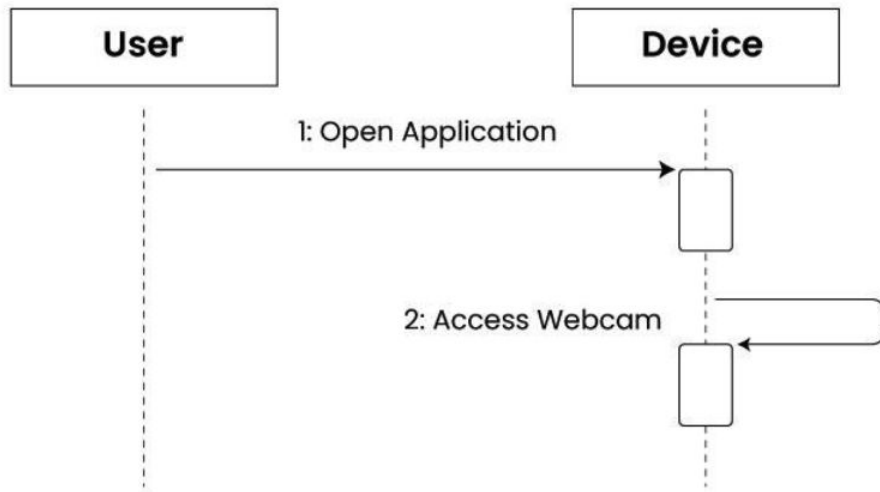
Самоповідомлення (Self Message)

Можуть виникнути певні сценарії, коли об'єкт повинен надіслати повідомлення самому собі. Такі повідомлення називаються власними повідомленнями та позначаються стрілкою у формі U.

Приклад:

Розглянемо сценарій, коли пристрій хоче отримати доступ до своєї веб-камери. Такий сценарій представлено за допомогою власного повідомлення.

Self Image



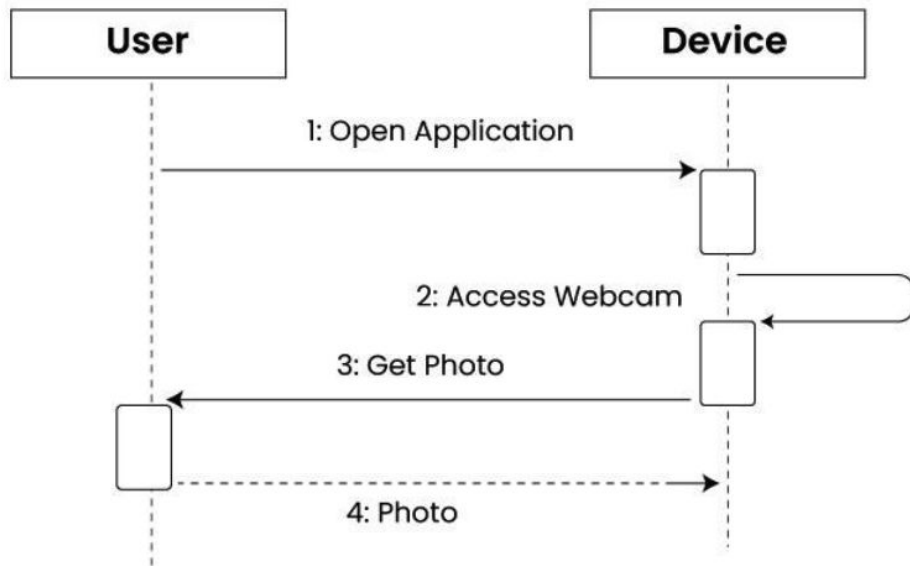


Повідомлення-відповіді (Reply messages) використовуються, щоб показати повідомлення, яке надсилається від одержувача до відправника. Ми представляємо повідомлення повернення/відповіді за допомогою відкритої стрілки з пунктирною лінією. Взаємодія продовжується лише тоді, коли одержувач надсилає повідомлення-відповідь.

Наприклад:

Розглянемо сценарій, коли пристрій запитує фотографію від користувача. Тут повідомлення, яке показує надіслане фото, є повідомленням-відповіддю.

