МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ФАХОВИЙ КОЛЕДЖ ПРОМИСЛОВОЇ АВТОМАТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

ЗАТВЕРДЖУЮ
Заступник директора з
навчально-методичної роботи
_____ Вікторія ОКСАНІЧЕНКО
___.___.2022 року

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

(назва навчальної дисципліни)

галузі знань _____12 «Інформаційні технології»_____
(шифр і назва галузі знань)

спеціальності_____121 «Інженерія програмного забезпечення»_____
(шифр і назва спеціальності)

освітня програма_____Інженерія програмного забезпечення_____
(назва освітньо-професійної програми)

Одеса 2022 рік

Конспект лекцій з дисципліни “Конструювання програмного забезпечення” для підготовки фахових молодших бакалаврів за спеціальністю 121 «Інженерія програмного забезпечення»
(назва освітньо-професійного ступеня)

Укладач: викладач вищої кваліфікаційної категорії ФКПАІТ ОНТУ Тетяна КОСТИРЕНКО

Конспект лекцій розглянуто і затверджено на засіданні циклової комісії ФКПАІТ ОНТУ “Комп’ютерних наук та інженерії програмного забезпечення”

Протокол від ___._____.20__ року №___

Голова циклової комісії “Комп’ютерних наук та інженерії програмного забезпечення”

_____Тетяна КОСТИРЕНКО
(підпис) (власне ім’я та прізвище)
___._____.20___ року

Зміст

Блок змістових модулів №1 Мови конструювання

Змістовий модуль 1.1 Організація процесу конструювання. Керівництво програмним проектом

Тема 1.1.1 Визначення технології конструювання програмного забезпечення. Стратегії конструювання

Лекція №1

(2 години)

Мета: Дати визначення технології конструювання програмного забезпечення (ТКПЗ), розглянути методи, засоби та процедури ТКПЗ, розглянути основні парадигми ТКПЗ. Ознайомити студентів з стратегіями конструювання та розглянути які моделі їх реалізують.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.
II. Повідомлення теми та мети заняття;
III. Виклад нового матеріалу:

План:

1. Технологія конструювання програмного забезпечення:
   a. Методи;
   b. Засоби;
   c. Процедури;
2. Парадигми ТКПЗ:
   a. Класичний життєвий цикл;
   b. Макетування;
3. Стратегії конструювання;
4. Інкрементна модель;
5. Спіральна модель;
6. Компонентно-орієнтована модель.

IV. Узагальнення та систематизація знань;
V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми 1.1.1.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Що таке технологія конструювання програмного забезпечення?
2. Що забезпечують методи ТКПЗ?
3. Для чого необхідні утиліти ТКПЗ?
4. Що визначають процедури технології конструювання ПЗ?
5. Що називають парадигмами ТКПЗ?
6. Назвіть основні парадигми технології конструювання.
7. В чому позитивні та негативні сторони макетування?
8. Перелічіть основні етапи класичного життєвого циклу.

Технологія конструювання програмного забезпечення (ТКПЗ) - система інженерних принципів для створення економічного ПЗ, яке надійно й ефективно працює в реальних комп'ютерах.

Розрізняють методи, засоби й процедури ТКПЗ.

Методи забезпечують розв'язок наступних завдань:

- планування й оцінка проекту;
- аналіз системних і програмних вимог;
- проектування алгоритмів, структур даних і програмних структур;
- кодування;
- тестування;
- супровід.

Засоби (утиліти) ТКПЗ забезпечують автоматизовану або автоматичну підтримку методів. З метою спільного застосування утиліти можуть поєднуватися в системи автоматизованого конструювання ПЗ. Такі системи прийнято називати Case- Системами. Абревіатура CASE розшифровується як Computer Aided Software Engineering (програмна інженерія з комп'ютерною підтримкою).

Процедури є "клеєм", який з'єднує методи й утиліти так, що вони забезпечують безперервний технологічний ланцюжок розробки. Процедури визначають:

- порядок застосування методів і утиліт;
- формування звітів, форм по відповідних до вимог;
- контроль, який допомагає забезпечувати якість і координувати зміни;
- формування "віх", по яких керівники оцінюють прогрес.

Процес конструювання програмного забезпечення складається з послідовності кроків, що використовують методи, утиліти й процедури. Ці послідовності кроків часто називають *парадигмами ТКПЗ*.

Застосування парадигм ТКПЗ гарантує систематичний, упорядкований підхід до промислової розробки, використання й супроводу ПЗ. Фактично, парадигми вносять у процес створення ПЗ організуючий інженерний початок, необхідність якого важко переоцінити.

Розглянемо найбільш популярні парадигми ТКПЗ.

I. Класичний життєвий цикл

Найстаршою парадигмою процесу розробки ПЗ є класичний життєвий цикл (автор Вінстон Ройс, 1970).

Дуже часто класичний життєвий цикл називають каскадною або водоспадною моделлю, підкреслюючи, що розробка розглядається як послідовність етапів, причому перехід на наступний, ієрархічно нижній етап відбувається тільки після повного завершення робіт на поточному етапі.

Охарактеризуємо зміст основних етапів.

Мається на увазі, що розробка починається на системному рівні й проходить через аналіз, проектування, кодування, тестування й супровід. При цьому моделюються дії стандартного інженерного циклу.

Системний аналіз задає роль кожного елемента в комп'ютерній системі, взаємодію елементів один з одним. Оскільки ПЗ є лише частиною великої системи, то аналіз починається з визначення вимог до всіх системних елементів і призначення підмножини цих вимог програмному "елементу". Необхідність системного підходу явно проявляється, коли формується інтерфейс ПО з іншими елементами (апаратурою, людьми, базами даних). На цьому ж етапі починається

розв'язок завдання планування проекту ПО. У ході планування проекту визначаються обсяг проектних робіт і їх ризик, необхідні працезатрати, формуються робочі завдання й план- графік робіт.

Аналіз вимог ставиться до програмного елемента - програмному забезпеченню. Уточнюються й деталізуються його функції, характеристики й інтерфейс.

Усі визначення документуються в специфікації аналізу. Тут же завершується розв'язок завдання планування проекту.

Проектування полягає в створенні вистав:

- архітектури ПЗ;
- модульної структури ПЗ;
- алгоритмічної структури ПЗ;
- структури даних;
- вхідного й вихідного інтерфейсу (вхідних і вихідних форм даних).

Вихідні дані для проектування втримуються в специфікації аналізу, тобто в ході проектування виконується трансляція вимог до ПЗ в безліч проектних вистав. При розв'язку завдань проектування основна увага приділяється якості майбутнього програмного продукту.

Кодування полягає в перекладі результатів проектування в текст мовою програмування.

Тестування - виконання програми для виявлення дефектів у функціях, логіку й формі реалізації програмного продукту.

Супровід - це внесення змін в експлуатоване ПО. Мети змін:

- виправлення помилок;
- адаптація до змін зовнішньої для ПЗ середовища;
- удосконалення ПЗ по вимогах замовника.

Супровід ПЗ полягає в повторному застосуванню кожного з попередніх кроків (етапів) життєвого циклу до існуючої програми але не в розробці нової програми.

Як і будь-яка інженерна схема, класичний життєвий цикл має гідності й недоліки.

Гідності класичного життєвого циклу: дає план і часовий графік по всіх етапах проекту, упорядковує хід конструювання.

Недоліки класичного життєвого циклу:

1) реальні проекти часто вимагають відхилення від стандартної послідовності кроків;

2) цикл заснований на точному формулюванні вихідних вимог до ПО (реально на початку проекту вимоги замовника визначена лише частково);

3) результати проекту доступні замовникові тільки наприкінці роботи.

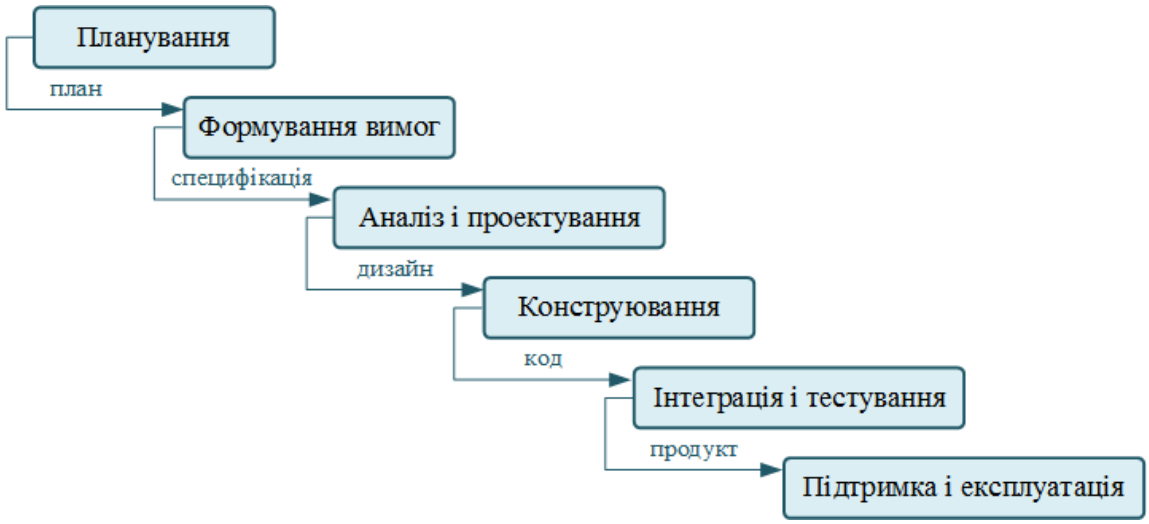


Рисунок 1.1 – Класичний життєвий цикл розробки ПЗ

II. Макетування

Досить часто замовник не може сформулювати докладні вимоги по введенню, обробці або висновку даних для майбутнього програмного продукту. З іншого боку, розроблювач може сумніватися в пристосованості продукту під операційну систему, формі діалогу з користувачем або в ефективності реалізованого алгоритму. У цих випадках доцільно використовувати макетування.

Основна мета макетування - зняти невизначеності у вимогах замовника.

Макетування (прототипування) - це процес створення моделі необхідного програмного продукту.

Модель може приймати одну із трьох форм:

1) *паперовий макет* або макет на основі ПК (зображує або малює людино-машинний діалог);

2) *працюючий макет* (виконує деяку частину необхідних функцій);

3) *існуюча програма* (характеристики якої потім повинні бути поліпшені).

Як показано на рис. 1.2, макетування ґрунтується на багаторазовому повторенні ітерацій, у яких беруть участь замовник і розроблювач.

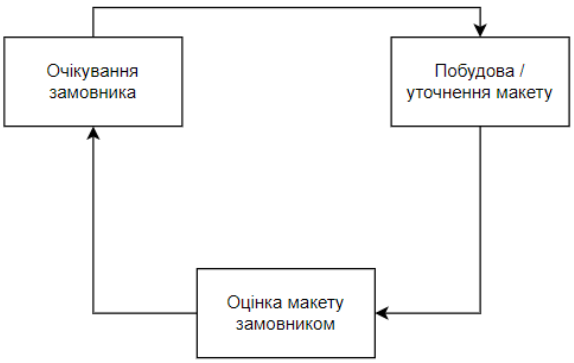


Рисунок 1.2 – Макетування

Послідовність дій при макетуванні представлена на рис. 1.3. Макетування починається зі збору й уточнення вимог до створюваного ПЗ Розроблювач і замовник зустрічаються й визначають усі цілі ПЗ, установлюють, які вимоги відомі, а які мають бути до визначені.

Потім виконується швидке проектування. У ньому увага зосереджується на тих характеристиках ПЗ, які повинні бути видимі користувачеві.

Швидке проектування приводить до побудови макета.

Макет оцінюється замовником і використовується для уточнення вимог до ПЗ.

Ітерації повторюються доти, поки макет не виявить усі вимоги замовника й, тим самим, не дасть можливість розроблювачеві зрозуміти, що повинне бути зроблене.

Гідність макетування: забезпечує визначення повних вимог до ПЗ.

Недоліки макетування:

- замовник може прийняти макет за продукт;
- розроблювач може прийняти макет за продукт.

Пояснимо суть недоліків. Коли замовник бачить працюючу версію ПЗ, він перестає усвідомлювати, що деталі макета скріплені "жувальною гумкою й дротом"; він забуває, що в погоні за працюючим варіантом залишені невирішеними питання якості й зручності супроводу ПЗ. Коли замовникові говорять, що продукт повинен бути перебудований, він починає обурюватися й вимагати, щоб макет "у

три приймання" був перетворений у робочий продукт. Дуже часто це негативно позначається на керуванні розробкою ПЗ.

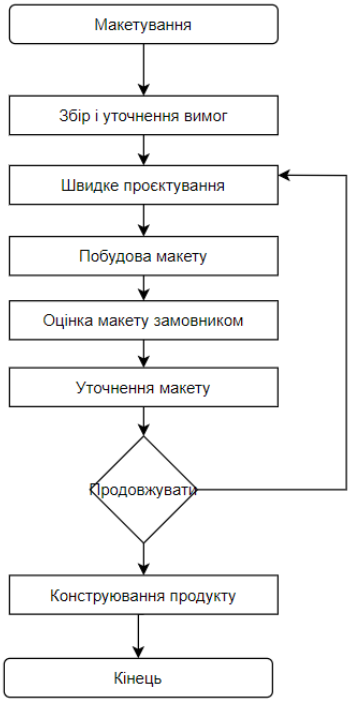


Рисунок 1.3 – Послідовність дій при макетуванні

З іншого боку, для швидкого одержання працюючого макета розроблювач часто йде на певні компроміси. Можуть використовуватися не самі підходящі мову програмування або операційна система. Для простої демонстрації можливостей може застосовуватися неефективний алгоритм. Через деякий час розроблювач забуває про причини, по яким ці засоби не підходять. У результаті далеко не ідеальний обраний варіант інтегрується в систему.

Очевидно, що подолання цих недоліків вимагає боротьби з життєвою спокусою - прийняти бажане за дійсне.

Існують 3 стратегії конструювання ПЗ:

- *однократний прохід* (водоспадна стратегія) - лінійна послідовність етапів конструювання (класичний життєвий цикл);
- *інкрементна стратегія*. На початку процесу визначаються всі користувацькі й системні вимоги частина, що залишився, конструювання виконується у вигляді послідовності версій. Перша версія реалізує частину запланованих можливостей версія, що випливає, реалізує додаткові можливості і т.д., поки не буде отримана повна система;

 еволюційна стратегія. Система також будується у вигляді послідовності версій, але на початку процесу визначені не всі вимоги. Вимоги уточнюються в результаті розробки версій.

Характеристики стратегій конструювання ПЗ відповідно до вимог стандарту IEEE/EIA 12207.2 наведені в табл. 2.1.

Таблиця 2.1 – Характеристики стратегій конструювання

Стратегія конструювання	**На початку процесу визначені всі вимоги?**	**Безліч циклів конструювання?**	**Проміжне ПЗ поширюється?**
Однократний прохід	Так	Ні	Ні
Інкримента (заплановане поліпшення продукту)	Так	Так	Може бути
Еволюційна	Ні	Так	Так

І. Інкрементна модель

Інкрементна модель являє класичний приклад інкрементної стратегії конструювання (рис. 2.1). Вона поєднує елементи послідовної водоспадної моделі з ітераційною філософією макетування.

Кожна лінійна послідовність тут виробляє, що поставляється інкремент ПЗ. Наприклад, ПЗ для обробки слів в 1- м інкременті реалізує функції базової обробки файлів, функції редагування й документування; в 2- му інкременті - більш складні можливості редагування й документування; в 3- му інкременті - перевірку орфографії й граматики; в 4- му інкременті - можливості компонування сторінки.

Перший інкремент приводить до одержання базового продукту, що реалізує базові вимоги (правда, багато допоміжних вимог залишаються нереалізованими).

План наступного інкременту передбачає модифікацію базового продукту, що забезпечує додаткові характеристики й функціональність.

По своїй природі інкрементний процес ітераційний, але, на відміну від макетування, інкрементна модель забезпечує на кожному інкременті працюючий продукт.

