



{**PROGRAMMING**}



**UNITED  
MODELING  
LANGUAGE**

# Урок №3

Діаграма класів,  
діаграма станів,  
діаграма активності

## ЗМІСТ

<b>Вступ .....</b>	<b>4</b>
<b>1. Діаграма класів .....</b>	<b>6</b>
1.1. Цілі даного типу діаграм.....	6
1.2. Базові поняття.....	9
1.2.1. Клас .....	9
1.2.2. Об'єкт .....	13
1.2.3. Інтерфейс.....	14
1.2.3. Шаблон.....	16
1.3. Зв'язки між класами .....	18
1.3.1. Зв'язок залежності .....	19
1.3.2. Зв'язок асоціації.....	21
1.3.3. Зв'язок узагальнення.....	23
1.3.4. Зв'язок реалізації.....	24
1.4. Порівняльний аналіз зв'язків.....	25

1.5. Практичні приклади побудови діаграм класів...	26
<b>2. Діаграма станів .....</b>	<b>29</b>
2.1. Цілі даного типу діаграм.....	29
2.2. Базові поняття.....	29
2.2.1. Автомати .....	29
2.2.2. Стан.....	30
2.2.3. Перехід .....	32
2.3. Складений стан і підстан.....	36
2.3.1. Послідовні підстани.....	37
2.3.2. Паралельні підстани .....	38
2.4. Історичний стан.....	38
2.5. Складні переходи .....	40
2.5.1. Переходи між паралельними станами .....	40
2.5.2. Переходи між складеними станами.....	40
2.5.3. Синхронізуючі стани.....	41
2.6. Практичні приклади побудови діаграм станів .....	42
<b>3. Діаграма активності.....</b>	<b>44</b>
3.1. Цілі даного типу діаграм.....	44
3.2. Базові поняття.....	44
3.2.1. Стан дії.....	44
3.2.2. Переходи.....	46
3.2.3. Доріжки.....	48
3.2.4. Об'єкти.....	50
3.3. Практичні приклади побудови діаграм діяльності.....	52

# Вступ

Розпочнемо з того, що пригадаємо основні етапи розробки програмного забезпечення. Взагалі, розробка складається з:

- аналізу проблеми і планування;
- моделювання;
- реалізації, тестування і документування;
- впровадження і супроводу.

Нас на даному етапі особливо цікавитимуть аналіз проблеми, планування і моделювання.

Розробка починається з етапу аналізу проблеми, поставленої перед групою розробників. На цьому етапі формується термінологія процесу вирішення проблеми, а також виділяються основні поняття, якими будуть оперувати процеси, що протікають в інформаційній системі, яка розробляється в рамках вирішення поставленої проблеми.

Вважаємо важливим зазначити, що будь-яке програмне забезпечення — це інформаційна система, а це означає насамперед, що це система; і, отже, розглядати її необхідно в контексті термінології теорії систем. Це спростить розуміння викладеного в уроці матеріалу.

Після того, як були виділені базові поняття та процеси, починається етап моделювання системи і її компонентів, які у більш детальному розгляді, як правило, теж являють собою системи.

І якщо вже йдеться про моделі, вважаємо за необхідне нагадати, що моделі за складністю поділяються на

глобальні і базові, а за ступенем деталізації — на узагальнені і деталізовані.

Етап моделювання передбачає складання як узагальнених моделей глобальних систем та процесів, так і деталізованих моделей базових елементів і активностей (під активністю розуміється виконання системою деякого алгоритму, яка зазвичай представляється у вигляді блок-схеми).

Моделювання важливе тому, що воно дозволяє всім членам команди розробників однаково надати кінцевий результат, представлений архітектором у вигляді UML-моделей. А аналіз важливий у тому сенсі, що дозволяє виділити та сформулювати поняття, які розробники використовуватимуть, описуючи аналіз і прийняття рішень в інформаційній системі.

Система понять (типів), якими оперує інформаційна система, зазвичай виявляється як структурна модель додатку, зазвичай представленої як сукупність діаграм класів, яким і буде присвячена наступна глава.

# 1. Діаграма класів

Діаграми класів — це спосіб представлення у UML внутрішньої структури класу та інтерфейсу взаємодії з ним. Нагадаємо, що під структурою маються на увазі поля класу. Тобто, це ті дані, які клас інкапсулює, а під інтерфейсом — усі загальнодоступні (public) методи, що визначають способи взаємодії з об'єктами цього класу. Також, завдяки тому, що в діаграмі класів передбачена можливість специфікації доступності атрибуту, існує можливість відобразити приховані перетворення, обумовлені класом — ті методи, які не доступні ззовні, але пояснюють внутрішню поведінку класу.

Діаграма класів є статичне представлення моделі додатка з погляду архітектора (особи, яка відповідає за проектування системи).

## 1.1. Цілі даного типу діаграм

Виділяють такі цілі використання діаграм класів:

- **Опис моделі системи типів, що використовується додатком.** Інакше модель системи типів ще називають словником системи, оскільки вона становить понятійну базу, використовувану системою для опису процесу, який вона підтримує;
- **Відображення простих кооперацій, які мають місце в модельованій системі.** Під коопераціями розуміється сукупності класів, інтерфейсів і т.д., сумісно функціонуючих для реалізації певної кооперативної поведінки.

Цю мету можна вважати найважливішою з погляду моделювання систем, оскільки вона відбиває емерджентність — найважливішу властивість, притаманне системам. Емерджентність — це загальна властивість систем, яке визначає, що система наділена деякими властивостями, які не належать до властивостей складових її елементів.

Розглядаючи окремі класи, які входять до кооперації, ми не бачимо тієї функції, яку виконує кооперація. Якщо розглядати кооперацію узагальнено, у її цілісності, ми знаходимо ту властивість, яку вона реалізує і, відповідно, вносить у кінцеву систему

- **Відображення структурного аспекту організації даних у додатку.** Також, діаграма класів може використовуватися для опису логічної схеми бази даних, використовуваної інформаційною системою.

За Мартіном Фаулером, існують три основні точки зору, з яких необхідно розглядати будь-яку модель, особливо діаграми класів, тобто структурну модель системи:

- **Концептуальна точка.** З концептуальної точки зору, діаграми класів слугують для «уявлення понять предметної області, що вивчається». Однак, внаслідок того, що часто реалізація досить сильно відрізняється від моделі, кінцеві типи даних (класи) можуть зовсім не відповідати поняттям, якими оперує архітектор при проектуванні системи. А це означає, що концептуальна модель може мати слабе (або зовсім не мати) відношення до кінцевого програмного забезпечення.

- **Точка зору специфікації.** З точки зору специфікації, значення має інтерфейс інформаційної системи і його реалізація. Але об'єктно-орієнтоване проектування розглядає лише інтерфейси програмної системи, у відриві від реалізації. Взагалі, коли говорять про проектування класу, переважно мають на увазі проектування інтерфейсу класу, оскільки з погляду кінцевого використання важлива не внутрішня структура об'єктів, а їх поведінка. Адже, погодьтеся, абсолютно не важливими будуть структурні відмінності між двома реалізаціями класу, якщо результат їхньої поведінки залишатиметься однаковим.
- **Точка зору реалізації.** З цього погляду ми проектуємо класами, відповідно, важливими є всі властивості притаманні класам як лексичним конструкціям певної мови програмування, оскільки ми «спускаємося» на «рівень реалізації». Ця точка зору найбільш наближена до реального програмного продукту. Однак, для проектування точка зору специфікації більш адекватна і переважна.

Описані вище точки зору можна умовно порівняти з різними етапами процесу розробки програмного забезпечення і визначити як рівні процесу розробки. Так, на етапі аналізу та планування ми розглядаємо інформаційну систему з концептуальної точки зору і визначаємо цей етап розробки як концептуальний рівень; на етапі моделювання (проектування) — з погляду специфікації, і визначаємо його як рівень моделювання; а на етапі розробки — з погляду реалізації, і визначаємо його як рівень реалізації.



## 1.2. Базові поняття

### 1.2.1. Клас

Клас є базовим поняттям об'єктно-орієнтованого підходу в програмуванні, а отже, це поняття має використовуватися в об'єктно-орієнтованому аналізі і проектуванні.

Як уже зрозуміло з вище зазначеного, будь-яке поняття мови моделювання UML залежить від рівня, де він використовується. Таким чином, на концептуальному рівні клас є одним з понять, яке використовується для опису проблемної області, в рамках якої вирішується поставлене перед проектувальниками завдання. На рівні моделювання — являє собою тип даних, а на рівні реалізації — формальний тип даних, описаний деякою об'єктно-орієнтованою мовою програмування.

Але, для успішного оперування поняттям класу необхідно одночасно мати на увазі усі три аспекти, в яких він може існувати, тобто, — сприймати поняття класу в його цілісності.

#### *Атрибут*

Спрощено можна сказати, що атрибут — це деякі дані, які зберігаються класом і певним чином його характеризують. Наприклад, візьмемо поняття посади. Необхідно, аби посада мала певну назву та зіставлялася з деякою стартовою (мінімальною) зарплатою. Таким чином, до поняття посади входять такі атрибути: «назва», — повідомляє нам назву посади, і «мінімальна заробітна плата», — повідомляє стартовий оклад працівника, який перебуває на цій посаді.

На концептуальному рівні, наявність атрибуту «назва» в понятті «посада», говорить нам про назву посади. На рівні специфікації ми розуміємо, що об'єкт-тип «посада» може повідомити нам і свою назву, а також, можливо, отримати від нас нову назву, якщо це передбачено концепцією. На рівні реалізації ми розуміємо, що клас `Position` буде мати поле `Name`.

Синтаксично, визначення атрибуту виглядає так:

```
видимість назва_атрибуту: тип =  
                                значення_за_замовчуванням
```

Синтаксис визначення атрибуту записаний у загальному вигляді. Тому, нижче ми вкажемо, як фактично описується кожен елемент:

- **видимість** вказується з використанням символів «+», «#» і «-», де «+» означає, що атрибут має специфікатор `public`, «#» — `protected`, а «-» — `private`;
- **назва\_атрибуту** вказується у вигляді послідовності символів;
- **тип** передбачає специфікацію назви типу атрибуту;
- як значення за замовчуванням, використовують вказівку фактичного значення, яке встановлюватиметься атрибуту як замовчуване. Як правило, значення по замовчуванню вказують атрибутам базових типів даних.