Reply Message Example



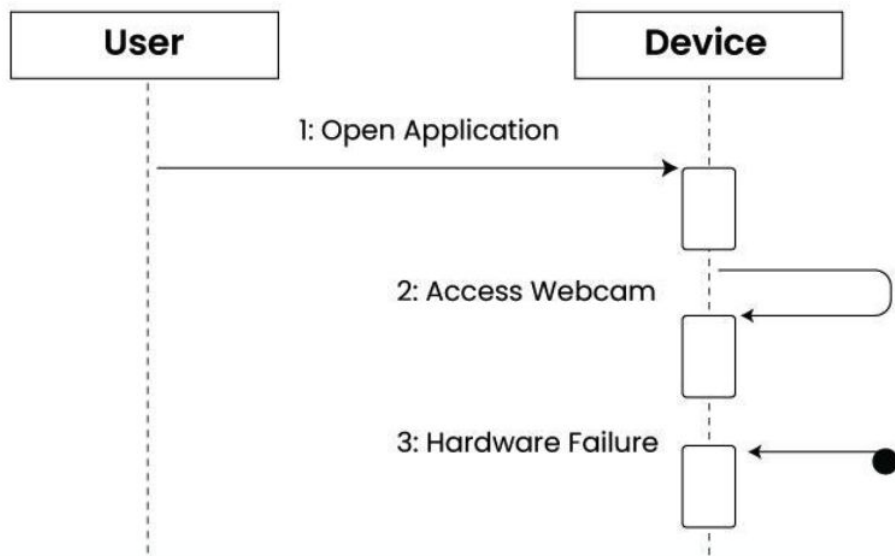
Повідомлення «Знайдено» (Found Message) використовується для представлення сценарію, коли невідоме джерело надсилає повідомлення. Він представлений за допомогою стрілки, спрямованої до лінії життя від кінцевої точки.

Наприклад:

Розглянемо сценарій апаратного збою.

Це може бути з кількох причин, і ми не впевнені, що спричинило збій обладнання.

Found Message Example





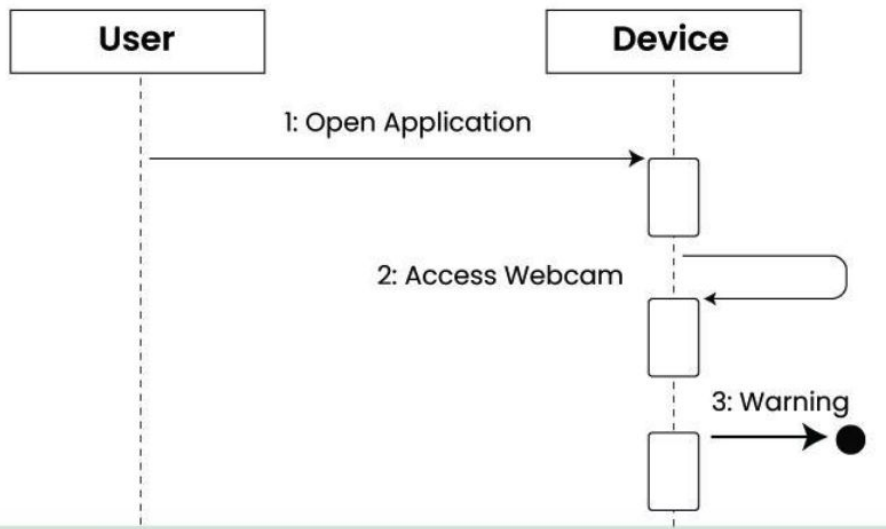
Втрачене повідомлення (lost message) використовується для представлення сценарію, коли одержувач не відомий системі. Він представлений за допомогою стрілки, спрямованої до кінцевої точки від лінії життя.

Наприклад:

Розглянемо сценарій, коли генерується попередження.

Попередження може бути згенеровано для користувача або іншого програмного забезпечення/об'єкта, з яким взаємодіє лінія життя. Оскільки адресат не відомий заздалегідь, ми використовуємо символ «Втрачене повідомлення».

Lost Image Example





Для моделювання умов ми використовуємо захисники в UML. Вони використовуються, коли нам потрібно обмежити потік повідомлень під приводом виконання умови. Охоронці відіграють важливу роль, повідомляючи розробникам програмного забезпечення про обмеження, пов'язані з системою чи певним процесом.

Наприклад:

Для того, щоб мати можливість зняти готівку, наявність балансу більше нуля є умовою, яка має бути виконана, як показано нижче.

Guards