Рисунок 2.1 – Інкрементна модель

Забігаючи вперед, відзначимо, що сучасна реалізація інкрементного підходу - екстремальне програмування XP (Кент Бек, 1999). Воно орієнтоване на дуже малі збільшення функціональності.

II. Спіральна модель

Спіральна модель - класичний приклад застосування еволюційної стратегії конструювання.

Спіральна модель (автор Барри Боэм, 1988) базується на кращих властивостях класичного життєвого циклу й макетування, до яких додається новий елемент - аналіз ризику, відсутній у цих парадигмах.

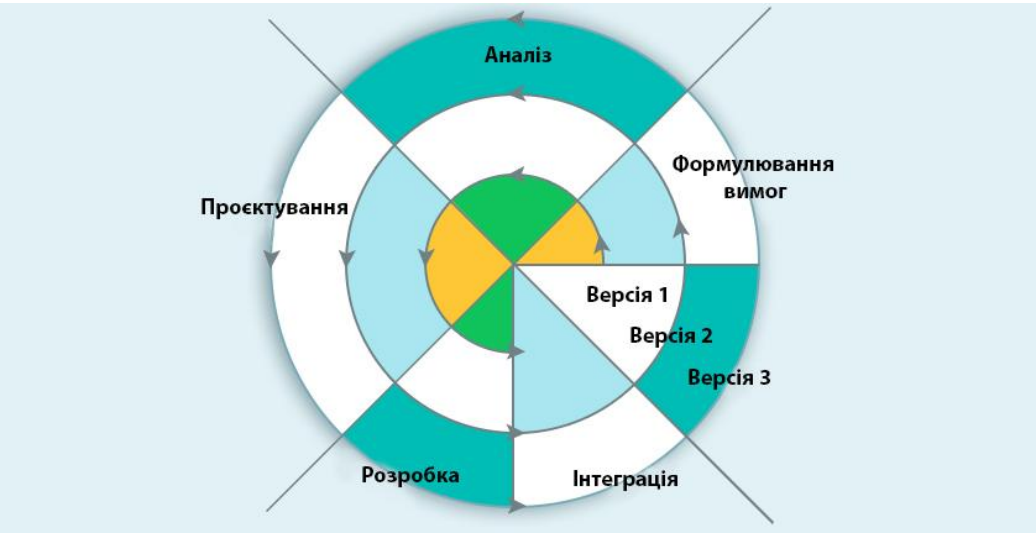


Рисунок 2.2 – Спіральна модель: 1 - початковий збір вимог і планування проекту; 2 - та ж робота, але на основі рекомендацій замовника; 3 - аналіз ризику на основі початкових вимог; 4 - аналіз ризику на основі реакції замовника; 5 –

перехід до комплексної системи; 6 - початковий макет системи; 7 - наступний рівень макета; 8 - сконструйована система; 9 - оцінювання замовником

Як показано на рис. 2.2, модель визначає чотири дії, що представляються чотирма квадрантами спіралі.

1. Планування - визначення цілей, варіантів і обмежень.

2. Аналіз ризику - аналіз варіантів і розпізнавання/вибір ризику.

3. Конструювання - розробка продукту наступного рівня.

4. Оцінювання - оцінка замовником поточних результатів конструювання.

Інтегруючий аспект спіральної моделі очевидний при обліку радіального виміру спирали. З кожною ітерацією по спіралі (просуванням від центру до периферії) будуються усе більш повні версії ПЗ.

У першому витку спирали визначаються початкові цілі, варіанти й обмеження, розпізнається й аналізується ризик. Якщо аналіз ризику показує невизначеність вимог, на допомогу розроблювачеві й замовникові приходить макетування (використовуване у квадранті конструювання). Для подальшого визначення проблемних і уточнених вимог може бути використане моделювання. Замовник оцінює інженерну (конструкторську) роботу й вносить пропозиції по модифікації (квадрант оцінки замовником). Наступна фаза планування й аналізу ризику базується на пропозиціях замовника. У кожному циклі по спіралі результати аналізу ризику формуються у вигляді "продовжувати, не продовжувати". Якщо ризик занадто великий, проект може бути зупинений.

У більшості випадків рух по спіралі триває, з кожним кроком просуваючи розроблювачів до більш загальної моделі системи. У кожному циклі по спіралі потрібне конструювання (нижній правий квадрант), яке може бути реалізоване класичним життєвим циклом або макетуванням. Помітимо, що кількість дій по розробці (що відбуваються в правому нижньому квадранті) зростає в міру просування від центру спіралі.

Гідності спіральної моделі:

1) найбільше реально (у вигляді еволюції) відображає розробку програмного забезпечення;

2) дозволяє явно враховувати ризик на кожному витку еволюції розробки;

3) включає крок системного підходу в ітераційну структуру розробки;

4) використовує моделювання для зменшення ризику й удосконалювання програмного виробу.

Недоліки спіральної моделі:

1) новизна (відсутня достатня статистика ефективності моделі);

2) підвищені вимоги до замовника;

3) труднощі контролю й керування часом розробки.

III. Компонентно-орієнтована модель

Компонентно-орієнтована модель є розвитком спіральної моделі й теж ґрунтується на еволюційній стратегії конструювання. У цій моделі конкретизується зміст квадранта конструювання - воно відбиває той факт, що в сучасних умовах нова розробка повинна ґрунтуватися на повторному використанні існуючих програмних компонентів (рис. 2.3).

Рисунок 2.3 – Компонентно-орієнтована модель

Програмні компоненти, створені в реалізованих програмних проектах, зберігаються в бібліотеці. У новому програмному проекті, виходячи з вимог замовника, виявляються кандидати в компоненти. Далі перевіряється наявність цих кандидатів у бібліотеці. Якщо вони знайдені, то компоненти визволяються з

бібліотеки й використовуються повторно. А якщо ні, то створюються нові компоненти, вони застосовуються в проекті й включаються в бібліотеку.

Гідності компонентно-орієнтованої моделі:

1) зменшує на 30% час розробки програмного продукту;

2) зменшує вартість програмної розробки до 70%;

3) збільшує в півтора рази продуктивність розробки.

Блок змістових модулів №2 Моделі конструювання

Змістовий модуль 2.1 Базис мови візуального моделювання

Тема 2.1.1 Уніфікована мова моделювання UML. Словник

Лекція №2

(2 години)

Мета: Ознайомитись з поняттям UML. Розглянути словник UML. Дати визначення предметам UML.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. UML;
2. Предмети в UML;
   a. Структурні предмети;
   b. Предмети поведінки;
   c. Предмети, що групують;
   d. Предмети, що пояснюють;
3. Відносини в UML;

4. Діаграми в UML.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичним матеріалом теми 2.1.1.
2. Вивчити основні поняття лекції.
3. Відповісти на контрольні питання.

Контрольні питання:

1. Що таке UML?
2. З чого складається словник UML?
3. Які предмети UML бувають?
4. Що таке структурні предмети? Перелічіть їх.
5. Які існують предмети поведінки?
6. Що таке пакет?
7. Для чого використовують примітку?
8. Що таке діаграма?

UML - стандартна мова для написання моделей аналізу, проектування й реалізації об'єктно-орієнтованих програмних систем. UML може використовуватися для візуалізації, специфікації, конструювання й документування результатів програмних проектів. UML - це не візуальна мова програмування, але його моделі прямо транслюються в текст на мовах програмування (Java, C++, Visual Basic, Ada 95, Object Pascal) і навіть у таблиці для реляційної БД.

Словник UML утворюють три різновиди будівельних блоків: предмети, відносини, діаграми.

Предмети – це абстракції, які є основними елементами в моделі, відносини зв'язують ці предмети, діаграми групують колекції предметів.

Предмети в UML

В UML є чотири різновиди предметів:

- структурні предмети;
- предмети поведінки;

- предмети, що групують;
- предмети, що пояснюють.

Ці предмети є базовими об'єктно-орієнтованими будівельними блоками. Вони використовуються для написання моделей.

Структурні предмети є іменниками в Uml- Моделях. Вони представляють статичні частини моделі - понятійні або фізичні елементи. Перелічимо вісім різновидів структурних предметів.

1. *Клас* - опис безлічі об'єктів, які розділяють однакові властивості, операції, відносини й семантику (зміст). Клас реалізує один або кілька інтерфейсів. Як показано на рис. 2.1, графічно клас відображається у вигляді прямокутника, що звичайно включає секції з іменем, властивостями (атрибутами) і операціями.

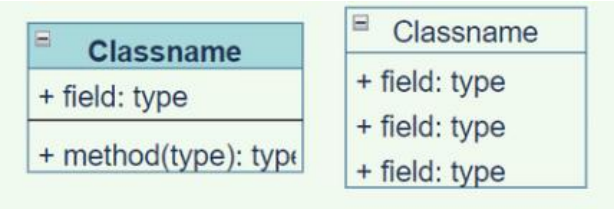


Рисунок 2.1 – Клас

2. *Інтерфейс* - набір операцій, які визначають послуги класу або компонента. Інтерфейс описує поведінку елемента, видиму ззовні. Інтерфейс може представляти повні послуги класу або компонент або частина таких послуг. Інтерфейс визначає набір специфікацій операцій (їх сигнатури), а не набір реалізацій операцій. Графічно інтерфейс зображується у вигляді кружка з іменем, як показано на рис. 2.2. Ім'я інтерфейсу звичайно починається з букви "I". Інтерфейс рідко показують самостійно. Звичайно його приєднують до класу або компонента, який реалізує інтерфейс.

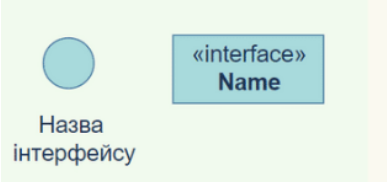


Рисунок 2.2 – Інтерфейс

3. *Кооперація (співробітництво)* визначає взаємодія і є сукупністю ролей і інших елементів, які працюють разом для забезпечення колективної поведінки більш складного, чому проста сума всіх елементів. Таким чином, кооперації мають як структурний, так і поведінковий вимір. Конкретний клас може

брати участь у декількох кооперaціях. Ці кооперації представляють реалізацію паттернів (зразків), які формують систему. Як показано на рис. 2.3, графічно кооперація зображується як пунктирний еліпс, у який вписується її ім'я.

Рисунок 2.3 – Кооперація

4. *Актор* - набір погоджених ролей, які можуть відіграти користувачі при взаємодії із системою (її елементами Use Case). Кожна роль вимагає від системи певного поведінки. Як показано на рис. 2.4, актор зображується як дротовий чоловічок з іменем.

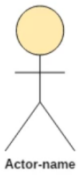


Рисунок 2.4 – Актор

5. *Елемент Use Case (Прецедент)* - опис послідовності дій (або декількох послідовностей), виконуваних системою в інтересах окремого актора й виробляючих видимий для актора результат. У моделі елемент Use Case застосовується для структурування предметів поведінки. Елемент Use Case реалізується кооперацією. Як показано на рис. 2.5, елемент Use Case зображується як еліпс, у який уписується його ім'я.

Рисунок 2.5 – Прецедент

6. *Активний клас* - клас, чиї об'єкти мають один або кілька процесів (або потоків) і тому можуть ініціювати керуючу діяльність. Активний клас схожий на

звичайний клас за винятком того, що його об'єкти діють одночасно з об'єктами інших класів. Як показано на рис. 2.6, активний клас зображується як стовщений прямокутник, що звичайно включає ім'я, властивості (атрибути) і операції.

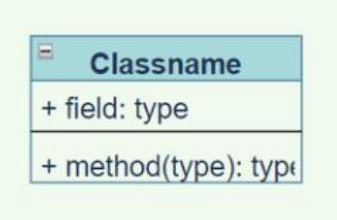


Рисунок 2.6 – Активний клас

7. *Компонент* - фізична й замінна частина системи, яка відповідає набору інтерфейсів і забезпечує реалізацію цього набору інтерфейсів. У систему включаються як компоненти, що є результатами процесу розробки (файли вихідного коду), так і різні різновиди використовуваних компонентів (COM+-компонента, Java Beans). Звичайно компонент - це фізичне впакування різних логічних елементів (класів, інтерфейсів і співробітництв). Як показано на рис. 2.7, компонент зображується як прямокутник із вкладками, що звичайно включає ім'я.

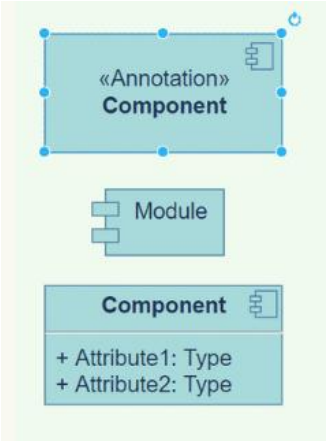


Рисунок 2.7 – Компонент

8. *Вузол* - фізичний елемент, який існує в період роботи системи й представляє ресурс, що звичайно має пам'ять і можливості обробки. У вузлі розміщається набір компонентів, який може переміщатися від вузла до вузла. Як показано на рис. 2.8, вузол зображується як куб з іменем.

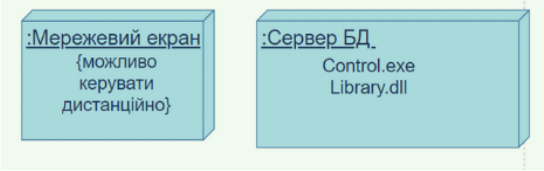


Рисунок 2.8 – Вузол

Предмети поведінки - динамічні частини Uml- Моделей. Вони є дієсловами моделей, виставою поведінки в часі й просторі. Існує два основні різновиди предметів поведінки.

1. *Взаємодія* - поведінка, що містить у собі набір повідомлень, якими обмінюється набір об'єктів у конкретному контексті для досягнення певної мети. Взаємодія може визначати динаміку як сукупності об'єктів, так і окремої операції. Елементами взаємодії є повідомлення, послідовність дій (поведінка, викликуване повідомленням) і зв'язку (з'єднання між об'єктами). Як показано на рис. 2.9, повідомлення зображується у вигляді спрямованої лінії з іменем її операції.

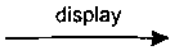


Рисунок 2.9 – Повідомлення

2. *Кінцевий автомат* - поведінка, яка визначає послідовність станів об'єкта або взаємодії, виконувані в ході його існування у відповідь на події (і з урахуванням обов'язків по цих подіях). За допомогою кінцевого автомата може визначатися поведінка індивідуального класу або кооперації класів. Елементами кінцевого автомата є стани, переходи (від стану до стану), події (предмети, що викликає переходи) і дії (реакції на перехід). Як показано на рис. 2.10, стан зображується як закруглений прямокутник, що звичайно включає його ім'я і його під стани (якщо вони є).

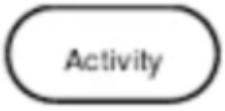


Рисунок 2.10 – Стан

Предмети, що групують, - організаційні частини Uml-Моделей. Це ящики, по яких може бути розкладена модель. Передбачено один різновид предмета, що групує, - пакет.

Пакет - загальний механізм для розподілу елементів по групах. У пакет можуть міститися структурні предмети, предмети поведінки й навіть інші угруповання предметів. На відміну від компонента (який існує в період виконання), пакет - чисто концептуальне поняття. Це означає, що пакет існує тільки в період

розробки. Як показано на рис. 2.11, пакет зображується як папка із закладкою, на якій позначено його ім'я й, іноді, його зміст.

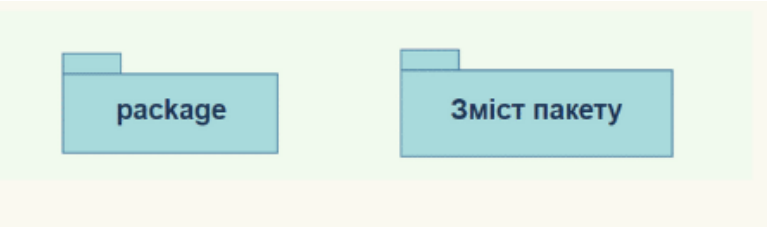


Рисунок 2.11 – Пакет

Предмети, що пояснюють, - частини, що роз'ясняють, Uml-Моделей. Вони є зауваженнями, які можна застосувати для опису, пояснення й коментування будь-якого елемента моделі. Передбачено один різновид предмета, що пояснює, - примітка.