Нижче наведено приклад опису атрибутів для класу «посада»

```
MinimalSalary: decimal  
Name: string
```

Далі наведено графічне уявлення класу **Position** в діаграмі класів, або, іншими словами, — зображення, яке ілюструє, як виглядатиме опис класу **Position** (посада) мовою UML у діаграмі класів.

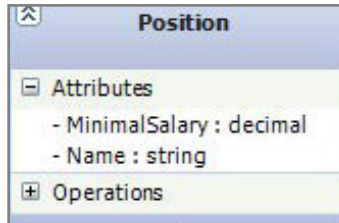


Рисунок 1.

### Операція

Під операціями маються на увазі процеси, які (реалізує) виконує певний клас. Зазвичай, операції, зумовлені на концептуальному рівні, відповідають методам класу лише на рівні специфікації.

Синтаксис визначення операції мовою UML виглядає наступним чином:

```
видимість назва операції(список аргументів):
тип_повернення_значення{властивості_операції}
```

- **видимість операцій** визначається так само, як і для атрибутів;
- **назва\_операції** визначається у вигляді послідовності символів;
- **список\_аргументів** складається з розділених комою параметрів, описаних наступним чином:

```
напрямок назва_параметра: тип = значення за замовчуванням
```

де напрямок вказує, як аргумент використовується операцією, і може приймати такі значення: (**in**) — вхідний аргумент, (**out**) — вихідний аргумент, тобто той, який ініціалізується у процесі операції; (**inout**) — аргумент використовується операцією в обох напрямках.

Якщо напрямок не вказано, то мається на увазі значення (**in**), тому найчастіше воно опускається.

- **тип повернення\_значення** має вигляд списку типів даних, розділених комою. Але більшість розробників використовують тільки один тип даних.
- **властивості\_операції** представлені у вигляді списку з властивостей, розділених комою, що застосовуються до операції.

Нижче наведено приклад опису операцій для типу даних «посада»

```
+ GetMinSalary() : decimal
+ GetName() : string
+ Position(Name : string, MinSalary: decimal)
```

Малюнок представлений далі, ілюструє те, як виглядатиме оголошення операцій для типу даних «посада» в реальній UML діаграмі.

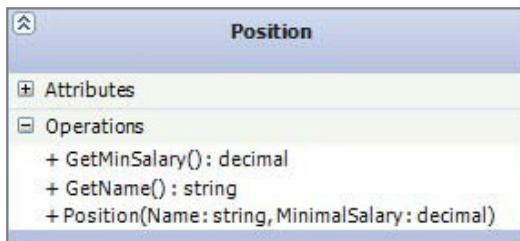


Рисунок 2.

### 1.2.2. Об'єкт

На концептуальному рівні і рівні моделювання, під об'єктом розуміють екземпляр певного класу. Інакше кажучи, якщо клас визначає певне поняття про деякий об'єкт предметної області, то поняття «об'єкт» являє собою цей об'єкт, який ми описуємо використовуючи клас.

На рівні реалізації, під об'єктом розуміють діючий (створений у пам'яті) екземпляр певного класу.

В UML передбачено такий вид діаграм, як діаграма об'єктів. Діаграма об'єктів не є класичною діаграмою для UML, але в деяких випадках використовується для уточнення способу використання класів у тих чи інших контекстах, а також для уявлення функціонального призначення класу в загальній системі.

Нижче представлена діаграма об'єктів, у якій фігурують три класи: Discipline, Teacher і Methodist. Діаграма ілюструє, що один вчитель може викладати декілька предметів і виконувати з предметом лише одну дію, тобто викладати його. Тоді як плануванням предмета займається методист. Інакше кажучи, на діаграмі концептуальне обмеження, яке вносить проектувальник і яке виражається у розподілі обов'язків між викладачем та методистом.

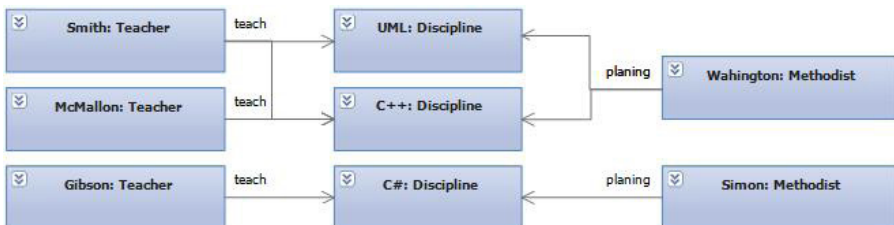


Рисунок 3.

На діаграмі видно, що при описі об'єкту, після назви через символ «:», вказано тип об'єкта. Таким чином, завжди можна відрізнити опис класу від опису об'єкта.

```
назва об'єкта: назва_шаблону
<назва_параметру_шаблону = значення>
```

Також, об'єкти можуть вноситися до діаграм класів для того, щоб описати константні об'єкти, які використовуються іншими класами.

### 1.2.3. Інтерфейс

З одного боку, під інтерфейсом ми розуміємо набір загальнодоступних (**public**) операцій в класі, які у сукупності представляють різні способи використання класу.

З іншого боку, інтерфейс — це ще одна категорія мови UML, який містить опис функціональності певного поняття. Звідси випливає, що інтерфейс — це іменований набір відкритих якостей, уявлений як тип. Основна ідея, яка визначає існування інтерфейсів — це внесення поділу між описом, функціональністю та реалізацією.

Синтаксично, опис інтерфейсу на UML практично не відрізняється від опису класу, за одним винятком: при описі інтерфейсу, за допомогою ключового слова «**interface**», вказується, що описується в інтерфейсі.

Важливо пам'ятати, що інтерфейс містить лише опис можливостей, але не включає жодної реалізації, а отже, від нього не можна створювати об'єкти.

В наведеному далі прикладі (рис. 4) оголошено два інтерфейси: **IDrawable** і **IFigure**, які успадковуються двома

класами: **Rectangle** і **Ellipse**. У такій ситуації прийнято вважати, що описані класи перебувають у зв'язку наслідування (у зв'язку спорідненості) з описаними інтерфейсами.

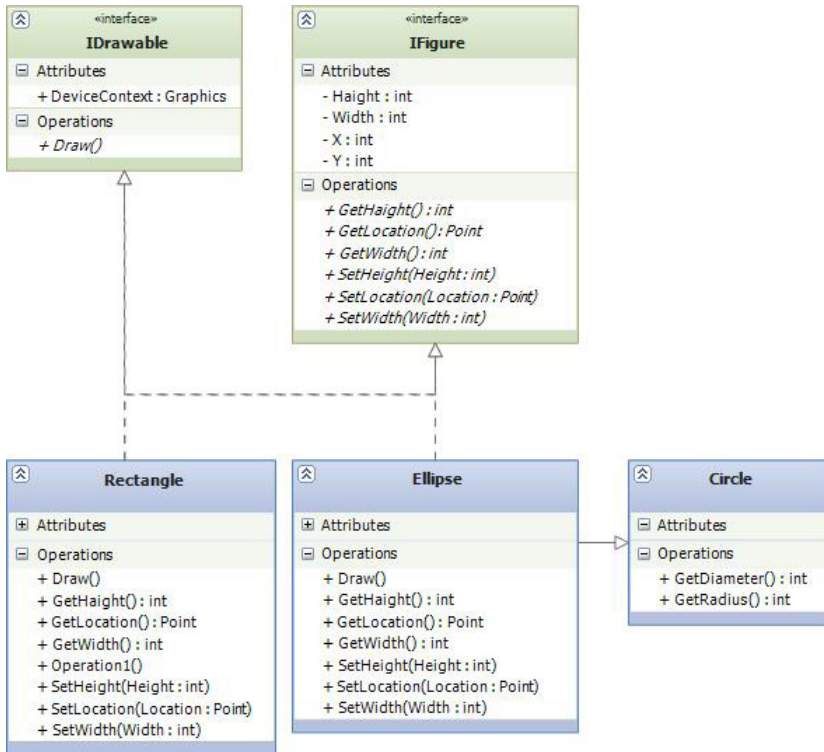


Рисунок 4.

При цьому функціональність, зазвичай, повністю описується інтерфейсами, а похідні класи містять лише реалізацію. Але така ситуація не завжди можлива. Іноді буває необхідно, щоб клас, який реалізує інтерфейс, містив додаткову функціональність, властиву лише для цього класу. Наприклад, методи необхідні для обчислення діаметра і радіуса кола не можна помістити в інтерфейс, оскільки

вони не потрібні у класах «прямокутник» та «еліпс», і їх наявність у цих класах не є доцільною. Виходячи з цього, ми визначаємо коло як окремий випадок еліпса і додаємо описаному класу «коло» (*Circle*) необхідну функціональність, а основну функціональність побічно успадковуємо від інтерфейсу **IFigure** через клас **Ellipse**.

На рівні реалізації у більшості об'єктно-орієнтованих мов програмування для опису інтерфейсів існує спеціальна однойменна мовна конструкція. У тих мовах, у яких такої конструкції немає, для здійснення поділу функціональності та реалізації використовують абстрактні класи.

### 1.2.3. Шаблон

Поняття шаблону відносять до галузі узагальненого програмування. Існує можливість при описі класу не вказувати явно типи, які приймаються або повертаються операціями значень. Натомість, можна використовувати так звані «структурні нулі» або «заповнювачі» (*placeholders*), які можуть бути замінені фактичним значенням типу під час створення нового класу. Такий підхід реалізує принцип так званого «повторного використання коду», а клас, описаний таким чином, називатиметься «шаблонним класом» або просто — «шаблоном».