Примітка - символ для відображення обмежень і зауважень, що приєднуються до елемента або сукупності елементів. Як показано на рис. 2.12, примітка зображується у вигляді прямокутника із загнутим кутом, у який уписується текстовий або графічний коментар.

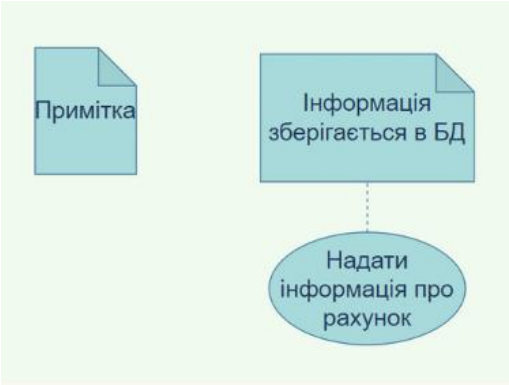


Рисунок 2.12 – Примітка

Відносини в UML

В UML є наступні різновиди відносин:

1) залежність;

2) асоціація;

3) узагальнення;

4) композиція;

5) асоціація.

Ці відносини є базовими будівельними блоками відносин. Вони використовуються при написанні моделей.

Діаграми в UML

Діаграма - графічна представлення безлічі елементів, найбільше часто зображується як зв'язний граф з вершин (предметів) і дуг (відносин). Діаграми рисуються для візуалізації системи з різних точок зору, потім вони відображаються в систему. Звичайно діаграма дає неповне представлення елементів, які становлять систему. Хоча той самий елемент може з'являтися у всіх діаграмах, на практиці він з'являється тільки в деяких діаграмах. Теоретично діаграма може містити будь-яку комбінацію предметів і відносин, на практиці обмежуються малою кількістю комбінацій, які відповідають п'яти представленням архітектури ПС.

Тема 2.1.2 Діаграми в UML. Механізми розширення.

Лекція №3

(2 години)

Мета: Розглянути різні види діаграма в UML, а також розглянути доступні види механізмів розширення.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Діаграми в UML;
2. Механізми розширення.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:
   1. Ознайомитись з теоретичним матеріалом теми 2.1.2.
   2. Вивчити основні поняття лекції.

3. Відповісти на контрольні питання.

Контрольні питання:

1. Які існують діаграми в UML?
2. Перелічіть діаграми взаємодії?
3. Які існують механізми розширення в UML?
4. Навіщо в UML необхідні розширення?
5. Поясните механізм обмежень в UML.
6. Поясните механізм тегових величин в UML.
7. У чому суть механізму стереотипів UML?

UML включає дев'ять видів діаграм:

1) діаграми класів;
2) діаграми об'єктів;
3) діаграми Use Case (діаграми прецедентів);
4) діаграми послідовності;
5) діаграми співробітництва (кооперації);
6) діаграми схем станів;
7) діаграми діяльності;
8) компонентні діаграми;
9) діаграми розміщення (розгортання).

Діаграма класів показує набір класів, інтерфейсів, співробітництв і їх відносин. При моделюванні об'єктно-орієнтованих систем діаграми класів використовуються найбільше часто. Діаграми класів забезпечують статична проектна вистава системи. Діаграми класів, що включають активні класи, забезпечують статична вистава процесів системи.

Діаграма об'єктів показує набір об'єктів і їх відносини. Діаграма об'єктів представляє статичний "моментальний знімок" з екземплярів предметів, які перебувають у діаграмах класів. Як і діаграми класів, ці діаграми забезпечують статична проектна вистава або статична вистава процесів системи (але з погляду реальних або фототипових випадків).

Діаграма Use Case (діаграма прецедентів) показує набір елементів Use Case, акторів і їх відносин. За допомогою діаграм Use Case для системи створюється

статична вистава Use Case. Ці діаграми особливо важливі при організації й моделюванні поведінки системи, завданні вимог замовника до системи.

Діаграми послідовності й діаграми співробітництва - це різновиди діаграм взаємодії.

Діаграма взаємодії показує взаємодію, що включає набір об'єктів і їх відносин, а повідомлення, що також пересилаються між об'єктами. Діаграми взаємодії забезпечують динамічна вистава системи.

Діаграма послідовності - це діаграма взаємодії, яка виділяє впорядкування повідомлень за часом.

Діаграма співробітництва (діаграма кооперації) - це діаграма взаємодії, яка виділяє структурну організацію об'єктів, що посилають повідомлення, що й ухвалюють. Діаграми послідовності й діаграми співробітництва ізоморфні, що означає, що одну діаграму можна трансформувати в іншу діаграму.

Діаграма схем станів показує кінцевий автомат, представляє стани, переходи, події й дії. Діаграми схем станів забезпечують динамічна вистава системи. Вони особливо важливі при моделюванні поведінки інтерфейсу, класу або співробітництва. Ці діаграми виділяють така поведінка об'єкта, яка управляється подіями, що особливо корисно при моделюванні реактивних систем.

Діаграма діяльності - спеціальний різновид діаграми схем станів, яка показує потік від дії до дії усередині системи. Діаграми діяльності забезпечують динамічна вистава системи. Вони особливо важливі при моделюванні функціональності системи й виділяють потік керування між об'єктами.

Компонентна діаграма показує організацію набору компонентів і залежності між компонентами. Компонентні діаграми забезпечують статична вистава реалізації системи. Вони пов'язані з діаграмами класів у тому розумінні, що в компонент звичайно відображається один або кілька класів, інтерфейсів або кооперацій.

Діаграма розміщення (діаграма розгортання) показує конфігурацію обробних вузлів періоду виконання, а також компоненти, що живуть у них. Діаграми розміщення забезпечують статична вистава розміщення системи. Вони пов'язані з компонентними діаграмами в тому розумінні, що вузол звичайно включає один або кілька компонентів.

Механізми розширення в UML

UML - розвита мова, що має великі можливості, але навіть вона не може відбити всі нюанси, які можуть виникнути при створенні різних моделей. Тому UML створювалася як відкрита мова, що допускає контрольовані *розширення*. *Механізмами розширення* в UML є:

- обмеження;
- тегові величини;
- стереотипи.

Обмеження (constraint) розширює семантику будівельного Uml-Блоку, дозволяючи додати нові правила або модифікувати існуючі. Обмеження показують як текстовий рядок, закладений в фігурні дужки {}. Наприклад, на рис. 6.1 введене просте обмеження на властивість *сума* класу *Сесія Банкомата* - його значення повинне бути кратне 20. Крім того, тут показане обмеження на два елементи (дві асоціації), воно розташовується біля пунктирної лінії, що з'єднує елементи, і має наступний сенс - власником конкретного рахунку не може бути й організація, і персона.

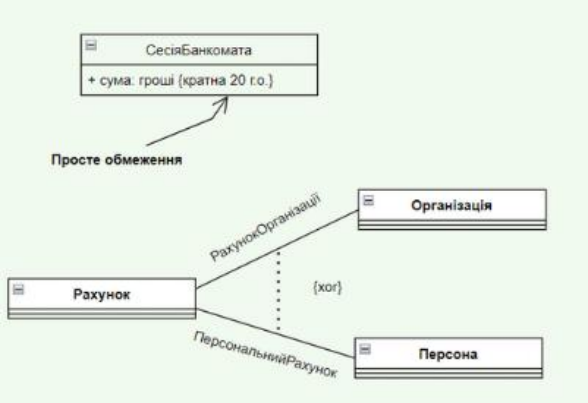


Рисунок 1 – Обмеження

Тегова величина (tagged value) розширює характеристики будівельного Uml-Блоку, дозволяючи створити нову інформацію в специфікації конкретного елемента. Тегову величину показують як рядок у фігурних дужках {}. Рядок має вигляд

ім'я тегової величини = значення.

Іноді (у випадку визначених тегів) вказується тільки ім'я тегової величини.

Відзначимо, що при роботі із продуктом, що мають багато реалізацій, корисно відслідковувати версію й автора певних блоків. Версія й автор не належать

до основних поняттям UML. Вони можуть бути додані до будь-якого будівельного блоку (наприклад, до класу) уведенням у блок нових тегових величин. Наприклад, на рис. 2 клас *Текстовий процесор* розширений шляхом явної вказівки його версії й автора.

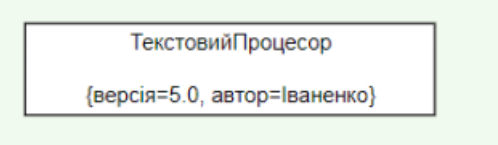


Рисунок 2 – Розширення класу

Стереотип (stereotype) розширює словник мови, дозволяє створювати нові види будівельних блоків, похідні від існуючих і враховуючі специфіку нової проблеми. Елемент зі стереотипом є варіацією існуючого елемента, що має таку ж форму, але одмінну по суті. У нього можуть бути додаткові обмеження й теговые величини, а також інша візуальна вистава. Він інакше обробляється при генерації програмного коду. Відображають стереотип як ім'я, що вказується в подвійних кутових дужках (або в кутових лапках).

Приклади елементів зі стереотипами наведені на рис. 3. Стереотип "exception" говорить про те, що клас *Втрата значимості* тепер розглядається як спеціальний клас, якому, покладемо, дозволяється тільки генерація й обробка сигналів виключень. Особливі можливості мета класу одержав клас *Елемент моделі*. Крім того, тут показане застосування стереотипу "call" до відношення залежності (у нього з'явився новий зміст).

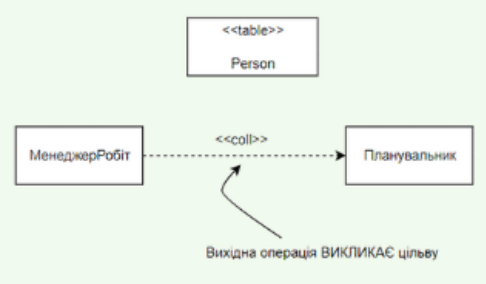


Рисунок 3 – Стереотипи

Таким чином, механізми розширення дозволяють адаптувати UML під потреби конкретних проектів і під нові програмні технології. Можливе додавання нових будівельних блоків, модифікація специфікацій існуючих блоків і навіть зміна їх семантики. Звичайно, дуже важливо забезпечити контрольоване введення розширень.

Тема 2.1.3 Діаграми Use Case

Лекція №4

(2 години)

Мета: Розглянути основні поняття діаграми Use Case. Ознайомитись з графічною нотацією діаграми.

Структура заняття:

I. Організаційний момент:
   1. Готовність групи до заняття;
   2. Психоемоційний настрій;
   3. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   1. Повідомлення теми та мети заняття;
   2. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Варіант використання
2. Актори
3. Інтерфейси
4. Примітки
5. Відносини на діаграмі варіантів використання
   i. Відношення асоціації
   ii. Відношення розширення
   iii. Відношення узагальнення
   iv. Відношення включення
6. Приклад побудови діаграми варіантів використання

IV. Підведення підсумків занять;

V. Домашнє завдання:
   1. Ознайомитись з теоретичним матеріалом теми 2.1.3.
   2. Вивчити основні поняття лекції.
   3. Відповісти на контрольні питання.

Контрольні питання:

1. З яких елементів полягає діаграма Use Case?
2. Які відносини дозволені між елементами діаграми Use Case?
3. Для чого застосовують діаграми Use Case?
4. Чим відрізняються друг від друга відносини включення й розширення з погляду керування?

Візуальне моделювання в UML можна представити як деякий процес порівневого спуска від найбільш загальної і абстрактної концептуальної моделі вихідної системи до логічної, а потім і до фізичної моделі відповідної програмної системи. Для досягнення цих цілей спочатку будується модель у формі так званої діаграми варіантів використання (use case diagram), яка описує функціональне призначення системи або, інакше кажучи, те, що система буде робити в процесі свого функціонування. Діаграма варіантів використання є вихідною концептуальною виставою або концептуальною моделлю системи в процесі її проектування й розробки.

Розробка діаграми варіантів використання переслідує wsks:

- Визначити загальні границі й контекст моделируемой предметної області на початкових етапах проектування системи.
- Сформулювати загальні вимоги до функціональної поведінки проектованої системи.
- Розробити вихідну концептуальну модель системи для її наступної деталізації у формі логічних і фізичних моделей.
- Підготувати вихідну документацію для взаємодії розроблювачів системи з її замовниками й користувачами.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей або акторів, взаємодіючих із системою за допомогою так званих варіантів використання. При цьому актором (actor) або діючою особою називається будь-яка сутність, взаємодіюча із системою ззовні. Це може бути людина, технічне обладнання, програма або будь-яка інша система, яка може служити джерелом впливу на моделируемую систему так, як визначить сам розроблювач. У свою чергу, варіант використання (use case) служить для опису сервісів, які система надає акторові. Інакше кажучи, кожний варіант використання визначає деякий набір дій, чинений системою при діалозі з актором. При цьому

нічого не говориться про те, яким образом буде реалізована взаємодія акторів із системою.

Варіант використання

Конструкція або стандартний елемент мови UML *варіант використання* застосовується для специфікації загальних особливостей поведінки системи або будь-якої інший сутності предметної області без розгляду внутрішньої структури цієї сутності. Кожний варіант використання визначає послідовність дій, які повинні бути виконані проектованою системою при взаємодії її з відповідним актором. Діаграма варіантів може доповнюватися пояснювальним текстом, який розкриває зміст або семантику складових її компонентів. Такий пояснювальний текст одержав назву примітки або сценарію.

Окремий варіант використання позначається на діаграмі еліпсом, усередині якого втримується його коротка назва або ім'я у формі дієслова з пояснювальними словами (рис. 1).

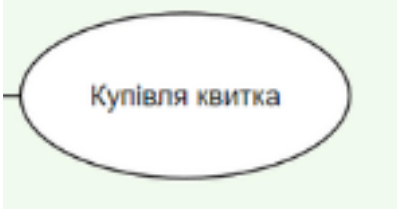


Рисунок 1 - Графічне позначення варіанта використання

Ціль варіанта використання полягає в тому, щоб визначити закінчений аспект або фрагмент поведінки деякої сутності без розкриття внутрішньої структури цієї сутності. У якості такої сутності може виступати вихідна система або будь-який інший елемент моделі, який має власну поведінку, подібно підсистемі або класу в моделі системи.

Кожний варіант використання відповідає окремому сервісу, який надає моделируемую сутність або систему по запиту користувача (актора), тобто визначає спосіб застосування цієї сутності. Сервіс, який инициализируется по запиту користувача, являє собою закінчену послідовність дій. Це означає, що після того як система закінчить обробку запиту користувача, вона повинна вернутися у вихідний стан, у якому готова до виконання наступних запитів.

Варіанти використання описують не тільки взаємодії між користувачами й сутністю, але також реакції сутності на одержання окремих повідомлень від

користувачів і сприйняття цих повідомлень за межами сутності. Варіанти використання можуть містити в собі опис особливостей способів реалізації сервісу й різних виняткових ситуацій, таких як коректна обробка помилок системи. Безліч варіантів використання в цілому повинне визначати всі можливі сторони очікуваного поведінки системи. Для зручності безліч варіантів використання може розглядатися як окремий пакет.

Із системно-аналітичної точки зору варіанти використання можуть застосовуватися як для специфікації зовнішніх вимог до проектованої системи, так і для специфікації функціональної поведінки вже існуючої системи.

Актори

Актор являє собою будь-яку зовнішню стосовно моделируемой системі сутність, яка взаємодіє із системою й використовує її функціональні можливості для досягнення певних цілей або розв'язку приватних завдань. При цьому актори служать для позначення погодженого безлічі ролей, які можуть відіграти користувачі в процесі взаємодії із проектованою системою. Кожний актор може розглядатися як якась окрема роль щодо конкретного варіанта використання. Стандартним графічним позначенням актора на діаграмах є фігурка "чоловічка", під якою записується конкретне ім'я актора (рис. 2).