Шаблон на UML описується подібно до класу, за тим винятком, що в правому верхньому кутку, в пунктирному прямокутному контейнері, через кому вказуються параметри шаблону в наступному форматі:

```
назва_параметра: тип_параметра =  
                значення_за_замовчуванням
```



Далі наведено приклад опису шаблонного класу і явного поєднання його з екземплярами, які його реалізують (явне зв'язування забезпечується шляхом позначення відношення залежності типу «зв'язок» між класом і створеними від нього об'єктами. Зв'язки будуть розглянуті нижче у відповідному параграфі):

Для цього, як бачимо з ілюстрації, необхідно зв'язати об'єкти із шаблоном за допомогою пунктирної стрілки і вказати з використанням стереотипу «**bind**» (нагадаємо, що стереотипи розширюють словник UML, дозволяючи створювати нові блоки з існуючих) явні значення параметрів шаблону в наступному форматі:

```
<назва_1_параметра -> значення, ..., назва_n_параметра ->
                                значення >
```

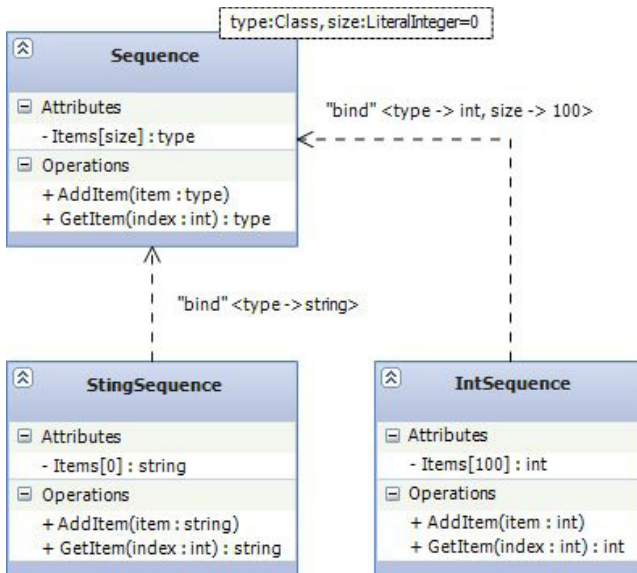


Рисунок 5.

Найчастіше використовують неявне зв'язування об'єкта, з реалізованим ним шаблоном. Неявне зв'язування об'єкта з шаблоном схоже на опис звичайного об'єкта, але з переліком параметрів шаблону в наступному вигляді:

```
назва_об'єкта: назва_шаблону <назва_параметра_шаблону =  
значення>
```

Значення параметрів шаблону розділяються комами.

Нижче наведено попередній приклад з тим винятком, що об'єкти пов'язані неявно:

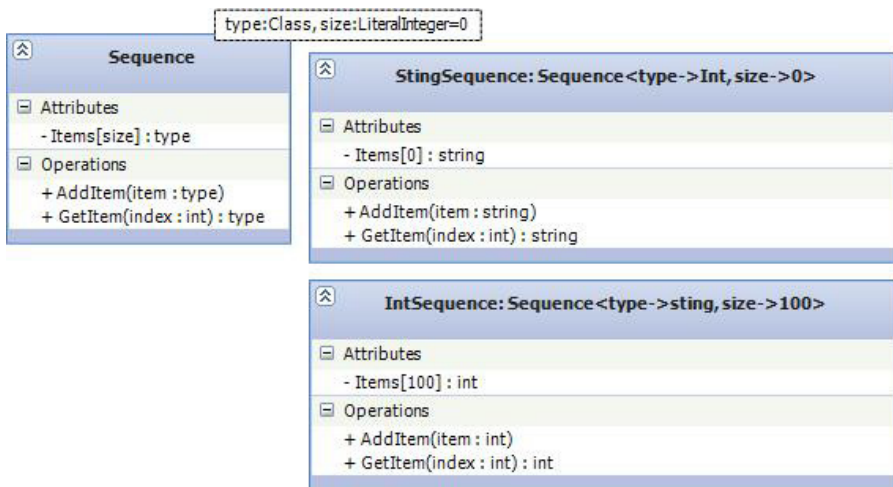


Рисунок 6.

### 1.3. Зв'язки між класами

Розглядаючи питання моделювання, необхідно розглянути поняття системи. Оскільки сам термін моделювання передбачає, що проектувальник займається створенням або відтворенням чогось цілого з деяких

частин. Зазвичай, це ціле називають системою, обумовлюючи необхідну умову наявності властивості емерджентності системи. Така властивість, власне кажучи, і є системовиділяючою.

Можна спрощено уявити систему як сукупність елементів і характери зв'язків між ними. Під емерджентністю, в свою чергу, розуміють наявність властивостей у системи, які не зводяться до сукупності властивостей складових її елементів.

Система має дві важливі характеристики: структуру і характер зв'язків між елементами. Наявність декількох елементів є аксіомою, яка, вочевидь, впливає з поняття моделювання.

Зазвичай, під структурою розуміють сукупність складових елементів системи, незалежно від цього, як вони між собою пов'язані або, інакше кажучи, у яких зв'язках вони між собою перебувають. Тоді як характер зв'язків між елементами системи значущий так само, як і структура. Оскільки саме він визначає характер процесів, що протікають у системі, а отже — впливає на властивості системи, як цілого.

Далі будуть розглянуті різні види зв'язків між класифікаторами, передбачені синтаксисом мови моделювання UML.

### 1.3.1. Зв'язок залежності

В UML, зв'язок залежності — це зв'язок, за яким один елемент, зазвичай — «клієнт» (*client*), використовує інший елемент, — «постачальник» (*supplier*), або залежить від нього.

Зв'язок залежності може використовуватися в діаграмах: класів, компонентів, розгортання і варіантів використання, щоб вказати, що зміни в «постачальнику» можуть вимагати відповідних змін у «клієнті».



Рисунок 7.

Також, можливе використання зв'язків залежності, щоб вказати на пріоритет елементів.

Зазвичай, зв'язки залежності не мають назв або підписів. Однак, їх можна використовувати для того, щоб визначити характер залежності між елементами моделі.

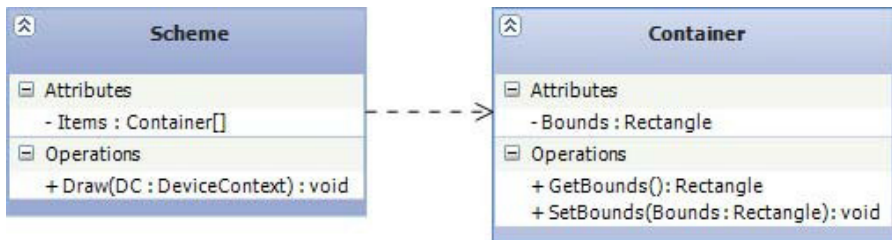


Рисунок 8.

Синтаксисом UML передбачені на наступні види (характери) зв'язків залежності:

- **абстракція** (ключові слова, які можуть бути використані для вираження цього типу залежності: **abstraction**, **derive**, **refine**, **trace**) — визначає зв'язок між двома або декількома елементами моделі, які визначають (виражають, описують) одне й те саме поняття на різних рівнях абстракції;

- **зв'язок** (ключове слово — **bind**) — пов'язує аргументи шаблону з параметрами шаблону в об'єктах;
- **реалізація** (ключове слово — **realize**) — вказує, що елемент клієнт — це реалізація елемент-постачальника;
- **заміщення/заміна** (ключове слово — **substitute**) — вказує, що елемент-клієнт займає місце елемента-постачальника. При використанні такого характеру залежності, інтерфейс клієнта обов'язково має відповідати інтерфейсу постачальника;
- **використання** (ключові слова: **use**, **call**, **create**, **instantiate**, **send**) — вказує, що один елемент вимагає інший елемент для реалізації тієї чи іншої функції.

Приклад залежності типу «зв'язок» був наданий в параграфі вище, присвяченому шаблонам.

### 1.3.2. Зв'язок асоціації

Зв'язок асоціації — це зв'язок між двома класифікаторами, який описує причину зв'язку і правило, яке зв'язок визначає. З одного боку, асоціація визначає структурне відношення, яке об'єднує два класифікатори, але з іншого — як атрибут, визначає властивості класифікатора. Наприклад, асоціації можуть бути використані для того, щоб представити проектні рішення, визначені для класифікаторів. Це може бути ілюстрація того, як об'єкт одного класу реалізує доступ до об'єкта іншого класу, а може бути і ілюстрація того, що між двома класифікаторами на концептуальному рівні є логічний зв'язок.

Нижче наведено приклад використання асоціації (рис. 9).

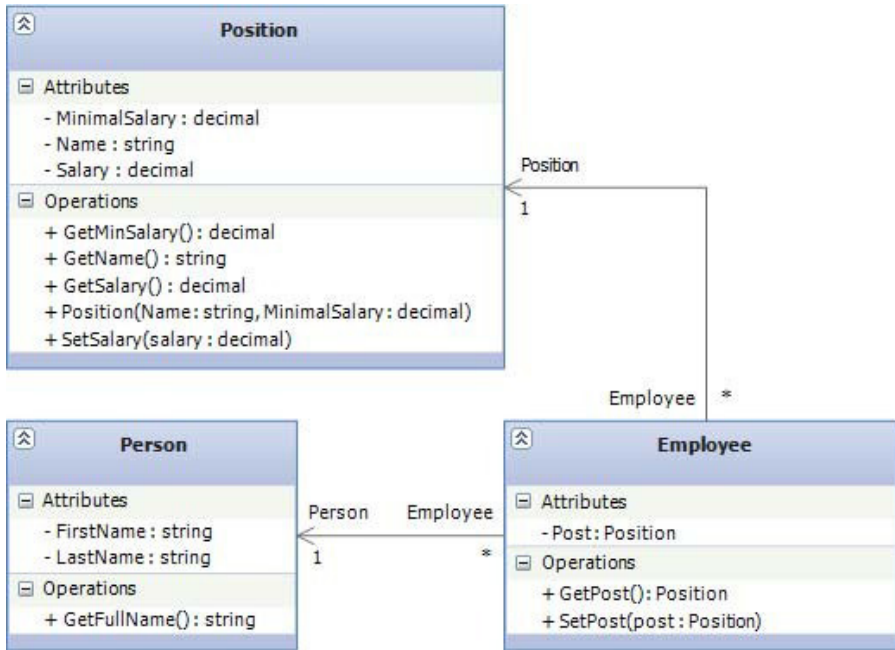


Рисунок 9.

Наведений приклад демонструє, що будь-який об'єкт, який описує працівника, може бути пов'язаний з одним об'єктом, що описує людину, яка є працівником, та з одним об'єктом, що описує посаду. Це можна трактувати так, що кожна людина може бути пов'язана лише з однією посадою за допомогою об'єкта допоміжного класу **Employee**.

Кінець асоціації може бути позначений міткою, яку називають «назвою ролі», і «кратністю», яка показує, скільки об'єктів можуть брати участь в асоціації. У наведеному нижче прикладі символ **\*** біля кожного класифікатора говорить про те, що, з одного боку, будь-яка лінія може містити будь-яку кількість точок,

а з іншого — будь-яка точка може належати будь-якій кількості ліній одночасно.

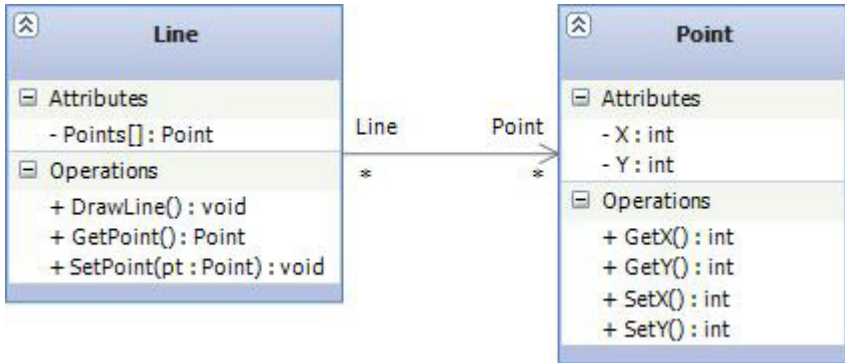


Рисунок 10.

### 1.3.3. Зв'язок узагальнення

Узагальнення — це зв'язок, у якому один елемент (дочірній) базується на іншому елементі (батьківському). Зв'язок узагальнення використовується в діаграмах: класів, компонентів, розгортання і варіантів використання, щоб проілюструвати, що дочірній елемент отримує всі атрибути, операції та зв'язки, визначені в базовому елементі.

Варто пам'ятати, що зв'язок узагальнення може використовуватися лише до однотипних класифікаторів. Наприклад, він може використовуватися для двох класів або для двох варіантів використання, але не може бути між класом та варіантом використання (про варіанти використання буде докладно описано в уроці, присвяченому цьому питанню). У зв'язку узагальнення один батьківський елемент може мати один і більше дочірніх

елементів. Один дочірній елемент може мати декілька батьківських елементів, але переважним вважається використання не більше одного батьківського елемента.

Зв'язок узагальнення має вид суцільної стрілки, спрямованої від дочірнього елемента до батьківського. Згідно з синтаксисом UML, зв'язки узагальнення не мають назв.

На рівні реалізації ставлення узагальнення можна умовно співвідносити зі спадкуванням.

У наведеному нижче прикладі ілюструється використання відношення узагальнення в діаграмах класів.

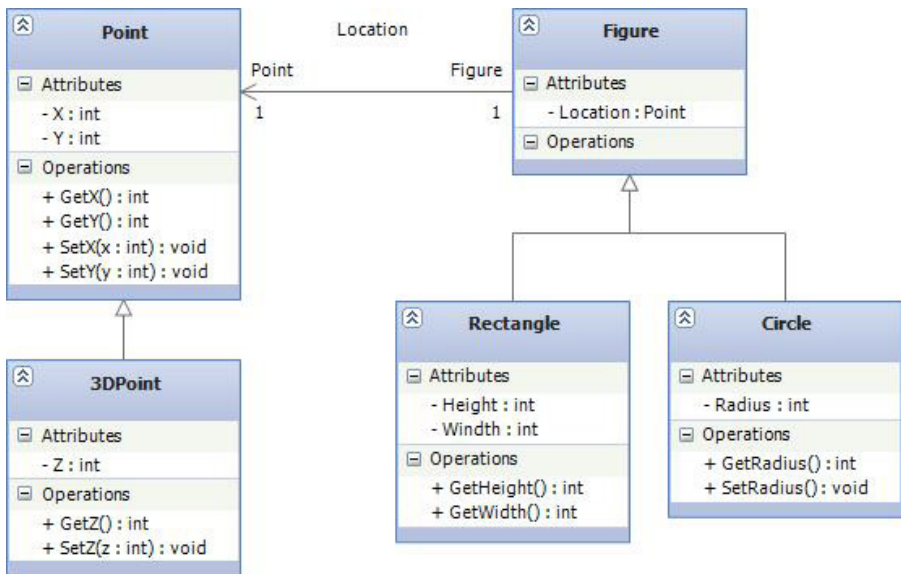


Рисунок 11.

#### 1.3.4. Зв'язок реалізації

Зв'язок реалізації — це зв'язок між двома елементами моделі, в якому один елемент моделі (елемент-клієнт)



реалізує поведінку, яка визначається іншим елементом моделі (елемент-постачальник). Декілька «клієнтів» можуть реалізовувати поведінку одного «постачальника».

Зв'язок реалізації може бути використане у діаграмах класів і компонентів. Графічно, зв'язок реалізації зображується пунктирною стрілкою, спрямованою від клієнта до постачальника.

Подібно до зв'язку узагальнення, зв'язок реалізації не має назви. Але, якщо зв'язок має назву, то описання має бути поряд зі з'єднувальною лінією, яка візуалізує зв'язок.

У наведеному нижче прикладі ставлення реалізації демонструє, що інтерфейс **IMovable** реалізується класом **Element**, який описує графічний елемент на діаграмі. На рівні реалізації, такий зв'язок буде наслідуванням.



Рисунок 12.

## 1.4. Порівняльний аналіз зв'язків

Порівнюючи зв'язки, можна дійти тільки одного загального висновку, що усі зв'язки без винятку необхідні і корисні при моделюванні. Нижче наведено порівняльну таблицю різних характеристик відносин.

Таблиця 1

Характеристика	Зв'язок залежності (dependency relationship)	Зв'язок асоціації (association relationship)	Зв'язок узагальнення (generalization relationship)	Зв'язок реалізації (realization relationship)
Графічне відображення				
Підпис	Не потрібен	З обох сторін класифікаторів мають бути мітки і підпис зв'язку	Не потрібен	Не потрібен
Діаграми, в яких використовується	Діаграми класів, діаграми компонентів, діаграми розгортання і діаграми варіантів використання	У всіх видах діаграм	Діаграми класів, діаграми компонентів, діаграми розгортання і діаграми варіантів використання	Діаграми класів, діаграми компонентів

## 1.5. Практичні приклади побудови діаграм класів

Далі ми пропонуємо розглянути приклад оформлення діаграми класів, яка відображає концептуальну модель RSS-клієнта. Заздалегідь хочемо зазначити, що у запропонованій моделі не враховується можливість виведення новин через графічний інтерфейс користувача або навіть у текстовому режимі, щоб не ускладнювати діаграму.

Концепція досить проста: згідно з цією моделлю, пропонується описати тип даних для представлення кожної окремої новини, організувати новини в колекцію, яка інкапсулюватиметься відповідним класом, який також за допомогою спеціального адаптера отримуватиме новини з RSS-каналу і організовувати їх у колекцію об'єктів відповідного класу, зазначеного вище.

Запропонована модель використовує більшість описаних вище зв'язків.

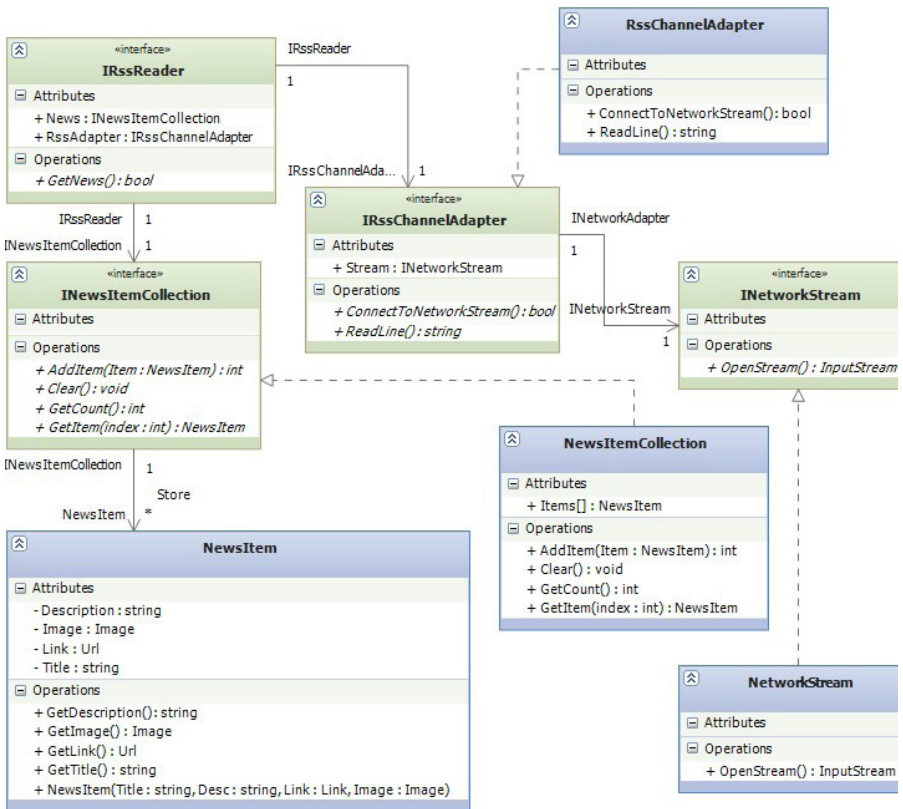


Рисунок 13.

Також зазначимо, що дана модель не є досконалою і може містити низку дефектів, внесених автором навмисне. Завдання читача полягає в тому, щоб спроектувати свій варіант запропонованої моделі (покращений і доповнений).

## 2. Діаграма станів

### 2.1. Цілі даного типу діаграм

Діаграма станів — це діаграма, в якій відображено безліч станів, у яких може міститися об'єкт під час його життєвого циклу, а також усі можливі переходи між цими станами, передбачені концепцією, розробленою проектувальником.

В більшості випадків, діаграма станів будується для одного класу, щоб показати динаміку поведінки об'єктів цього класу.