Рисунок 2 - Графічне позначення актора

У деяких випадках актор може позначатися у вигляді прямокутника класу із ключовим словом "актор" і звичайними складовими елементами класу.

Актори використовуються для моделювання зовнішніх стосовно проектованої системи сутностей, які взаємодіють із системою й використовують її як окремі користувачів. У якості акторів можуть виступати інші системи, підсистеми проектованої системи або окремі класи. Важливо розуміти, що кожний актор визначає деяку погоджену безліч ролей, у яких можуть виступати користувачі даної системи в процесі взаємодії з нею. У кожний момент часу із системою взаємодіє цілком певний користувач, при цьому він відіграє або виступає в одній з таких ролей. Найбільш наочний приклад актора - конкретний користувач системи зі своїми власними параметрами аутентификации.

Будь-яка сутність, яка узгодиться з подібним неформальним визначенням актора, являє собою екземпляр або приклад актора. Для моделируемой системи акторами можуть бути як суб'єкти-користувачі, так і інші системи. Оскільки користувачі системи завжди є зовнішніми стосовно цієї системи, то вони завжди представляються у вигляді акторів.

Тому що в загальному випадку актор завжди перебуває поза системою, його внутрішня структура ніяк не визначається. Для актора має значення тільки його зовнішня вистава, тобто те, як він сприймається з боку системи. Актори взаємодіють із системою за допомогою передачі й приймання повідомлень від варіантів використання. Повідомлення являє собою запит актором сервісу від системи й одержання цього сервісу. Ця взаємодія може бути виражене за допомогою асоціацій між окремими акторами й варіантами використання або класами. Крім цього, з акторами можуть бути зв'язані інтерфейси, які визначають, яким образом інші елементи моделі взаємодіють із цими акторами.

Два й більш актори можуть мати загальні властивості, тобто взаємодіяти з тим самим безліччю варіантів використання однаковим образом. Така спільність властивостей і поведінки представляється у вигляді розглянутого нижче відношення узагальнення з іншим, можливо, абстрактним актором, який моделює відповідну спільність ролей.

Інтерфейси

Інтерфейс (interface) служить для специфікації параметрів моделі, які видимі ззовні без вказівки їх внутрішньої структури. У мові UML інтерфейс є класифікатором і характеризує тільки обмежену частину поведінки моделируемой сутності. Стосовно до діаграм варіантів використання, інтерфейси визначають сукупність операцій, які забезпечують необхідний набір сервісів або функціональності для акторів. Інтерфейси не можуть містити ні атрибутів, ні станів, ні спрямованих асоціацій. Вони містять тільки операції без вказівки особливостей їх реалізації. Формально інтерфейс еквівалентний абстрактному класу без атрибутів і методів з наявністю тільки абстрактних операцій.

На діаграмі варіантів використання інтерфейс зображується у вигляді маленького кола, поруч із яким записується його ім'я. Якщо ім'я записується на

англійському, то воно повинне починатися із заголовної букви I, наприклад, Isecurelnformation, Isensor (рис. 3).

Рисунок 3 - Графічне зображення інтерфейсів на діаграмах варіантів використання

Графічний символ окремого інтерфейсу може з'єднуватися на діаграмі суцільною лінією з тим варіантом використання, який його підтримує. Суцільна лінія в цьому випадку вказує на той факт, що пов'язаний з інтерфейсом варіант використання повинен реалізовувати всі операції, необхідні для даного інтерфейсу, а можливо й більше. Крім цього, інтерфейси можуть з'єднуватися з варіантами використання пунктирною лінією зі стрілкою, що означає, що варіант використання призначений для специфікації тільки того сервісу, який необхідний для реалізації даного інтерфейсу.

Рисунок 4 - графічне зображення взаємозв'язків інтерфейсів з варіантами використання

Примітки

Примітки (notes) у мові UML призначені для включення в модель довільної текстової інформації, що має безпосереднє відношення до контексту розроблювального проекту. У якості такої інформації можуть бути коментарі розроблювача (наприклад, дата й версія розробки діаграми або її окремих компонентів), обмеження (наприклад, на значення окремих зв'язків або екземпляри сутностей) і позначені значення. Стосовно до діаграм варіантів використання примітка може носити саму загальну інформацію, що ставиться до загального контексту системи.

Графічно примітки позначаються прямокутником з "загнутим" верхнім правим куточком (рис. 5). Усередині прямокутника втримується текст примітки. Примітка може ставитися до будь-якого елемента діаграми, у цьому випадку їх

з'єднує пунктирна лінія. Якщо примітка ставиться до декільком елементам, то від нього проводяться, відповідно, кілька ліній. Зрозуміло, примітки можуть бути присутнім не тільки на діаграмі варіантів використання, але й на інших канонічних діаграмах.

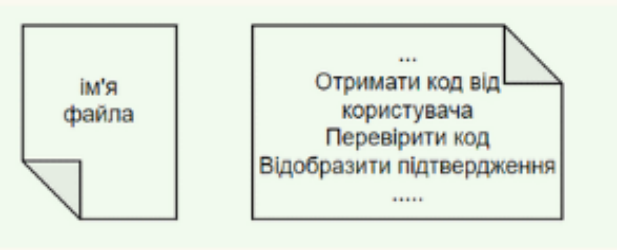


Рисунок 5 - Приклади приміток у мові UML

Лекція №5

Відносини на діаграмі варіантів використання

Між компонентами діаграми варіантів використання можуть існувати різні відносини, які описують взаємодія екземплярів одних акторів і варіантів використання з екземплярами інших акторів і варіантів. Один актор може взаємодіяти з декількома варіантами використання. У цьому випадку цей актор звертається до декільком сервісам даної системи. У свою чергу один варіант використання може взаємодіяти з декількома акторами, надаючи для всіх їх свій сервіс. Слід помітити, що два варіанти використання, певні для однієї й тієї ж сутності, не можуть взаємодіяти один з одним, оскільки кожний з них самостійно описує закінчений варіант використання цієї сутності. Більше того, варіанти використання завжди передбачають деякі сигнали або повідомлення, коли взаємодіють із акторами за межами системи. У той же час можуть бути визначені інші способи для взаємодії з елементами усередині системи.

У мові UML є кілька стандартних видів відносин між акторами й варіантами використання:

- Відношення асоціації (association relationship)
- Відношення розширення (extend relationship)
- Відношення узагальнення (generalization relationship)
- Відношення включення (include relationship)

При цьому загальні властивості варіантів використання можуть бути представлено трьома різними способами, а саме за допомогою відносин розширення, узагальнення й включення.

Відношення асоціації

Відношення асоціації є одним з фундаментальних понять у мові UML і тією чи іншою мірою використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Стосовно до діаграм варіантів використання воно служить для позначення специфічної ролі актора в окремому варіанті використання. Інакше кажучи, асоціація специфицирует семантичні особливості взаємодії акторів і варіантів використання в графічній моделі системи. Таким чином, це відношення встановлює, яку конкретну роль відіграє актор при взаємодії з екземпляром варіанта використання. На діаграмі варіантів використання, так само як і на інших діаграмах, відношення асоціації позначається суцільною лінією між актором і варіантом використання. Ця лінія може мати додаткові умовні позначки, такі, наприклад, як ім'я й кратність (рис. 6).

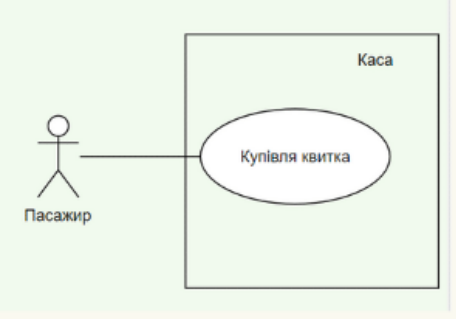


Рисунок 6 - Приклад графічної вистави відносини асоціації між актором і варіантом використання

Відношення розширення

Відношення розширення визначає взаємозв'язок екземплярів окремого варіанта використання з більш загальним варіантом, властивості якого визначаються на основі способу спільного об'єднання даних екземплярів. У метамодели відношення розширення є спрямованим і вказує, що стосовно до окремих прикладів деякого варіанта використання повинні бути виконані конкретні умови, певні для розширення даного варіанта використання. Так, якщо має місце відношення розширення від варіанта використання A до варіанта використання B,

те це означає, що властивості екземпляра варіанта використання В можуть бути доповнені завдяки наявності властивостей у розширеного варіанта використання А.

Відношення розширення між варіантами використання позначається пунктирною лінією зі стрілкою (варіант відносини залежності), спрямованої від того варіанта використання, який є розширенням для вихідного варіанта використання. Дана лінія зі стрілкою позначається ключовим словом "extend" ("розширює"), як показано на рис. 7.

Рисунок 7 - Приклад графічного зображення відносини розширення між варіантами використання

Відношення розширення відзначає той факт, що один з варіантів використання може приєднувати до своєї поведінки деяка додаткова поведінка, певне для іншого варіанта використання. Дане відношення містить у собі деяка умова й посилання на крапки розширення в базовому варіанті використання. Щоб розширення мало місце, повинне бути виконана певна умова даного відношення. Посилання на крапки розширення визначають ті місця в базовому варіанті використання, у які повинне бути поміщене відповідне розширення при виконанні умови.

Один з варіантів використання може бути розширенням для декількох базових варіантів, а також мати в якості власних розширень трохи інших варіантів. Базовий варіант використання може додатково ніяк не залежати від своїх розширень.

Семантика відношення розширення визначається в такий спосіб. Якщо екземпляр варіанта використання виконує деяку послідовність дій, яка визначає його поведінка, і при цьому є крапка розширення на екземпляр іншого варіанта використання, яка є першою із усіх крапок розширення у вихідного варіанта, то

перевіряється умова даного відношення. Якщо умова виконується, вихідна послідовність дій розширюється за допомогою включення дій екземпляра іншого варіанта використання. Слід помітити, що умова відносини розширення перевіряється лише один раз - при першому посиланні на крапку розширення, і якщо воно виконується, то всі розширювальні варіанти використання вставляються в базовий варіант.

У представленому вище прикладі (рис. 8) при оформленні замовлення на придбання товару тільки в деяких випадках може знадобитися надання клієнтові каталогу всіх товарів. При цьому умовою розширення є запит від клієнта на одержання каталогу товарів. Очевидно, що після одержання каталогу клієнтові необхідно якийсь час на його вивчення, протягом якого оформлення замовлення припиняє. Після ознайомлення з каталогом клієнт вирішує або на користь вибору окремого товару, або відмови від покупки взагалі. Сервіс або варіант використання "Оформити замовлення на придбання товару" може відреагувати на вибір клієнта вже після того, як клієнт одержить для ознайомлення каталог товарів.

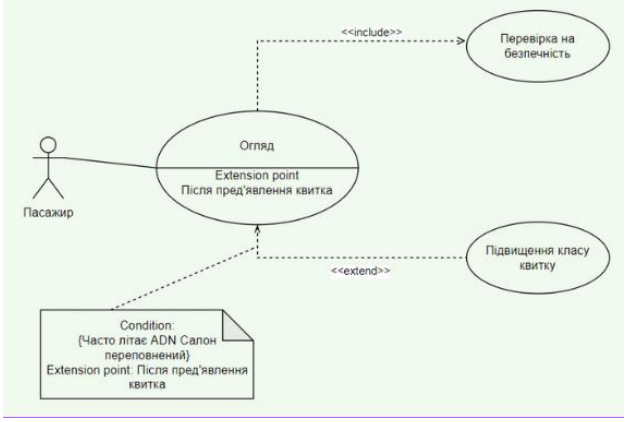


Рисунок 8 - Графічне зображення відносини розширення із примітками умов виконання варіантів використання

Відношення узагальнення

Відношення узагальнення служить для вказівки того факту, що деякий варіант використання А може бути узагальнений до варіанта використання В. У цьому випадку варіант А буде спеціалізацією варіанта В. При цьому В називається предком або батьком по відношенню А, а варіант А - нащадком стосовно варіанта використання В. Слід підкреслити, що нащадок успадковує всі властивості й поведінка свого батька, а також може бути доповнений новими властивостями й особливостями поведінки. Графічно дане відношення позначається суцільною

лінією зі стрілкою у формі незафарбованого трикутника, яка вказує на батьківський варіант використання (рис. 9). Ця лінія зі стрілкою має спеціальна назва - стрілка "узагальнення".

Рисунок 9 - Приклад графічного зображення відносини узагальнення між варіантами використання

Відношення узагальнення між варіантами використання застосовується в тому випадку, коли необхідно відзначити, що дочірні варіанти використання мають усі атрибути й особливостями поведінки батьківських варіантів. При цьому дочірні варіанти використання беруть участь у всіх відносинах батьківських варіантів. У свою чергу, дочірні варіанти можуть наділятися новими властивостями поведінки, які відсутні в батьківських варіантів використання, а також уточнювати або модифікувати наслідувані від них властивості поведінки.

Стосовно до даного відношення, один варіант використання може мати кілька батьківських варіантів. У цьому випадку реалізується множинне спадкування властивостей і поведінки відносини предків: З іншого боку, один варіант використання може бути предком для декількох дочірніх варіантів, що відповідає таксономическому характеру відносини узагальнення.

Між окремими акторами також може існувати відношення узагальнення. Дане відношення є спрямованим і вказує на факт спеціалізації одних акторів щодо інших. Наприклад, відношення узагальнення від актора А до актора У відзначає той факт, що кожний екземпляр актора А є одночасно екземпляром актора В и має всі його властивості. У цьому випадку актор У є батьком стосовно актора А, а актор А, відповідно, нащадком актора В. При цьому актор А має здатність відіграти таке ж безліч ролей, що й актор В. Графічно дане відношення також

позначається стрілкою узагальнення, тобто суцільною лінією зі стрілкою у формі незафарбованого трикутника, яка вказує на батьківського актора (рис. 10).

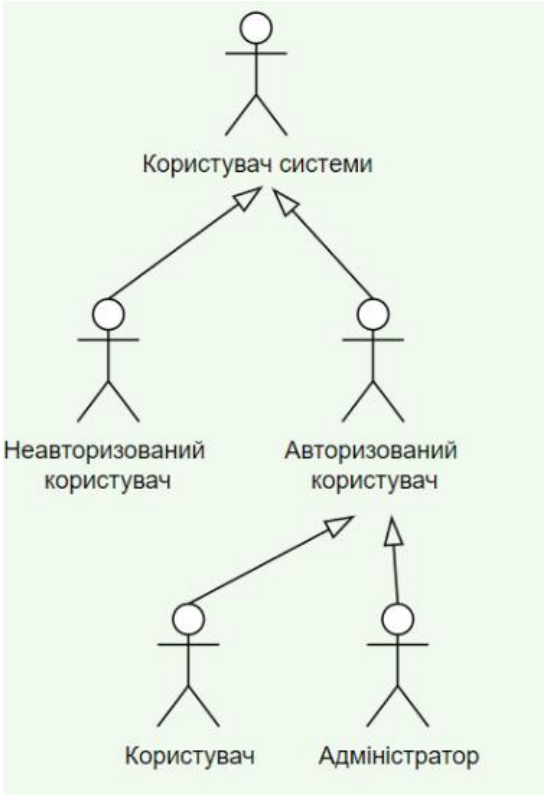


Рисунок 10 - Приклад графічного зображення відносини узагальнення між акторами

Відношення включення

Відношення включення між двома варіантами використання вказує, що деяка задана поведінка для одного варіанта використання включається як складеного компонента в послідовність поведінки іншого варіанта використання. Дане відношення є спрямованим бінарним відношенням у тому розумінні, що пари екземплярів варіантів використання завжди впорядкована відносно включення.