Діаграми станів зазвичай використовують для опису поведінки певного об'єкта в різних варіантах використання. Тоді як для опису поведінки кількох об'єктів доцільніше використовувати діаграми взаємодії.

Використання діаграм станів дозволяє всім членам групи розробників однаково представляти концепцію динамічної складової системи, запропоновану проектувальником. Також, даний тип діаграм дає можливість представити зовнішні (поведінкові) характеристики реалізованого класу, що дозволяє спроектувати його внутрішню структуру так, аби вона максимально відповідала вимогам експлуатації об'єктів даного класу.

### 2.2. Базові поняття

#### 2.2.1. Автомати

Кожна діаграма станів є автоматом. Під автоматом у UML розуміється формалізм, який використовується для

моделювання поведінки елементів моделі і самої системи. З іншого боку, автомат — це послідовність станів, які охоплюють усі етапи життєвого циклу об'єкта. Основними формалізмами, які використовуються при описі автомата, є стан і перехід.

Прийнято вважати, що тривалість знаходження об'єкта в одному з можливих станів набагато перевищує час, необхідний для здійснення переходу з одного стану в інший.

Ідеальним було б рішення, зовсім не витратити час на перехід між станами, або, — якщо б час дорівнював нулю.

### 2.2.2. Стан

#### *Поняття стану*

У мові моделювання UML, під станом розуміється метаклас, який використовується для моделювання окремо взятої ситуації, яка передбачає виконання певної умови. Весь спектр станів певного об'єкта або класу може бути у вигляді набору значень відповідного атрибуту класу. Очевидно, що зміна значення атрибута виражає зміну стану об'єкта.

Але не будь-який атрибут може бути використаний як індикатор стану. Зазвичай, коли говорять про стани, то мають на увазі такі атрибути системи, які відображають динамічний аспект поведінки.

Важливо також розуміти, що в даному випадку стани — це лише метафора, яка відображає внутрішні зміни, що відбуваються в системі.

Графічно, стан зображують овалом із округлими краями, у центрі якого вказується назва стану. Приклад зображення станів на рисунку 14.

Очікування

Здійснення діалогу з користувачем

Рисунок 14.

*Назва стану*

Текст-назва стану має розкривати зміст і суть дій, які виконуються системою в цьому стані (саме в цей момент її функціонування).

Для назв станів слід вибирати дієслова теперішнього часу або відповідні дієприкметники. Назву завжди слід починати з великої літери.

*Список внутрішніх дій*

Список внутрішніх дій містить перелік дій, які система виконує у відповідному стані. Дії слід зазначати у порядку їх виконання системою. Кожна дія записується з нового рядка у наступному форматі:

```
назва_дії/зміст_дії
```

На рисунку 15 показано, як виглядатиме у графічному уявленні подія із зазначеним списком внутрішніх дій для стану передачі даних.

Передача даних

```
-----
connect / Ініціалізація з'єднання
send / Відправлення даних
disconnect / Закриття з'єднання
```

Рисунок 15.

### Початковий стан

Початковий стан — це окремий випадок стану, в якому об'єкт перебуває у початковому моменті часу. У початкового стану відсутні внутрішні дії, тому що воно передує першим діям виконуваним об'єктам, а значить і першому внутрішньому стану об'єкта. Початковий стан необхідний лише для того, щоб позначити момент початку існування об'єкта. Виходячи з вищезазначеного, початковий і кінцевий стан називають псевдостанами.

Початковий стан зображується у вигляді зафарбованого кола. Приклад зображення початкового стану наведено на рисунку 16.

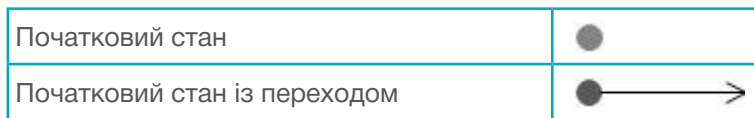


Рисунок 16.

### Кінцевий стан

Кінцеве значення, як і початкове, є псевдо-станом і відбиває момент завершення життєвого циклу автомата. Графічно, кінцевий стан зображують у вигляді зафарбованого кола, поміщеного в коло, як це показано на рисунку праворуч.

#### 2.2.3. Перехід

##### Поняття переходу

Перехід (*transition*) — це зв'язок між двома послідовними станами, який вказує на заміну одного стану іншим. Той стан, з якого виконується перехід, називають



початковим станом, а той, в який виконується перехід, — цільовим станом.

Як правило, перебуваючи у деякому стані, система виконує відповідні для цього стану дії. Тому, перехід буде можливий лише після виконання цих дій і певних супутніх умов, які відповідають переходу в цільовий стан. Виконання переходу між станами прийнято називати спрацюванням переходу.

Графічно перехід зображується суцільною стрілкою від початкового стану до цільового, як це показано на рисунку 17.

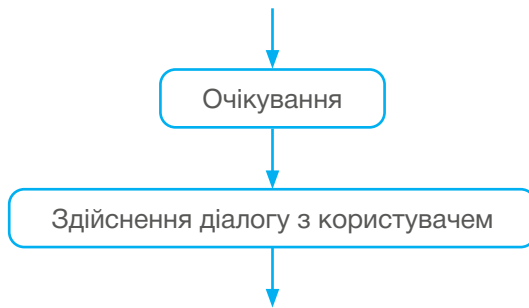


Рисунок 17.

Якщо біля стрілки не має жодного підпису, такий перехід вважається нетригерним (тригерні переходи — це переходи, які відбуваються як реакція на певну подію. Про події йтиметься далі). Якщо перехід вказано як нетригерний, то з контексту діаграми має бути однозначно зрозуміло, після якої дії він здійснюється.

Кожен перехід може бути, за потреби, позначений рядком тексту в наступному форматі:

сигнатура\_події[сторожова умова]/вираз дії

В свою чергу, сигнатура події визначає саму подію, з необхідними аргументами (аргументи розділяються комами) наступним чином:

```
назва_події (список_параметрів)
```

## Подія

Подія (*event*) — декларація певного факту, який може мати місце у просторі та часі. Події суворо впорядковані у часі, а отже, після того, як деяка подія сталася, то повернутися до попередньої вже неможливо (за винятком тих випадків, коли в моделі явно вказана циклічна послідовність подій).

Події, як правило, вказують на деякі зовнішні зміни при переході системи з одного стану в інший. Наприклад, подія увімкнення говорить про те, що система перейшла з пасивного стану в активний. Або перемикання представлення документа в html-редакторі свідчить про те, що графічна підсистема перейшла з текстового уявлення в графічне.

якщо: поява картки/ отримати пін-код



Рисунок 18.

Графічно, подія вказується як текст, підписаний біля переходу. Наведений рисунок 18 ілюструє використання події.

### Сторожова умова

Сторожова умова (*guardcondition*) — це умова, яка визначає, у якій ситуації, або в яких визначеннях атрибутів буде здійснений перехід між станами.

Сторожова умова записується у прямокутних дужках після події-тригера. Необхідно пам'ятати, що семантика умови, зазначеної як сторожової, має бути зрозумілою з контексту діаграми. На рисунку 19 показано, як виглядає використання сторожової умови графічній формі.

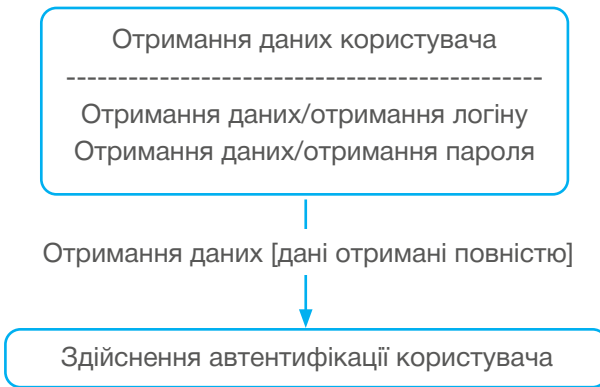


Рисунок 19.

### Вираз дії

Вираз дії (*action expression*) — це опис дії, виконуваної відразу після того, і лише за умови того, коли спрацьовує перехід. Вираз дії має бути виконано до початку виконання будь-якої дії у цільовому стані. Вираз дії є атомарною операцією, тобто вона не може бути перервана до свого завершення, що виключає можливість обривання, відкату переходу або запобігання переходу в цільовий стан. Вираз дії вказується після символу «/» у назві переходу,

до якого він додається. Якщо використовується декілька виразів дії, їх вказують через «,».

Нижче наведено приклад (рис. 20) використання висловлювання дій і як це виглядає графічно.

Відсутність відповіді [перевищено ліміт очікування]/  
розірвати з'єднання, звільнити ресурси



Рисунок 20.

### 2.3. Складений стан і підстан

Складений/Композитний стан (*compositestate*) — це стан, що складається з інших, вкладених у нього станів. По відношенню до вкладених станів, складений стан виступає в ролі суперстану або надстану (*superstate*), а вони в свою чергу, до нього — як підстани (*substates*).

Графічно, складений стан зображується як звичайний стан, але з вкладеними всередину нього фігурами підстанів і переходів між ними (рис. 21).

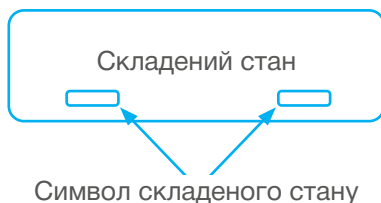


Рисунок 21.

Також, можливе використання станів із прихованою внутрішньою структурою. У такому разі, мається на увазі,

що існує окрема діаграма станів, яка описує структуру такого складового стану. Саме цей стан описується у звичний спосіб із зазначенням символу складеного стану і усіх переходів, які виконуються в ньому.

Складений стан може містити два або більше паралельних підавтоматів або декілька послідовних підстанів.

### 2.3.1. Послідовні підстани

Послідовні підстани (*sequential substates*) використовуються для моделювання такого об'єкта, який на всьому протязі цього стану може одночасно перебувати в одному, і тільки в одному підстані. Хоча поведінка об'єкта такого типу і може бути виражена через послідовну зміну звичайних станів, використання складеного стану дає можливість більш тонко і детально представити специфіку його поведінки.

Складовий стан може містити початковий та кінцевий стани як підстани.

На представленому рисунку 22 показано, як графічно виглядає складовий стан з послідовними підстанами.

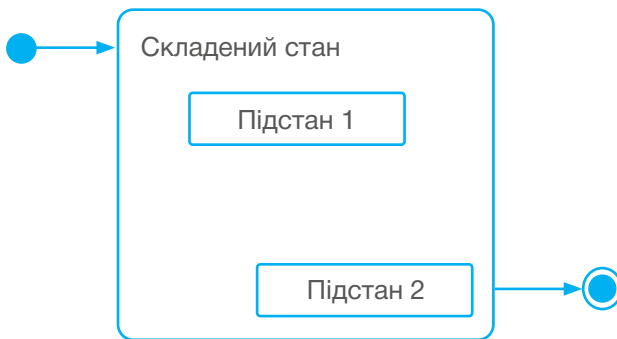


Рисунок 22.

### 2.3.2. Паралельні підстани

Паралельні підстани (*concurrentsubstates*) дозволяють уявити стан, в якому паралельно виконуються декілька підавтоматів, кожен з яких окремо є діаграмою станів, порівняно невеликого обсягу, поміщену усередину складеного стану.

На наведеному рисунку 23 показаний спосіб графічного уявлення паралельних підавтоматів складеного стану. З нових графічних елементів можна виділити лише пунктирні розділові лінії, які візуально відокремлюють паралельні підстани один від одного.

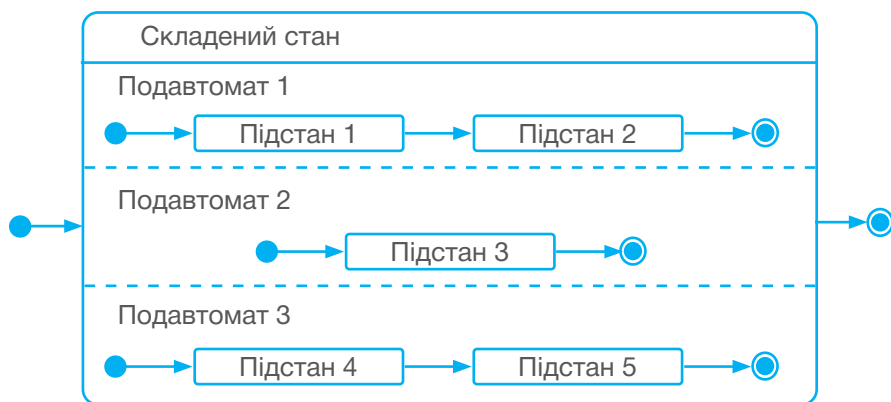


Рисунок 23.

### 2.4. Історичний стан

Існування історичного стану обумовлено необхідністю, у ряді випадків, виходити з поточного стану із збереженням для подальшої можливості повернутися до нього. При цьому, враховується та частина діяльності, яка була виконана до здійснення переходу.

Історичний стан (*history state*) застосовується у контексті складеного стану. Він використовується для запам'ятовування того підстану, який був поточним на момент виходячи зі стану.

Існує два види історичного стану: недавній стан (*shallow history state*) і давній стан, або стан глибокої історії (*deep history state*).

Графічно, стан недавньої історії зображується у вигляді кола з великою літерою «Н» у ньому, а стан глибокої історії — з літерою «Н» і символом «\*».

Нижче на рисунку 24 наведено графічні позначки для історичних станів.

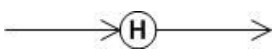

Стан недавньої історії	
Стан глибокої історії	

Рисунок 24.

Історичний стан недавньої історії має бути першим підстаном у складеному стані і при першому потраплянні до цього історичного підстану, де він виступає в якості початкового стану підавтомату. Якщо під час роботи підавтомату відбувається вихід, то історичний стан запам'ятовує, який із підстанів був активним у момент виходу, і при поверненні виклику в даний підавтомат, перенаправляє його до необхідного підстану.

Оскільки запам'ятований підстан теж може бути складеним, існує історичний стан глибокої історії, який використовується для запам'ятовування усіх підстанів поточного підавтомату будь-якого рівня вкладення.

## 2.5. Складні переходи

### 2.5.1. Переходи між паралельними станами

Паралельний перехід — це перехід, який може мати одночасно декілька станів-джерел і декілька цільових станів. Графічно, такий перехід позначається зафарбованим вертикальним прямокутником, як показано на рисунку 25.

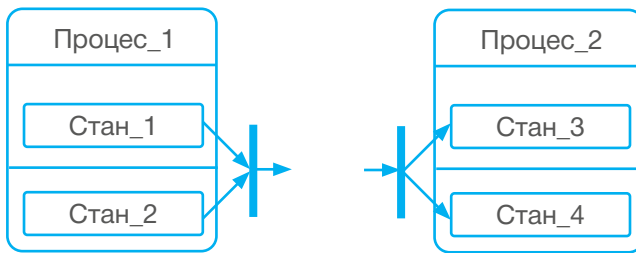


Рисунок 25.

Якщо перехід має багато вхідних переходів, його називають з'єднанням або об'єднанням (*join*), а у випадку, коли багато вихідних та один вхідний — розгалуженням (*fork*).

### 2.5.2. Переходи між складеними станами

Перехід до складеного стану позначається стрілкою, спрямованою до межі цього складеного стану. Це графічний елемент означає перехід до початкового стану усіх підавтоматів суперстану.

Перехід, що виходить із межі складеного стану, передбачає вихід деякого об'єкта з підстану суперстану. Тоді як перехід, спрямований із вкладеного підстану за межі суперстану, означає вихід із складеного стану.



На представленому рисунку 26, зображені усі зазначені переходи між складеними станами.

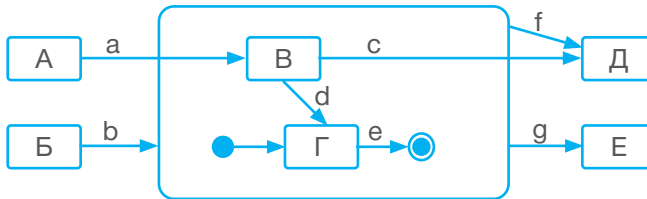


Рисунок 26.

### 2.5.3. Синхронізуючі стани

Синхронізуючий стан (*sync state*) використовується для синхронізації настання окремих подій у паралельних підавтоматах з метою їхньої синхронізації. Синхронізуюча подія позначається невеликим колом з символом «\*» у ньому. Такий стан використовується спільно з переходом-розгалуженням і переходом об'єднання, аби явно вказати події в паралельних підавтоматах, які безпосередньо впливають на поведінку того підавтомата, для якого вони призначені.

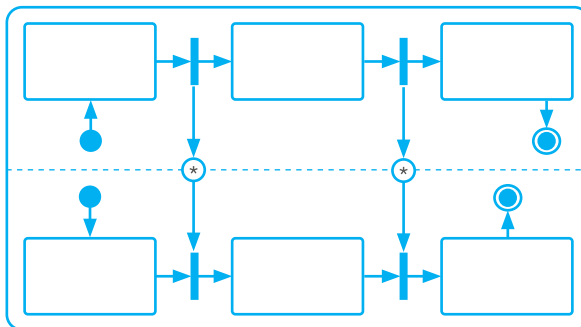


Рисунок 27.

## 2.6. Практичні приклади побудови діаграм станів

В якості прикладу ми розглянемо діаграму станів для візуалізації поведінки пристрою банкомата.

В рамках запропонованої моделі передбачається, що банкомат перебуває в режимі очікування доти, доки користувач не помістить картку до приймача. Після цього банкомат здійснює отримання пінкоду.

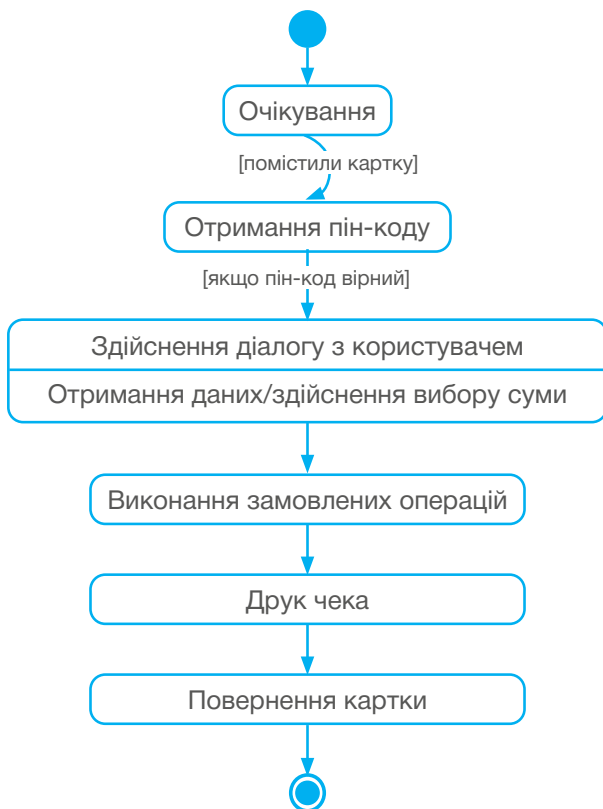


Рисунок 28.

Якщо пін-код вірний, то банкомат здійснює діалог з користувачем для отримання необхідної йому суми, і після

цього здійснює виконання відкладених операцій, — друк чека і повернення картки.

У поданій моделі навмисне не передбачена ситуація, при якій користувач забув пін-код і хоче отримати картку назад, або припинити взаємодію з банкоматом під час виконання діалогу (рис. 28).

Пропонуємо вам для домашнього завдання самостійно доопрацювати запропоновану модель.

## 3. Діаграма активності

У процесі моделювання системи важливо не тільки вказати її структуру, характер зв'язків між складовими елементами системи, а й стани, в яких може знаходитися система, і переходів між цими станами. Але буває абсолютно необхідним специфікувати алгоритмічні особливості функціональних модулів системи і визначити характер процесів в ній. Зазвичай, для опису алгоритму використовують такі алгоритмічні мови, як умовна алгоритмічна мова (УАМ), мова блок-схем або структурні схеми алгоритмів. У мові UML для цих цілей використовуються діаграми діяльності (*activity diagrams*), призначені для уявлення алгоритмів у графічному вигляді.