Семантика цього відношення визначається в такий спосіб. Коли екземпляр першого варіанта використання в процесі свого виконання досягає крапки включення в послідовність поведінки екземпляра другого варіанта використання, екземпляр першого варіанта використання виконує послідовність дій, що визначає поведінка екземпляра другого варіанта використання, після чого продовжує виконання дій своєї поведінки. При цьому передбачається, що навіть якщо екземпляр першого варіанта використання може мати екземплярів, що трохи включаються в себе, інших варіантів, виконувані ними дії повинні закінчитися до деякого моменту, після чого повинне бути продовжене виконання перерваних дій

екземпляра першого варіанта використання відповідно до заданого для нього поведінкою.

Один варіант використання може бути включений у трохи інших варіантів, а також містити в собі інші варіанти. варіант, що включається, використання може бути незалежним від базового варіанта в тому розумінні, що він надає останньому деяке инкапсулированное поведінка, деталі реалізації якого сховані від останнього й можуть бути легко перерозподілені між декількома варіантами, що включаються, використання. Більше того, базовий варіант може залежати тільки від результатів виконання, що включається в нього поведінки, але не від структури варіантів, що включаються в нього.

Відношення включення, спрямоване від варіанта використання A до варіанта використання B, указує, що кожний екземпляр варіанта A містить у собі функціональні властивості, задані для варіанта B. Ці властивості спеціалізують поведінка відповідного варіанта A на даній діаграмі. Графічно дане відношення позначається пунктирною лінією зі стрілкою (варіант відносини залежності), спрямованої від базового варіанта використання до, що включається. При цьому дана лінія зі стрілкою позначається ключовим словом "include" ("включає"), як показано на рис. 11.

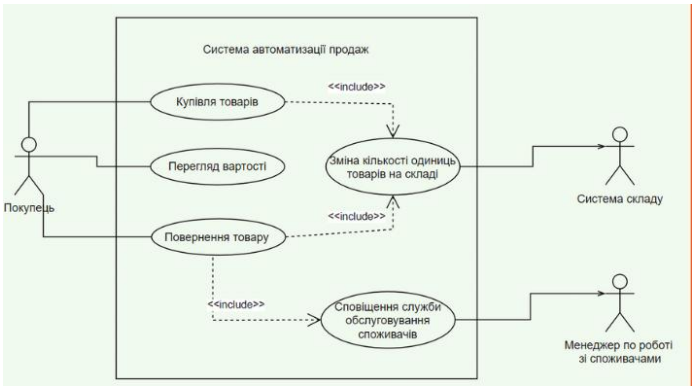


Рисунок 11 - Приклад графічного зображення відносини включення між варіантами використання

Приклад побудови діаграми варіантів використання

Секретар розміщує на сервері меню обідніх страв на тиждень. Співробітники повинні мати можливість ознайомитися з меню і зробити замовлення, вибравши страви на кожен день наступного тижня. Офіс-менеджер повинен мати можливість сформувати рахунок і оплатити його. Система повинна

бути написана на ASP.NET. Таке ось нехитрий інтернет-додаток для автоматизації замовлень обідів в офіс.

Прецедент	Дієва особа
Розташувати меню	Секретар
Ознайомитись з меню	Співробітник, секретар, офіс-менеджер
Зробити замовлення	Співробітник, секретар, офіс-менеджер
Сформувати рахунок	Офіс-менеджер
Оплатити рахунок	Офіс-менеджер

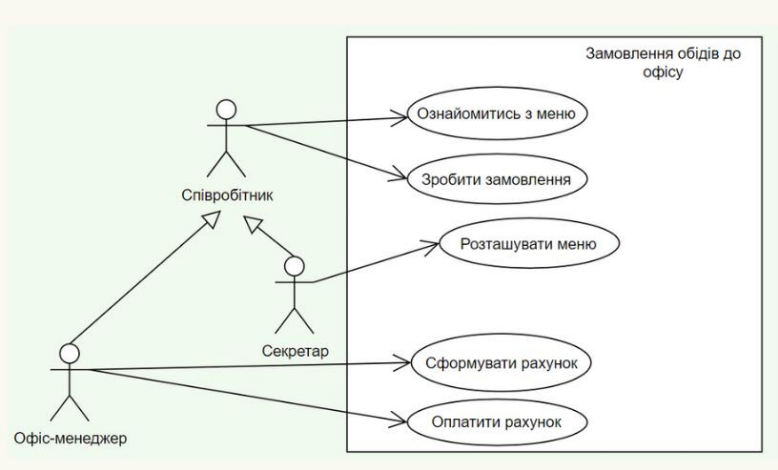


Рисунок 12 – Приклад діаграми прецедентів

Тема 2.1.4 Класи. Загальна характеристика

Лекція №6

(4 години)

Мета: Ознайомитись з поняттям клас. Розглянути поняття атрибуту та операції класу, а також розібрати форми запису атрибутів і операцій. Розглянути можливі види відносин між класами, такі як: залежність, асоціація, агрегація, композиція та узагальнення

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Загальна характеристика класів;
2. Терміни і поняття:
    a. Ім’я;
    b. Атрибут;
    c. Операція.
3. Відносини між класами:
    a. Відношення залежності;
    b. Відношення асоціації;
    c. Відношення агрегації;
    d. Відношення композиції
    e. Відношення узагальнення;

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичним матеріалом теми 2.1.4.
2. Вивчити основні поняття лекції.
3. Відповісти на контрольні питання.

Контрольні питання:

1. Що таке клас?
2. Дайте визначення поняттю інтерфейс.
3. В чому різниця між простим та складеним ім’ям класу?
4. Що таке атрибут класу?
5. Що таке квантор видимості?
6. Які значення може приймати кратність атрибута?
7. Що таке тип атрибуту?
8. Що таке вихідне значення?
9. Що таке операція класу?
10. Як записуються операції?
11. Як записується параметр операції?
12. Які види відносин можуть існувати між класами?

13. Що таке залежність?

14. Що таке стереотип залежності?

15. Що таке бінарна асоціація?

16. Що таке Тернарна асоціація?

17. Що таке агрегація? В чому різниця агрегації від композиції?

18. Як графічно позначається композиція?

Поняття об'єкта й класу тісно зв'язані. Проте існує важлива відмінність між цими поняттями. Клас - це абстракція істотних характеристик об'єкта.

Загальна характеристика класів

Клас - опис безлічі об'єктів, які розділяють однакові властивості, операції, відносини й семантику (зміст). Будь-який об'єкт - просто екземпляр класу.

Як показано на рис. 1, розрізняють внутрішнє представлення класу (реалізацію) і зовнішнє представлення класу (інтерфейс).

Інтерфейс повідомляє можливості (послуги) класу, але приховує його структуру й поведінку. Іншими словами, інтерфейс демонструє зовнішньому миру абстракцію класу, його зовнішній вигляд. Інтерфейс в основному складається з оголошень усіх операцій, застосовних до екземплярів класу. Він може також включати оголошення типів, змінних, констант і виключень, необхідних для повноти даної абстракції.

Рисунок 1 – Структурне представлення класу

Інтерфейс може бути розділений на 3 частині:

1) публічну (public), оголошення якої доступні всім клієнтам;

2) захищену (protected), оголошення якої доступні тільки самому класу, його підкласам і друзям;

3) приватну (private), оголошення якої доступні тільки самому класу і його друзям.

Другом класу називають клас, який має доступ до всіх частин цього класу (публічної, захищеної й приватної).

Реалізація класу описує секрети поведінки класу. Вона включає реалізації всіх операцій, певних в інтерфейсі класу.

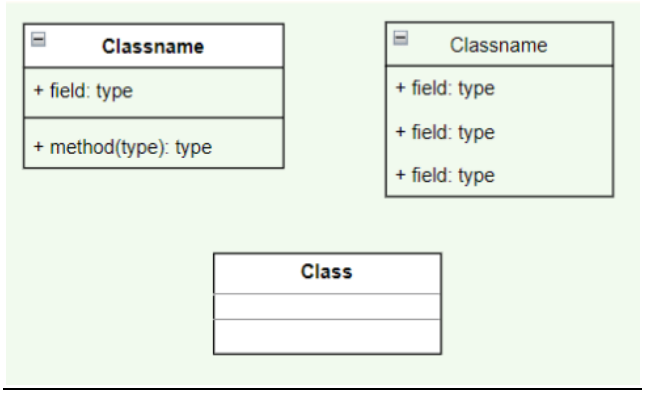


Рисунок 2 – Графічне представлення класу на діаграмі

Обов'язковим елементів позначення класу є його ім'я. На початкових етапах розробки діаграми окремі класи можуть позначатися простим прямокутником із вказівкою тільки імені відповідного класу. У міру проробляння окремих компонентів діаграми опису класів доповнюються атрибутами і операціями.

Передбачається, що остаточний варіант діаграми містить найбільш повний опис класів, які складаються із трьох розділів або секцій. Іноді в позначеннях класів використовується додатковий четвертий розділ, у якому приводиться семантична інформація довідкового характеру або явно вказуються виняткові ситуації.

Навіть якщо секція атрибутів і операцій є порожній, у позначенні класу вона виділяється горизонтальною лінією, щоб відразу відрізнити клас від інших елементів мови UML. Приклади графічного зображення класів на діаграмі класів наведені на рис. 3. У першому випадку для класу "Прямокутник" (рис. 3) зазначені тільки його атрибути - крапки на координатній площині, які визначають його розташування. Для класу "Вікно" (3) зазначені тільки його операції, секція атрибутів залишена порожній. Для класу "Рахунок" (рис 3) додатково зображена четверта секція, у якій зазначено виключення - відмова від обробки простроченої кредитної картки.

Рисунок 3 – Приклади графічного представлення класів на діаграмі

Терміни і поняття

У кожного класу повинне бути *ім'я*, що відрізняє його від інших класів. Ім'я класу - це текстовий рядок. Взяте саме по собі, воно називається *простим іменем*; до *складеного імені* попереду додане ім'я пакета, куди входить клас. Ім'я класу в об'ємлющому пакеті повинне бути унікальним. При графічному зображенні класу показується тільки його ім'я.

У другій зверху секції прямокутника класу записуються його *атрибути* (attributes) або *властивості*. У мові UML прийнята певна стандартизація запису атрибутів класу, яка підкоряється деяким синтаксичним правилам. Кожному атрибуту класу відповідає окремий рядок тексту, який складається із квантора видимості атрибута, імені атрибута, його кратності, типу значень атрибута й, можливо, його вихідного значення:

<квантор видимості><ім'я атрибута>[кратність]:

<тип атрибута> = <вихідне значення>{ рядок-властивість}

Квантор видимості може ухвалювати одне із трьох можливих значень і, відповідно, відображається за допомогою спеціальних символів:

• Символ "+" позначає атрибут з областю видимості типу *загальнодоступний* (public). Атрибут із цією областю видимості доступний або видний з будь-якого іншого класу пакета, у якому визначена діаграма.

• Символ "#" позначає атрибут з областю видимості типу *захищений* (protected). Атрибут із цією областю видимості недоступний або невидний для всіх класів, за винятком підкласів даного класу.

• Символ "-" позначає атрибут з областю видимості типу *закритий* (private). Атрибут із цією областю видимості недоступний або невидний для всіх класів без винятку.

Квантор видимості може бути опущений. У цьому випадку його відсутність проста означає, що видимість атрибута не вказується. Ця ситуація відрізняється від прийнятих за замовчуванням угод у традиційних мовах програмування, коли відсутність квантора видимості трактується як public або private. Однак замість умовних графічних позначень можна записувати відповідне ключове слово: public, protected, private.

Ім'я атрибута являє собою рядок тексту, який використовується в якості ідентифікатора відповідного атрибута й тому повинна бути унікальної в межах даного класу. Ім'я атрибута є єдиним обов'язковим елементом синтаксичного позначення атрибута.

Кратність атрибута характеризує загальна кількість конкретних атрибутів даного типу, що входять до складу окремого класу. У загальному випадку кратність записується у формі рядка тексту у квадратних дужках після імені відповідного атрибута:

[нижня_границя1 .. верхня_границя1, нижня_границя2.. верхня_грашца2, ..., нижняя_границak .. верхня_границak],

де нижня_границя й верхня_границя є позитивними цілими числами, кожна пара яких служить для позначення окремого замкненого інтервалу цілих чисел, у якого нижня (верхня) границя дорівнює значенню нижня_границя (верхня_границя). У цілому дана умовна позначка кратності відповідає теоретико-множинному об'єднанню відповідних інтервалів. У якості верхньої_границі може використовуватися спеціальний символ "*", який означає довільне позитивне ціле число. Інакше кажучи, це означає необмежене зверху значення кратності відповідного атрибута.

Значення кратності з інтервалу випливають у монотонно зростаючому порядку без пропуску окремих чисел, що лежать між нижньої й верхньої границями. При цьому дотримуються наступного правила: відповідні нижні й верхні границі інтервалів включаються в значення кратності. Якщо в якості кратності вказується однина , то кратність атрибута ухвалюється рівної даному

числу. Якщо ж вказується єдиний знак "*", то це означає, що кратність атрибута може бути довільним позитивним цілим числом або нулем.

Як приклад розглянемо наступні варіанти завдання кратності атрибутів.

[0..1] означає, що кратність атрибута може ухвалювати значення Про або 1. При цьому 0 означає відсутність значення для даного атрибута.

[0..*] означає, що кратність атрибута може ухвалювати будь-яке позитивне ціле значення більше або рівне 0. Ця кратність може бути записана коротше у вигляді простого символу - [*].

[1.:*] означає, що кратність атрибута може ухвалювати будь-яке позитивне ціле значення більше або рівне 1.

[1..5] означає, що кратність атрибута може ухвалювати будь-яке значення із чисел: 1, 2, 3, 4, 5.

[1..3,5,7] означає, що кратність атрибута може ухвалювати будь-яке значення із чисел: 1, 2, 3, 5, 7.

[1..3,7.. 10] означає, що кратність атрибута може ухвалювати будь-яке значення із чисел: 1, 2, 3, 7, 8, 9, 10.

[1..3,7..*] означає, що кратність атрибута може ухвалювати будь-яке значення із чисел: 1, 2, 3, а також будь-яке позитивне ціле значення більше або рівне 7.

Якщо кратність атрибута не зазначена, то за замовчуванням ухвалюється її значення рівне 1..1, тобто в точності 1.

Тип атрибута являє собою вираження, семантика якого визначається мовою специфікації відповідної до моделі. У нотації UML тип атрибута іноді визначається залежно від мови програмування, яка передбачається використовувати для реалізації даної моделі. У найпростішому випадку тип атрибута вказується рядком тексту, що має осмислене значення в межах пакета або моделі, до яких ставиться розглянутий клас.

Можна привести наступні приклади завдання імен і типів атрибутів класів:

• колір: Color - тут колір є іменем атрибута, Color - іменем типу даного атрибута. Зазначений запис може визначати традиційно використовувану Rgb-Модель (червоний, зелений, синій) для вистави кольору. У цьому випадку ім'я типу

Color саме й характеризує семантичну конструкцію, яка застосовується в більшості мов програмування для вистави кольору.

• ім'я_співробітника [1..2] : String - тут ім'я_співробітника є іменем атрибута, який служить для вистави інформації про імені, а можливо, і по батькові конкретного співробітника. Тип атрибута String (Рядок) саме й указує на той факт, що окреме значення імені являє собою рядок тексту з одного або двох слів (наприклад, "Кирило" або "Дмитро Іванович"). Оскільки в багатьох мовах програмування існує тип даних String, використання відповідного англомовного терміна не викликає непорозуміння в більшості програмістів. Однак, хоча в мові UML усі терміни даються в англомовній виставі, використання як тип атрибута Рядок у даній ситуації не виключається й визначається тільки міркуваннями зручності.

• видимість:Boolean - тут видимість є ім'я абстрактного атрибута (курсив тут не випадковий), який може характеризувати наявність візуальної вистави відповідного класу на екрані монітора. У цьому випадку тип Boolean означає, що можливими значеннями даного атрибута є одне із двох логічних значень: істина (true) або неправда (false). При цьому значення істина може відповідати наявності графічного зображення на екрані монітора, а значення неправда - його відсутності, про що додатково вказується в пояснювальному тексті. Оскільки кратність атрибута видимість не зазначена, вона ухвалює значення 1 за замовчуванням. У цій ситуації англомовне ім'я типу атрибута цілком виправдане наявністю відповідного базового типу в мовах програмування. Абстрактний характер даного атрибута позначається курсивним текстом у записі даного атрибута.

• форма:Багатокутник - тут ім'я атрибута форма може характеризувати такий клас, який є геометричною фігурою на площині. У цьому випадку тип атрибута Багатокутник указує на той факт, що окрема геометрична фігура може мати форму трикутника, прямокутника, ромба, п'ятикутника й будь-якого іншого багатокутника, але не окружності або еліпса. Цілком очевидно, що в даній ситуації використання відповідного англомовного терміна чи навряд доцільно, оскільки тип Багатокутник не є базовим для мов програмування.

Вихідне значення служить для завдання деякого початкового значення для відповідного атрибута в момент створення окремого екземпляра класу. Тут

необхідно дотримуватися правила приналежності значення типу конкретного атрибута. Якщо вихідне значення не зазначене, то значення відповідного атрибута не визначене на момент створення нового екземпляра класу. З іншого боку, конструктор відповідного об'єкта може перевизначати вихідне значення в процесі виконання програми, якщо в цьому виникає необхідність.

У якості прикладів вихідних значень атрибутів можна привести наступні доповнені вище варіанти завдання атрибутів:

• колір:Color = (255, 0, 0) - в Rgb- Моделі кольору це відповідає чистому червоному кольору в якості вихідного значення для даного атрибута.

• ім'я_співробітника[1..2]:String = Іван Іванович - можливо, це нетиповий випадок, який, скоріше, відповідає ситуації ім'я_керівника[2]:81пп§ = Іван Іванович.

• видимість:Boolean = істина - може відповідати ситуації, коли в момент створення екземпляра класу створюється видиме на екрані монітора вікно, відповідне до даного об'єкта.

• форма:Багатокутник = прямокутник - чи навряд вимагає коментарів, оскільки тут мова йде про геометричну форму створюваного об'єкта.

У третій зверху секції прямокутника записуються *операції або методи класу*. Операція (operation) являє собою деякий сервіс, що надає кожний екземпляр класу на певному вимогу. Сукупність операцій характеризує функціональний аспект поведінки класу. Запис операцій класу в мові UML також стандартизована й підкоряється певним синтаксичним правилам. При цьому кожної операції класу відповідає окремий рядок, який складається із квантора видимості операції, імені операції, вираження типу, що вертається операцією значення й, можливо, рядок-властивість даної операції:

<квантор видимості><ім'я операції>(список параметрів):

<вираження типу значення, що вертається, >{ рядок-властивість}

Квантор видимості, як і у випадку атрибутів класу, може ухвалювати одне із трьох можливих значень і, відповідно, відображається за допомогою спеціального символу. Символ "+" позначає операцію з областю видимості типу загальнодоступний (public). Символ "#" позначає операцію з областю видимості

типу захищений (protected). І, нарешті, символ "-" використовується для позначення операції з областю видимості типу закритий (private).

Квантор видимості для операції може бути опущений.

Ім'я операції являє собою рядок тексту, який використовується в якості ідентифікатора відповідної до операції й тому повинна бути унікальної в межах даного класу. Ім'я атрибута є єдиним обов'язковим елементом синтаксичного позначення операції.

Список параметрів є переліком розділених комами формальних параметрів, кожний з яких може бути представлений у наступному виді:

<вид параметра><ім'я параметра>:<вираження типу>=<значення параметра за замовчуванням>.

Тут *вид параметра* - є одне із ключових слів in, out або inout зі значенням in за замовчуванням, у випадку якщо вид параметра не вказується. *Ім'я параметра* є ідентифікатор відповідного формального параметра. *Вираження типу* є залежною від конкретної мови програмування специфікацією типу значення, що вертається, для відповідного формального параметра. Нарешті, *значення за замовчуванням* у загальному випадку являє собою вираження для значення формального параметра, синтаксис якого залежить від конкретної мови програмування й підкоряється прийнятим у ньому обмеженням.

Вираження типу значення, що вертається, також є залежною від мови реалізації специфікацією типу або типів значень параметрів, які вертаються об'єктом після виконання відповідної операції.

Рядок-Властивість служить для вказівки значень властивостей, які можуть бути застосовані до даного елемента. Рядок-Властивість не є обов'язкової, вона може отсутствовать, якщо ніякі властивості не специфіковані.

У якості прикладів записи операцій можна привести наступні позначення окремих операцій:

• +створити() - може позначати абстрактну операцію по створенню окремого об'єкта класу, яка є загальнодоступною й не містить формальних параметрів. Ця операція не повертає ніякого значення після свого виконання.

• +намалювати(форма: Багатокутник = прямокутник, колір_заливання: Color = (О, О, 255)) - може позначати операцію по зображенню на екрані монітора

прямокутної області синього кольору, якщо не вказуються інші значення як аргументи даної операції.

• видати_повідомлення():{"Помилка розподілу на нуль"} - зміст даної операції не вимагає пояснення, оскільки втримується в рядку-властивості операції. Дане повідомлення може з'явитися на екрані монітора у випадку спроби розподілу деякого числа на нуль, що неприпустимо.

Крім внутрішнього обладнання або структури класів на відповідній діаграмі вказуються різні відносини між класами. При цьому сукупність типів таких відносин фіксована в мові UML і визначена семантикою цих типів відносин. Базовими відносинами або зв'язками в мові UML є:

• Відношення залежності (dependency relationship)

• Відношення асоціації (association relationship)

• Відношення узагальнення (generalization relationship)

• Відношення реалізації (realization relationship)

Кожне із цих відносин має власна графічна вистава на діаграмі, яке відбиває взаємозв'язку між об'єктами відповідних класів.

Лекція №7

Відношення залежності

Відношення залежності в загальному випадку вказує деяке семантичне відношення між двома елементами моделі або двома множинами таких елементів, яке не є відношенням асоціації, узагальнення або реалізації. Воно стосується тільки самих елементів моделі й не вимагає безлічі окремих прикладів для пояснення свого змісту. Відношення залежності використовується в такій ситуації, коли деяка зміна одного елемента моделі може зажадати зміни іншого залежного від нього елемента моделі.

Відношення залежності графічно зображується пунктирною лінією між відповідними елементами зі стрілкою на одному з її кінців ("->" або "<-"). На діаграмі класів дане відношення зв'язує окремі класи між собою, при цьому стрілка спрямована від класу-клієнта залежності до незалежного класу або класу-джерелу (рис. 4). На даному малюнку зображено два класи: Клас_А і Кяасс_Б, при цьому Клас_Б є джерелом деякої залежності, а Клас_А - клієнтом цієї залежності.

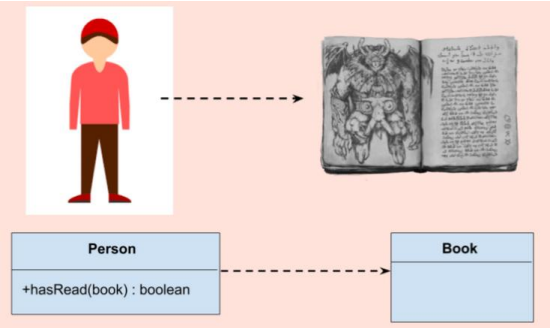


Рисунок 4– Графічне зображення відношення залежності на діаграмі

У якості класу-клієнта й класу-джерела залежності можуть виступати цілі безлічі елементів моделі. У цьому випадку одна лінія зі стрілкою, що виходить від джерела залежності, розщеплюється в деякій крапці на кілька окремих ліній, кожна з яких має окрему стрілку для класу-клієнта. Наприклад, якщо функціонування Класу_А залежить від особливостей реалізації Класу_Б, то дана залежність може бути зображена в такий спосіб (рис. 5).

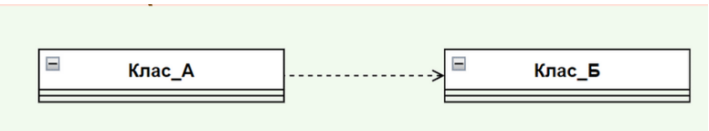


Рисунок 12.2 - Графічне представлення залежності між класом-клієнтом (Клас_С) і класами-джерелами (Клас_А и Клас_Б)

Стрілка може позначатися необов'язковим, але стандартним ключовим словом у лапках і необов'язковим індивідуальним іменем. Для відношення залежності визначені ключові слова, які позначають деякі спеціальні види залежностей. Ці ключові слова *(стереотипи)* записуються в лапках поруч зі стрілкою, яка відповідає даній залежності. Приклади стереотипів для відношення залежності представлені нижче:

• "access" - служить для позначення доступності відкритих атрибутів і операцій класу-джерела для класів-клієнтів;

• "bind" - клас-клієнт може використовувати деякий шаблон для своєї наступної параметризації;

• "derive" - атрибути класу-клієнта можуть бути обчислені по атрибутах класу-джерела;

• "import" - відкриті атрибути й операції класу-джерела стають частиною класу-клієнта, як якби вони були оголошені безпосередньо в ньому;

• та інші

Відношення асоціації

Відношення асоціації відповідає наявності деякого відношення між класами. Дане відношення позначається суцільною лінією з додатковими спеціальними символами, які характеризують окремі властивості конкретної асоціації. У якості додаткових спеціальних символів можуть використовуватися ім'я асоціації, а також імена й кратність класів-ролей асоціації. Ім'я асоціації є необов'язковим елементом її позначення.

Найбільш простий випадок даного відношення - *бінарна асоціація*. Вона зв'язує в точності два класи й, як виключення, може зв'язувати клас із самим собою. Для бінарної асоціації на діаграмі може бути зазначений порядок проходження класів з використанням трикутника у формі стрілки поруч із іменем даної асоціації. Напрямок цієї стрілки вказує на порядок класів, один з яких є першим (з боку трикутника), а іншої - другим (з боку вершини трикутника). Відсутність даної стрілки поруч із іменем асоціації означає, що порядок проходження класів у розглянутому відношенні не визначений.

У якості простого прикладу відносини бінарної асоціації розглянемо відношення між двома класами - класом "Компанія" і класом "Співробітник" (рис. 6). Вони зв'язані між собою бінарною асоціацією Робота, ім'я якої зазначено на малюнку поруч із лінією асоціації. Для даного відношення визначений порядок проходження класів, першим з яких є клас "Співробітник", а другим - клас "Компанія".

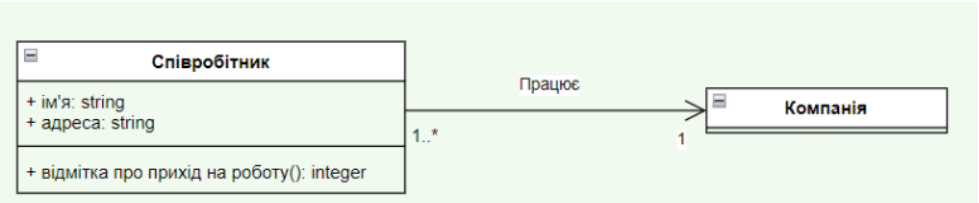


Рисунок 6 - Графічне зображення відносини бінарної асоціації між класами

Тернарна асоціація й асоціації більш високої арності в загальному випадку називаються *N- Арной асоціацією* (читається - "эн арная асоціація"). Така асоціація зв'язує деяким відношенням 3 і більш класів, при цьому один клас може брати участь в асоціації більш ніж один раз. Клас асоціації має певну роль у відповідному відношенні, що може бути явно зазначене на діаграмі. Кожний екземпляр N- Арной асоціації являє собою N- Арний кортеж значень об'єктів з відповідних класів.

Бінарна асоціація є часним випадком N- Арной асоціації, коли значення N=2, і має своє власне позначення.

N-Арная асоціація графічно позначається ромбом, від якого ведуть лінії до символів класів даної асоціації. У цьому випадку ромб з'єднується із символами відповідних класів суцільними лініями. Звичайно лінії проводяться від вершин ромба або від середини його сторін. Ім'я N- Арної асоціації записується поруч із ромбом відповідної асоціації.

Порядок класів в N- Арнії асоціації на діаграмі не фіксується. Деякий клас може бути приєднаний до ромба пунктирною лінією. Це означає, що даний клас забезпечує підтримку властивостей відповідної N-Арної асоціації, а сама N-Арна асоціація має атрибути, операції й/або асоціації. Інакше кажучи, така асоціація, у свою чергу, є класом з відповідним позначенням у вигляді прямокутника і є самостійним елементом мови UML - асоціацією-класом (Association Class). N-Арная асоціація не може містити символ агрегації ні для якої зі своїх ролей.

Як приклад конкретної тернарной асоціації розглянемо відношення між трьома класами: "Футбольна команда", "Рік" і "Гра". Дана асоціація вказує на наявність відносини між цими трьома класами, яке може представляти інформацію про ігри футбольних команд у національному чемпіонаті протягом декількох останніх років (рис. 7).

Як уже згадувалося, окремий клас асоціації має власну роль у відношенні. Ця роль може бути зображена графічно на діаграмі класів.

Рисунок 7 - Графічне зображення тернарной асоціації між трьома класами

Одним з додаткових позначень є *ім'я ролі* окремого класу, що входить в асоціацію. Ім'я ролі являє собою рядок тексту поруч із кінцем асоціації для відповідного класу. Вона вказує специфічну роль, яку відіграє клас. Ім'я ролі не є обов'язковим елементом позначень і може отсутствовать на діаграмі.

Кратність класу позначається у вигляді інтервалу цілих чисел, аналогічно кратності атрибутів і операцій класів. Інтервал записується поруч із кінцем асоціації й для N-Арной асоціації означає потенційне число окремих екземплярів.

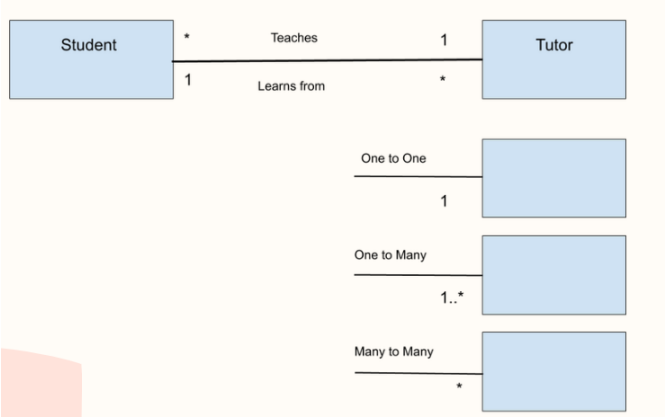


Рисунок 8 – приклади кратності

Відношення агрегації

Відношення агрегації має місце між декількома класами в тому випадку, якщо один із класів являє собою деяку сутність, що включає в себе як складені частини інші сутності.

Дане відношення має фундаментальне значення для опису структури складних систем, оскільки застосовується для вистави системних взаємозв'язків типу "частина-ціле". Розкриваючи внутрішню структуру системи, відношення агрегації показує, з яких компонентів складається система і як вони зв'язані між собою. З погляду моделі окремі частини системи можуть виступати як у вигляді елементів, так і у вигляді підсистем, які, у свою чергу, теж можуть утворювати складені компоненти або підсистеми. Це відношення по своїй суті описує декомпозицію або розбивку складної системи на більш прості складові частини, які також можуть бути піддані декомпозиції, якщо в цьому виникне необхідність надалі .

Очевидно, що розглянуте в такому аспекті розподіл системи на складові частини являє собою деяку ієрархію її компонентів, однак дана ієрархія принципово відрізняється від ієрархії, породжуваної відношенням узагальнення. Відмінність полягає в тому, що частини системи ніяк не зобов'язано успадковувати її властивості й поведінка, оскільки є цілком самостійними сутностями. Більше того, частини цілого мають свої власні атрибути й операціями, які суттєво відрізняються від атрибутів і операцій цілого.