Візуально, діаграма діяльності представлена у вигляді графа, вершинами якого є стани активності, а дугами — переходи від одного стану дії до іншого.

Під діями (*actions*) розуміються елементарні обчислення, які призводять до певного конкретного результату.

### 3.1. Цілі даного типу діаграм

Основною метою використання діаграми діяльності є візуалізація особливостей реалізації операцій класів, а також опис реакцій на внутрішні події системи.

### 3.2. Базові поняття

#### 3.2.1. Стан дії

Стан дії (*actionstate*) — це стан із вхідною дією і, з щонайменше, одним вихідним переходом. З огляду на свою

елементарність, стан дії не має внутрішніх переходів. А умовою переходу є завершення дії.

Як правило, дія відповідає одному кроку модельованого алгоритму.

Графічно, стан дії зображують прямокутником із заокругленими кінцями, із зазначенням всередині виразом-дією (*action-expression*). Вираз-дія має бути унікальним у межах однієї діаграми діяльності. Дія може бути описана звичною нам мовою (наприклад: «порівняти ідентифікатори»), а може — деяким псевдокодом або мовою програмування.

Якщо для дії використовується звична мова, то для визначення (назви) дії прийнято використовувати дієслово, яке, як правило, виражає твердження в спонукальному відмінку і з додатковими поясненнями (наприклад: «виконати перевірку диска») (рис. 29).

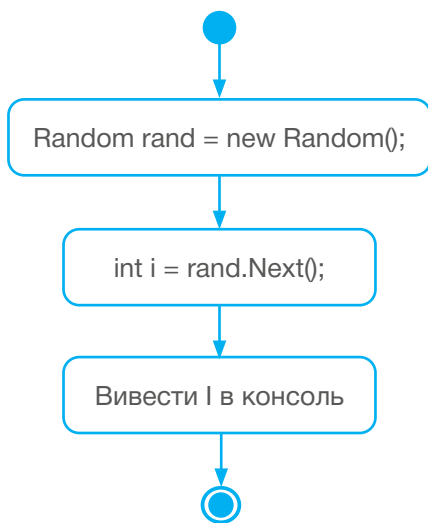


Рисунок 29.

Як і в діаграмі станів, в діаграмі діяльності є два види підстанів: початковий і кінцевий стани. Діаграма діяльності повинна мати лише один початковий стан, з якого діаграма починається, і один кінцевий стан, яким вона закінчується.

Розташовувати дії в діаграмах діяльності потрібно зверху вниз.

### 3.2.2. Переходи

Перехід, як формалізм, ми розглядали в попередньому розділі, присвяченому діаграмам станів. У цьому розділі ми розглянемо тільки специфіку використання переходів в діаграмах діяльності. По перше, слід відзначити, що в діаграмах діяльності використовуються тільки нетригерні переходи. Тобто, для переходу не потрібно чекати настання деякої події, перехід спрацьовує відразу після завершення дії.

Якщо зі стану дії виходить лише один перехід, то він, як правило, не має підпису. В тому випадку, коли переходів більше одного, то кожен із них слід помітити умовою, яка називається «сторожовою» і вказується у прямокутних дужках біля лінії переходу. Очевидно, що всі сторожові умови мають взаємно виключати одна одну.

Сам перехід, як і діаграмі станів, зображується у вигляді суцільної стрілки, спрямованої від вихідного стану до цільового.

У тих випадках, коли потік виконання, залежно від деякої умови, має розділитися на два альтернативні напрямки, прийнято говорити про розгалуження.

Розгалуження графічно помічається за допомогою ромба з одним вхідним і, щонайменше, двома вихідними переходами. Вхідний перехід прийнято з'єднувати з

верхньою чи лівою вершиною ромба. Для вихідних переходів розгалуження, як зазначалося вище, обов'язково мають бути встановлені сторожові умови.

На рисунку 30 наведено приклад використання переходів і розгалуження в діаграмах діяльності.

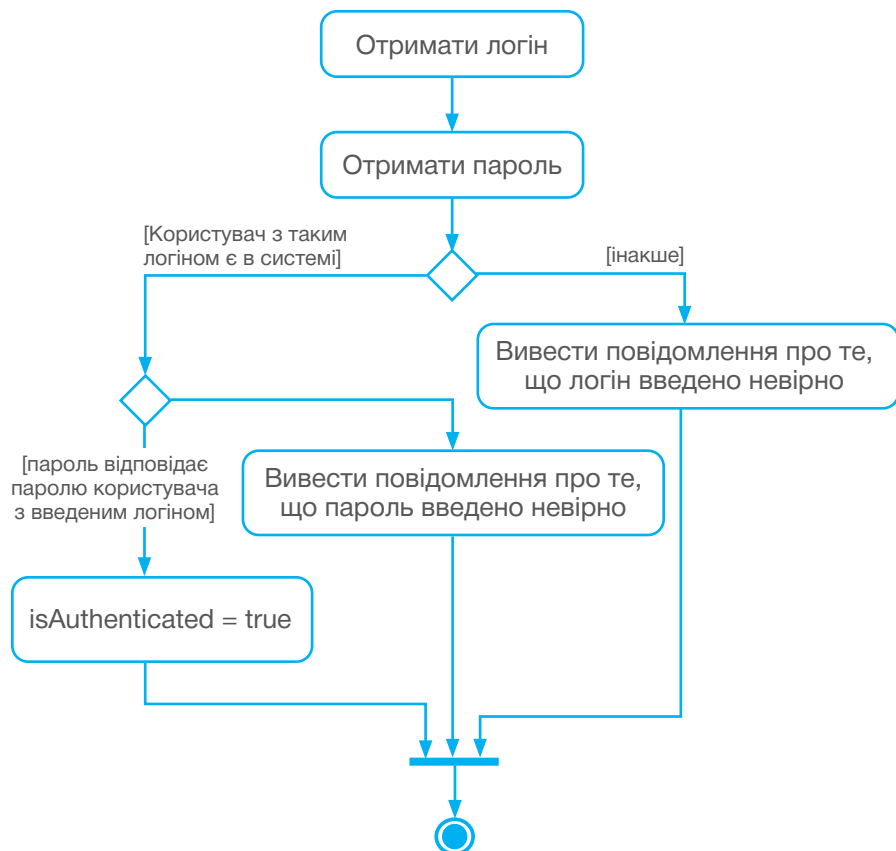


Рисунок 30.

Окремо хочеться відзначити один графічний елемент, про який ще не зазначалося — це об'єднання переходів. Об'єднання переходів використовується у випадках, коли

різні гілки коду «сходяться». Як можна побачити з прикладу, об'єднання зображують у вигляді зафарбованого прямокутника. Об'єднання переходів може мати безліч вхідних переходів і один вихідний.

### 3.2.3. Доріжки

Діаграми діяльності можуть використовуватися не тільки для візуалізації алгоритмів і специфікації прикладних протоколів модельованої системи. На сучасному етапі розвитку програмного забезпечення і мереж комунікації, на передній план виходить завдання моделювання, так званих, бізнес-процесів, тобто порядку виконання процедур, пов'язаних із забезпеченням функціонування комерційних структур та організацій.

Необхідно відзначити, що абсолютно необов'язково, аби ці моделі знаходили своє вираження в діючому програмному забезпеченні. Потрібним є також і моделювання людських операцій, з одного боку — для їх аналізу, а з іншого — для досягнення найбільшої ефективності функціонування окремих операторів і колективів.

Але бізнес має власну специфіку. Як правило, комерційні організації мають складну розгалужену структуру, яка поділяє усі функції підприємства між відділами і підрозділами, а бізнес-процес представлені у вигляді переходів дій з одного підрозділу до іншого.

Для візуалізації цих переходів, в UML створені спеціальні графічні конструкції, які називають доріжками (*swimlanes* — з англ. «плавальні доріжки в басейні»). На діаграмі доріжки відокремлені вертикальними лініями (вони наче розділяють діаграму на відсіки), а у верхній частині



доріжки вказується текстовий підпис, який відповідає назві відділу, виконуючого дії, вказані на відповідній доріжці.

Нижче (рис. 31) ілюструється використання доріжок на прикладі моделювання прийому і виконання замовлень у центрі сервісного обслуговування.

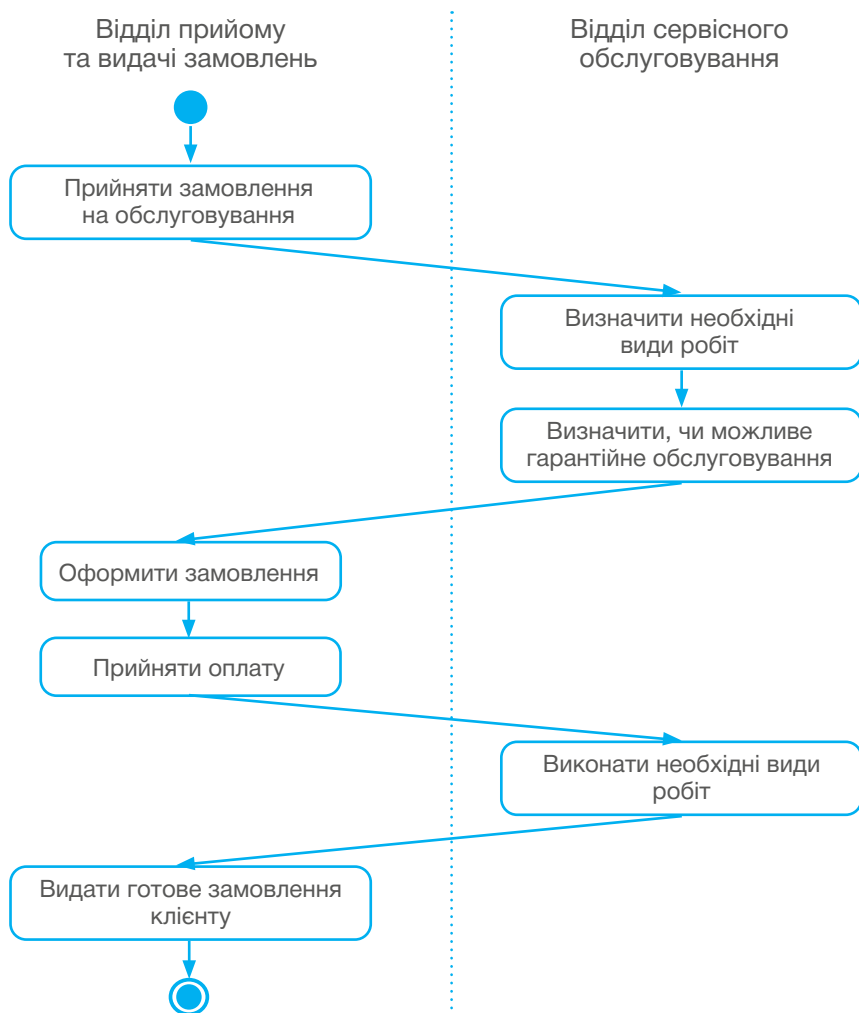


Рисунок 31.