Як приклад відносини агрегації розглянемо взаємозв'язок типу " частина-ціле", яка має місце між сутністю "Вантажний автомобіль" і такими компонентами, як "Двигун", "Шасі", "Кабіна", "Кузов". Не претендуючи на точну відповідність термінології даної предметної області, неважко уявити собі, що вантажний автомобіль складається із двигуна, шасі, кабіни й кузова. Саме це відношення між класом "Вантажний_автомобіль" і класами "Двигун", "Шасі", "Кабіна", "Кузов" описує відношення агрегації.

Графічно відношення агрегації зображується суцільною лінією, один з кінців якої являє собою незафарбований усередині ромб. Цей ромб указує на той із класів, який являє собою "ціле". Інші класи є його "частинами".

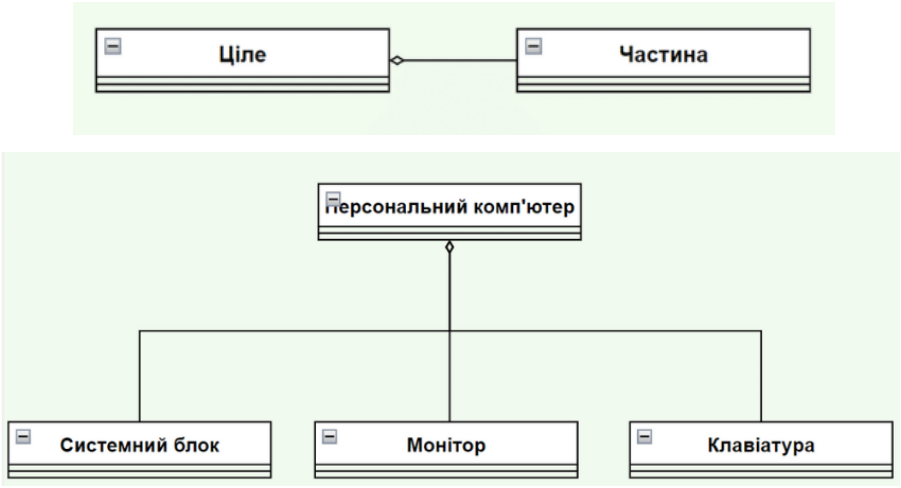


Рисунок 9 – Графічне представлення та приклад агрегації

Відношення композиції

Відношення композиції, як уже згадувалося раніше, є частним випадком відносини агрегації. Це відношення служить для виділення спеціальної форми відносини "частина-ціле", при якій складові частини в деякому змісті перебувають усередині цілого. Специфіка взаємозв'язку між ними полягає в тому, що частини не можуть виступати у відриві від цілого, тобто зі знищенням цілого знищуються й усі його складові частини.

Приклад - вікно інтерфейсу програми, яке може складатися з рядка заголовка, кнопок керування розміром, смуг прокручування, головного меню, робочої області й рядка стану. Неважко зрозуміти, що подібне вікно являє собою клас, а його компоненти є як класами, так і атрибутами або властивостями вікна. Остання обставина досить характерна для відношення композиції, оскільки відбиває різні способи вистави даного відношення.

Графічно відношення композиції зображується суцільною лінією, один з кінців якої являє собою зафарбований усередині ромб. Цей ромб указує на той із класів, який являє собою клас-композицію або "ціле". Інші класи є його "частинами".

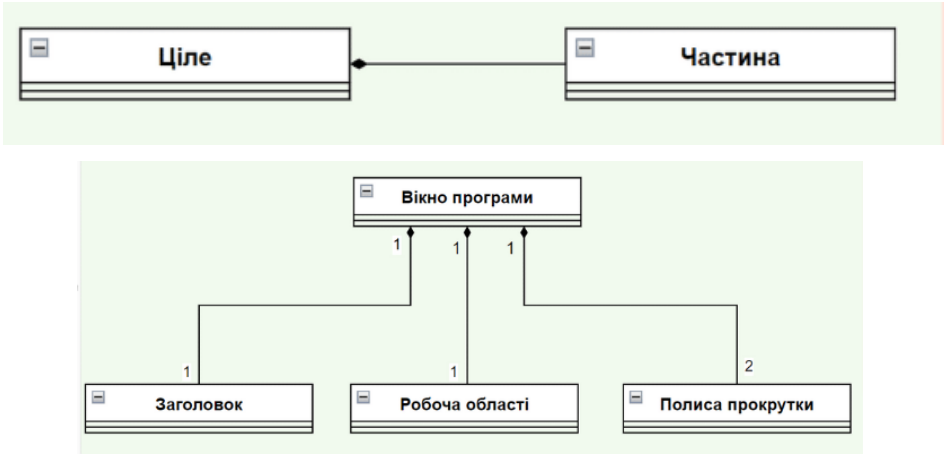


Рисунок 10 - Графічне представлення та приклад композиції

Відношення узагальнення

Відношення узагальнення є звичайним таксономічним відношенням між більш загальним елементом (батьком або предком) і більш приватним або спеціальним елементом (дочірнім або нащадком).

Дане відношення описує ієрархічну будову класів і спадкування їх властивостей і поведінки. При цьому передбачається, що клас-нащадок має всі властивості й поведінку класу-предка, а також має свої власні властивості й поведінку, які відсутні в класі-предку. На діаграмах відношення узагальнення позначається суцільною лінією із трикутною стрілкою на одному з кінців. Стрілка вказує на більш загальний клас (клас-предок або суперклас), а її відсутність - на більш спеціальний клас (клас-нащадок або підклас).

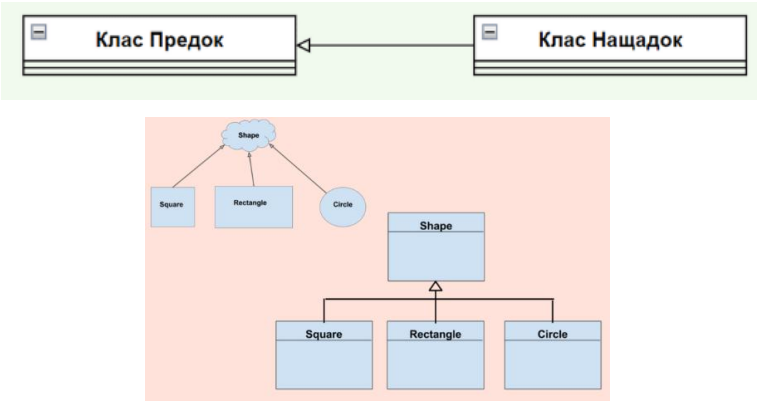


Рисунок 11 - Графічне представлення та приклад узагальнення

Тема 2.1.5 Вершини та відносини в діаграмах класів

Лекція №8

(2 години)

Мета: Розглянути основний засіб представлення статичних моделей – діаграму класів. Ознайомитись з організацією властивостей і операцій, з поняттям множинності. Розглянути відносини в діаграмі класів. Розглянути приклади діаграм.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Статичні моделі;
2. Організація властивостей і операцій;
3. Множинність;
4. Відносини в діаграмах класів;
5. Дерева спадкування;
6. Приклади діаграм класів.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:
   1. Ознайомитись з теоретичним матеріалом теми 2.1.5.
   2. Вивчити основні поняття лекції.
   3. Відповісти на контрольні питання.

Контрольні питання:

1. Поясните призначення статичних моделей об'єктно-орієнтованих програмних систем.

2. Що є основним засобом для представлення статичних моделей?
3. Як використовуються статичні моделі?
4. Які секції входять у графічне позначення класу?
5. Які секції класу можна не показувати?
6. Які є різновиди області дії властивості (операції)?
7. Поясните загальний синтаксис вистави властивості.
8. Які рівні видимості ви знаєте? Їхній зміст?
9. Поясните загальний синтаксис вистави операції.
10. Який вид має форма вистави параметра операції?
11. Що означають три крапки в списку властивостей (операцій)?
12. Як організує угруповання властивостей (операцій)?
13. Як обмежити кількість екземплярів класу?
14. Перелічите відомі вам "прикраси" відносини асоціації.
15. Що таке абстрактний клас (операція) і як він (вона) відображається?
16. Як заборонити поліморфізм операції?
17. Як позначити кореневий клас?

Статичні моделі забезпечують представлення структури систем у термінах базових будівельних блоків і відносин між ними. "Статичність" цих моделей полягає в тому, що тут не показується динаміка змін системи в часі. Разом з тим слід розуміти, що ці моделі несуть у собі не тільки структурні описи, але й опису операцій, що реалізують задане поведінку системи. Основним засобом для представлення статичних моделей є діаграми класів. Вершини діаграм класів навантажені класами, а дуги (ребра) - відносинами між ними. Діаграми використовуються:

- у ході аналізу - для вказівки ролей і обов'язків сутностей, які забезпечують поведінка системи;
- у ході проектування - для фіксації структури класів, які формують системну архітектуру.

Позначення класу показане на рис. 1.

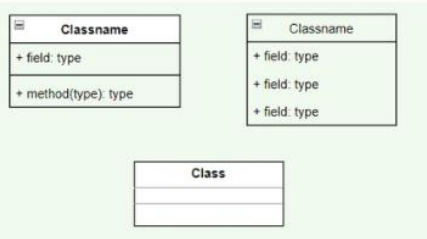


Рисунок 1 – Позначення класу

Ім'я класу вказується завжди, властивості й операції - вибірково.

Організація властивостей і операцій

Відомо, що піктограма класу включає три секції (для імені, для властивостей і для операцій). Пустота секції не означає, що в класі відсутні властивості або операції, просто в цей момент вони не показуються. Можна явно визначити наявність у класі більшої кількості властивостей або атрибутів. Для цього наприкінці показаного списку проставляються три крапки. Як показано на рис. 2, у довгих списках властивостей і операцій дозволяється угруповання - кожна група починається зі свого стереотипу.

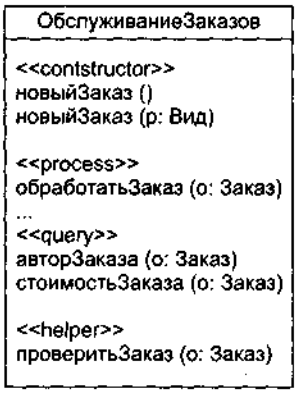


Рисунок 2 – Стереотипи для характеристик класу

Множинність

Іноді буває необхідно обмежити кількість екземплярів класу:

- задати нуль екземплярів (у цьому випадку клас перетворюється в утиліту, яка пропонує свої властивості й операції);
- задати один екземпляр (клас-singleton);
- задати конкретну кількість екземплярів;
- не обмежувати кількість екземплярів (це випадок, передбачуваний за замовчуванням).

Кількість екземплярів класу називається його множинністю. Вираження множинності записується в правому верхньому куті значка класу. Наприклад, як показано на рис. 3, *Контроллер углов* - це клас-singleton, а для класу *Датчик угла* дозволено три екземпляри.

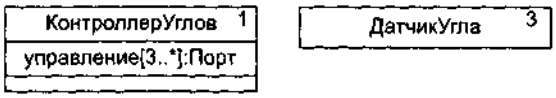


Рисунок 3– Множинність

Відносини в діаграмах класів

Відносини, використовувані в діаграмах класів, показані на рис. 4.

Рисунок 4 – Відношення на діаграмах класів

Асоціації відображають структурні відносини між екземплярами класів, тобто з'єднання між об'єктами.

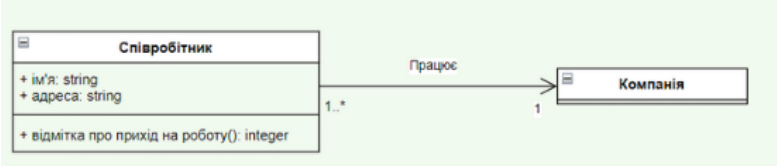


Рисунок 5 – Приклади асоціації

Агрегація показує відношення по посиланню (в агрегат включені тільки покажчики на частині), композиція - відношення фізичного включення (в агрегат включені самі частини).

Залежність є відношенням використання між клієнтом (залежним елементом) і постачальником (незалежним елементом). Звичайно операції клієнта:

- викликають операції постачальника;
- мають сигнатури, у яких значення, що вертається, або аргументи належать класу постачальника.

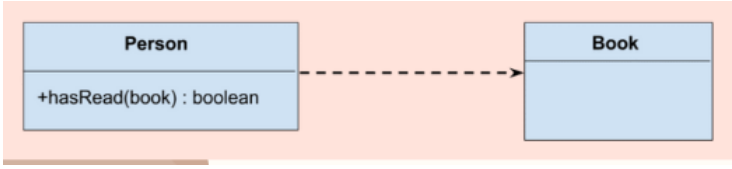


Рисунок 6 – Приклад залежності

Узагальнення - відношення між загальним предметом (суперкласом) і спеціалізованим різновидом цього предмета (підкласом). Підклас може мати одного батька (один суперклас) або кілька батьків (кілька суперкласів). У другому випадку говорять про множинне спадкування.

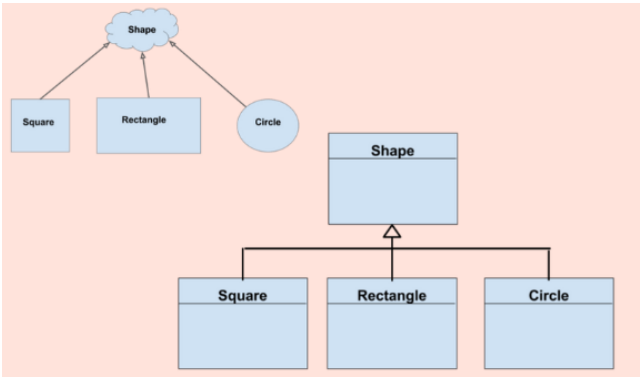


Рисунок 7 – Приклад множинного спадкування та ромбовидної решітки спадкування

Реалізація - семантичне відношення між класами, у якому клас-приймач виконує реалізацію операцій інтерфейсу класу-джерела. Наприклад, на мал. 8 показане, що клас Каталог повинен реалізувати інтерфейс Оброблювач каталогу, тобто Оброблювач каталогу розглядається як джерело, а Каталог - як приймач.

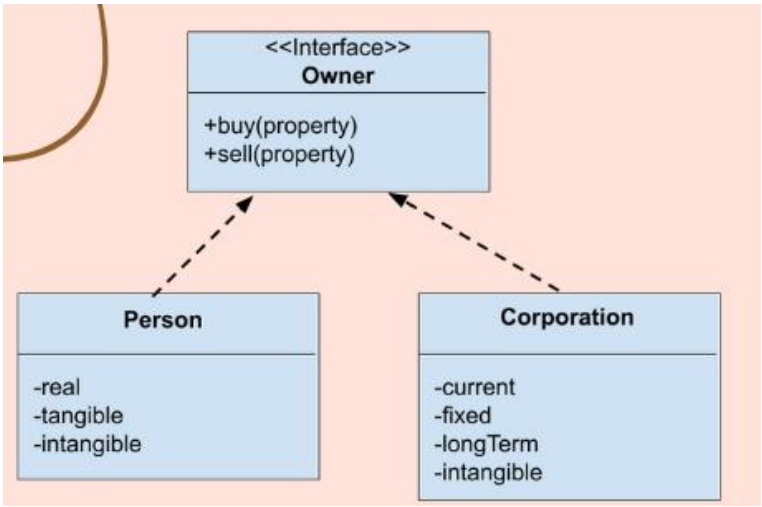


Рисунок 8 – Реалізація інтерфейсу

Інтерфейс Оброблювач каталогу дозволяє клієнтам взаємодіяти з об'єктами класу Каталог без знання тієї дисципліни доступу, яка тут реалізована.

Дерева спадкування

При використанні відносин узагальнення будується ієрархія класів. Деякі класи в цій ієрархії можуть бути абстрактними. Абстрактним називають клас, який не може мати екземплярів. Імена абстрактних класів записуються курсивом. Наприклад, на рис. 0 показані абстрактні класи *Ссавці, Собаки, Кішки*.