### 3.2.4. Об'єкти

В даному випадку, говорячи про об'єкти, їх необхідно розуміти в контексті бізнес-процесів, оскільки бізнес-процеси зазвичай мають справу з деякими об'єктами. Наприклад, у випадку з описом роботи сервіс центру — це замовлення, яке прийняте у відділі прийому та видачі замовлень і надходить на обробку до відділу сервісного обслуговування.

За час проходження усіх етапів процесу, замовлення переходить з одного стану в інший, який, у нашому випадку, визначає готовність замовлення. Тому, зазвичай говорять не про об'єкт, а лише про його стан. Стан об'єкта графічно відображається практично так само, як і в діаграмах класів. Цей стан представлено у вигляді прямокутника, у верхній частині якого вказується його назва (зазвичай, назва об'єкта підкреслюється). Після назви, через двокрапку, може зазначатися тип об'єкта, якщо він визначений у моделі. Відмінність полягає тільки в тому, що після назви об'єкта прямокутних дужках вказується стан об'єкта при переході між діями. У цьому контексті, дії розглядаються як набір операцій із зміни станів цільового об'єкта.

Перехід від дії до стану об'єкта зображується суцільною стрілкою, направленою від дії до стану об'єкта, а перехід від стану об'єкта до наступної дії зображується пунктирною стрілкою і називається потоком об'єктів.

На рисунку 32 наведено попередній приклад, але вже з використанням станів об'єкта «Замовлення».

Необхідно відзначити ще один важливий момент: об'єкт може бути розташований на межі між доріжками.

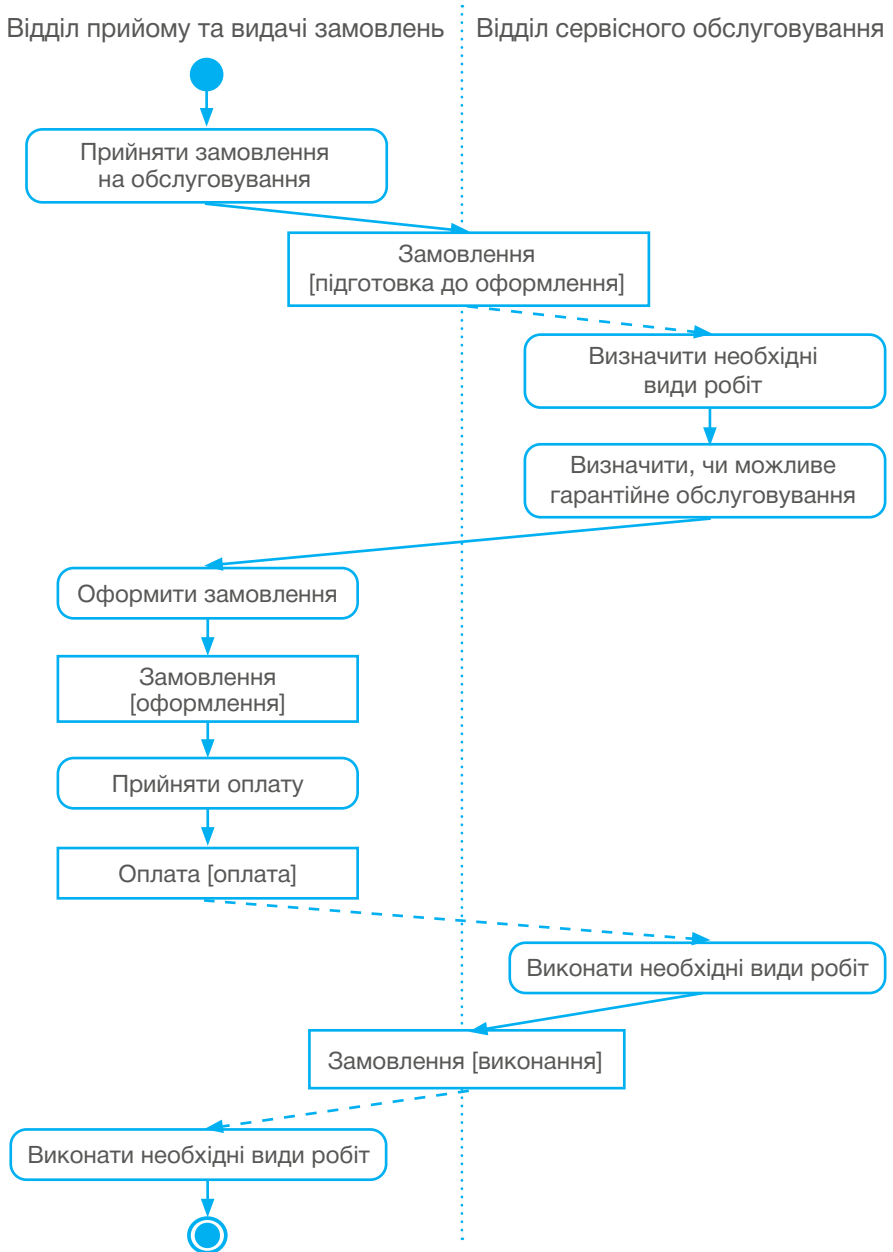


Рисунок 32.

Таке розташування стану об'єкта слід розуміти як необхідність підготовки деяких документів, що супроводжують бізнес-процес, без яких завершення переходу об'єкта в інший стан неможливе.

### **3.3. Практичні приклади побудови діаграм діяльності**

Для прикладу діаграми діяльності ми хочемо навести таку діаграму, яка ілюструє алгоритм визначення факторіалу натурального числа (рис. 33). У цій моделі представлена ітеративна форма усім відомого алгоритму.

Слід звернути увагу на те, що при використанні розгалуження, у випадку, коли визначаються лише дві альтернативні гілки, умову можна вказувати тільки для однієї гілки, а в другій гілці використовувати твердження «інакше», яке і є умовою альтернативної гілки.

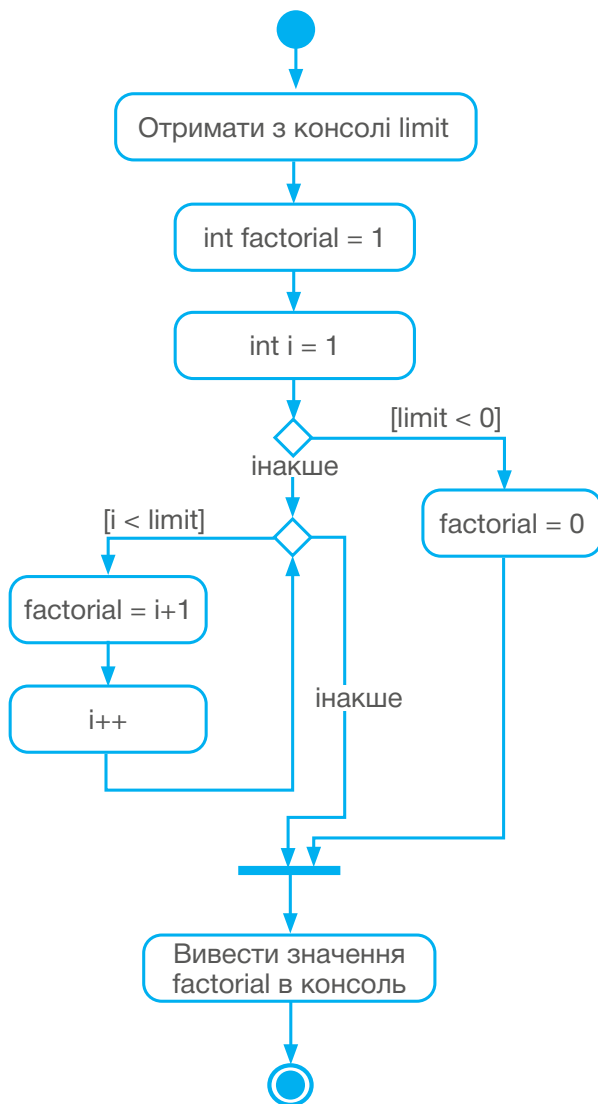


Рисунок 33.



## Урок №3

# Діаграма класів, діаграма станів, діаграма активності

© STEP IT Academy, [www.itstep.org](http://www.itstep.org)

Усі права на фото-, аудіо- і відеотвори, що охороняються авторським правом і фрагменти яких використані в матеріалі, належать їх законним власникам. Фрагменти творів використовуються в ілюстративних цілях в обсязі, виправданому поставленим завданням, у рамках учбового процесу і в учбових цілях, відповідно до законодавства про вільне використання твору без згоди його автора (або іншої особи, яка має авторське право на цей твір). Обсяг і спосіб цитованих творів відповідає прийнятим нормам, не завдає збитку нормальному використанню об'єктів авторського права і не обмежує законні інтереси автора і правовласників. Цитовані фрагменти творів на момент використання не можуть бути замінені альтернативними аналогами, що не охороняються авторським правом, і відповідають критеріям добросовісного використання і чесного використання.

Усі права захищені. Повне або часткове копіювання матеріалів заборонене. Узгодження використання творів або їх фрагментів здійснюється з авторами і правовласниками. Погоджене використання матеріалів можливе тільки якщо вказано джерело.

Відповідальність за несанкціоноване копіювання і комерційне використання матеріалів визначається чинним законодавством.