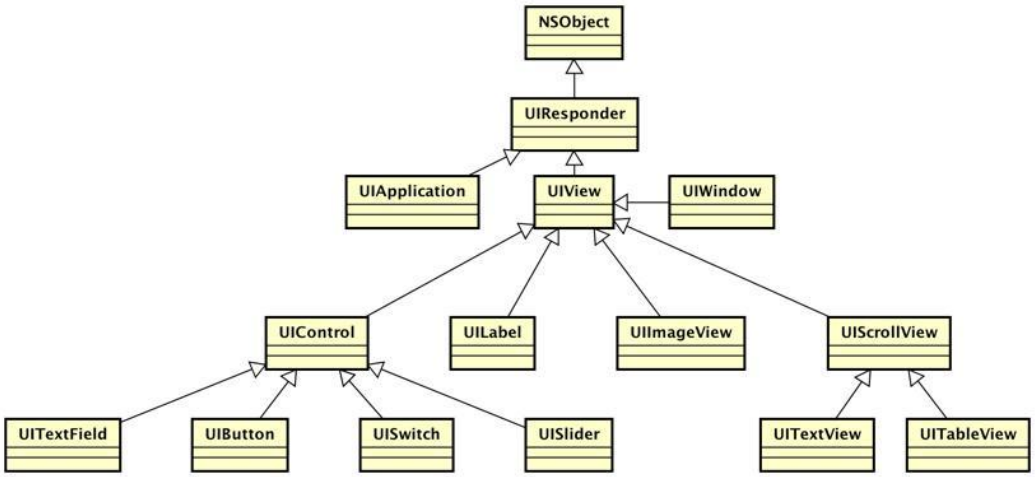


Рисунок 9 – Абстракція та поліморфізм

Крім того, тут є конкретні класи Охотничьи собаки, Сетер, кожний з яких може мати екземпляри.

Звичайно клас успадковує якісь характеристики класу-батька й передає свої характеристики класу-нащадкові. Іноді потрібно визначити кінцевий клас, який не може мати дітей. Такі класи позначаються теговою величиною (характеристикою) leaf, записуваної за іменем класу. Наприклад, на малюнку показаний кінцевий клас Сетер.

Іноді корисно відзначити кореневий клас, який не може мати батьків. Такий клас позначається теговою величиною (характеристикою) root, записуваної за іменем класу. Наприклад, на малюнку показаний кореневий клас Ссавці.

Аналогічні властивості мають і операції. Звичайно операція є поліморфної, це значить, що в різних крапках ієрархії можна визначати операції зі схожою сигнатурою. Такі операції з дочірніх класів перевизначають поведінка відповідних операцій з батьківських класів. При обробці повідомлення (у період виконання) проводиться поліморфний вибір однієї з операцій ієрархії відповідно до типу об'єкта. Наприклад, ОтображатьВнешнийВид () і ВлезатьНаДерево (дуб) - поліморфні операції. До того ж операція Ссавці::ОтображатьВнешнийВид ( ) є

абстрактної, тобто неповної й потребуючої для своєї реалізації нащадка. Ім'я абстрактної операції записується курсивом (як і ім'я класу). З іншого боку, Ссавці::УзнатьВес () - кінцева операція, що відзначається характеристикою leaf. Це значить, що операція не полиморфна й не може перекриватися.

Приклади діаграм класів

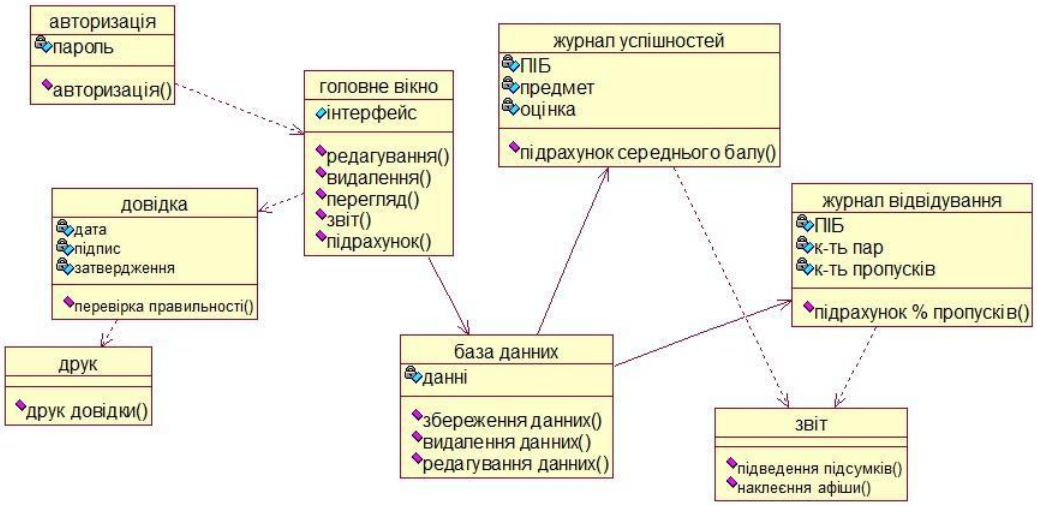


Рисунок 10 - Діаграма класів системи керування польотом

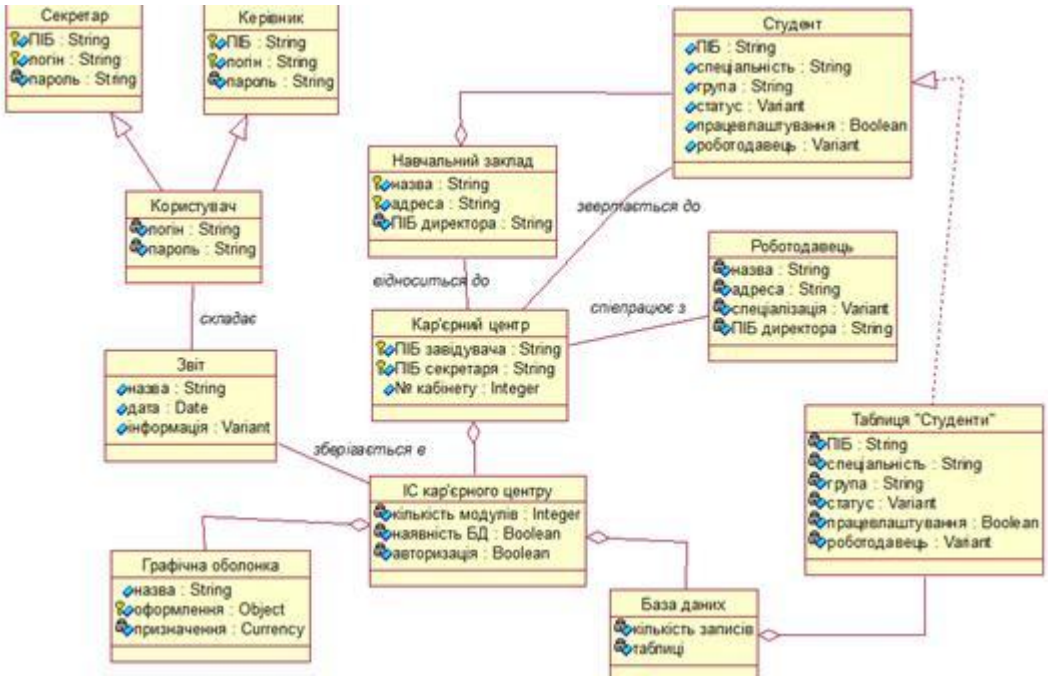


Рисунок 11 – Діаграма класів кар’єрного центру ЗВО

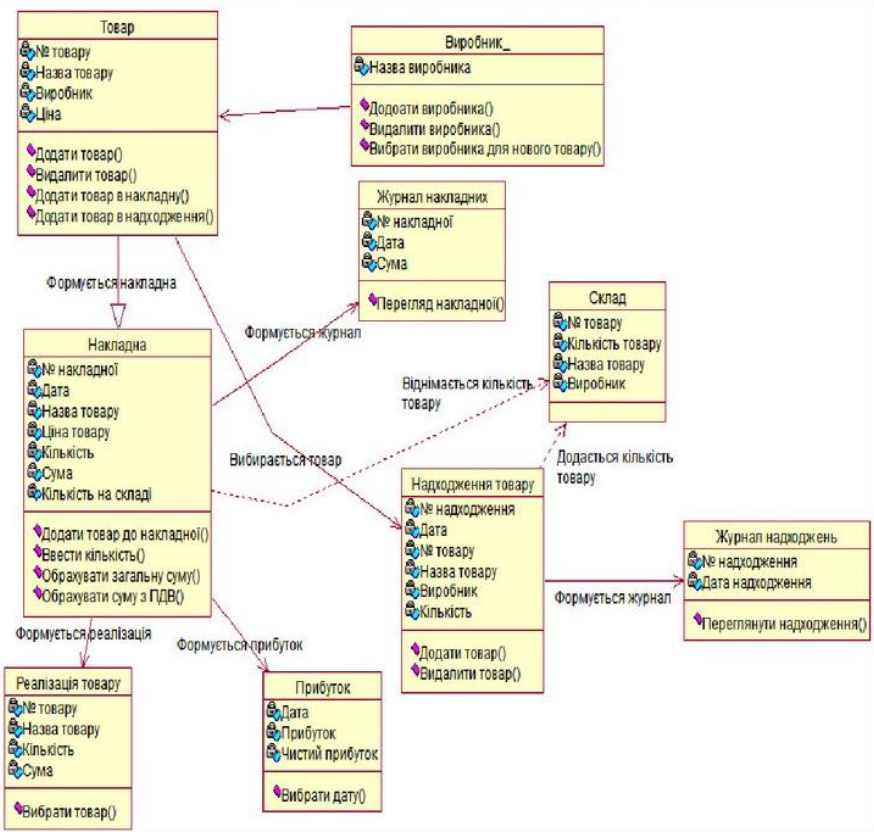


Рисунок 12 – Діаграма класів продажі товарів

Тема 2.1.6 Діаграми співпраці. Діаграми послідовностей

Лекція №9

(2 години)

Мета: Розглянути основні поняття діаграми співпраці. Ознайомитись з графічною нотацією діаграми. Розглянути основні поняття діаграми послідовностей. Ознайомитись з графічною нотацією діаграми.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Діаграми співпраці;
    a. Синтаксис імені об’єкта;
    b. Синтаксис властивості об’єкта;
    c. Повідомлення;
2. Діаграма послідовності;
    a. Об’єкти;
    b. Лінія життя об'єкта;
    c. Фокус керування;
    d. Повідомлення;
    e. Розгалуження потоку керування.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичним матеріалом теми 2.1.6.
2. Вивчити основні поняття лекції.
3. Відповісти на контрольні питання.

Контрольні питання:

1. Як представляється ім'я об'єкта в діаграмі співробітництва?
2. Поясните синтаксис вистави властивості в діаграмі співробітництва.
3. Які стереотипи видимості використовуються в діаграмі співробітництва? Поясните їхній зміст.
4. У якій формі записуються повідомлення в мові UML? Поясните зміст повідомлення.
5. У якому відношенні перебувають повідомлення й дії? Перелічите різновиду дій.
6. Чим відрізняється процедурний потік від асинхронного потоку повідомлень?
7. Як вказується повторення повідомлень?
8. Як показати розгалуження повідомлень?

9. Чим відрізняється процедурний потік від асинхронного потоку повідомлень?
10. Як вказується повторення повідомлень?
11. Як показати розгалуження повідомлень?
12. Що загального в діаграмі послідовності й діаграмі співробітництва? Чим вони відрізняються друг від друга?
13. Як відображається порядок передачі повідомлень у діаграмі послідовності?
14. Коли зручніше застосовувати діаграми послідовності?

Діаграми взаємодії призначені для моделювання динамічних аспектів системи. Діаграма взаємодії показує взаємодію, що включає набір об'єктів і їх відносин, а повідомлення, що також пересилаються між об'єктами. Існують два різновиди діаграми взаємодії - діаграма послідовності й діаграма співробітництва. Діаграма послідовності - це діаграма взаємодії, яка виділяє впорядкування повідомлень за часом. Діаграма співробітництва - це діаграма взаємодії, яка виділяє структурну організацію об'єктів, що посилають повідомлення, що й ухвалюють. Елементами діаграм взаємодії є учасники взаємодії - об'єкти, зв'язки, повідомлення.

Діаграми співробітництва

Діаграми співробітництва відображають взаємодія об'єктів у процесі функціонування системи. Такі діаграми моделюють сценарії поведінки системи. У російській літературі діаграми співробітництва часто називають діаграмами кооперації.

Позначення об'єкта показане на мал. .1.

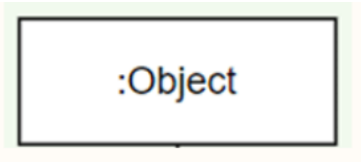


Рисунок 1 – Позначення об’єкта

Ім'я об'єкта підкреслюється й вказується завжди, властивості вказуються вибірково. Синтаксис представлення імені має вигляд:

ІмяОб’єкта : ІмяКласу

Приклади запису імені:

Адам : Люди Ім'я об'єкта й класу

: Користувач Тільки ім'я класу (анонімний об'єкт)

мойКомпьютер Тільки ім'я об'єкта (мається на увазі, що ім'я класу відомо)

агент : Об'єкт - сирота (мається на увазі, що ім'я класу невідомо)

Синтаксис представлення властивості має вигляд:

Ім'я : Тип = Значення

Приклади запису властивості:

номер:Телефон = "7350-420" Ім'я, тип, значення

активний = True Ім'я й значення

Об'єкти взаємодіють один з одним за допомогою *зв'язків* - каналів для передачі повідомлень. Зв'язок між парою об'єктів розглядається як екземпляр асоціації між їхніми класами. Іншими словами, зв'язок між двома об'єктами існує тільки тоді, коли є асоціація між їхніми класами. Неявно всі класи мають асоціацію самі із собою, отже, об'єкт може послати повідомлення самому собі.

Отже, зв'язок - це шлях для пересилання повідомлення. Шлях може бути постачений характеристикою видимості. Характеристика видимості проставляється як стандартний стереотип над далеким кінцем зв'язку. У мові передбачені наступні стандартні стереотипи видимості:

«global»	Об'єкт-Постачальник перебуває в глобальній області визначення
«local»	Об'єкт-Постачальник перебуває в локальній області визначення об'єкта-клієнта
«parameter»	Об'єкт-Постачальник є параметром операції об'єкта-клієнта
«self»	Той самий об'єкт є й клієнтом, і постачальником

Повідомлення - це специфікація передачі інформації між об'єктами чекаючи того, що буде забезпечена необхідна діяльність. Приймання повідомлення розглядається як подія.

Результатом обробки повідомлення звичайно є дія. У мові UML моделюються наступні різновиди дій:

Виклик	В об'єкті запускається операція
Повернення	Повернення значення в ухвалий об'єкт
Посилка(Send)	В об'єкт посилається сигнал
Створення	Створення об'єкта, виконується по стандартному повідомленню "create"

Знищення	Знищення об'єкта, виконується по стандартному повідомленню "destroy"

Для запису повідомлень у мові UML прийнятий наступний синтаксис:

ВозврВеличина := ИмяСообщения (Аргументи),

де ВозврВеличина задає величину, що вертається як результат обробки повідомлення.

Приклади запису повідомлень:

Коорд:=ТекущПоложение(самолетТ1)	Вызов операции, возврат значения
оповещение( )	Посылка сигнала
УстановитьМаршрут(х)	Вызов операции с действительным параметром
«create»	Стандартное сообщение для создания объекта

Коли об'єкт посилає повідомлення в інший об'єкт (делегуючи деяка дію одержувачеві), об'єкт-одержувач, у свою чергу, може послати повідомлення в третій об'єкт, і т.д. Так формується потік повідомлень - послідовність керування. Очевидно, що повідомлення в послідовності повинні бути пронумеровані. Номери записуються перед іменами повідомлень, напрямку повідомлень вказуються стрілками (розміщаються над лініями зв'язків).

Найбільш загальну форму керування задає процедурний або вкладений потік (потік синхронних повідомлень). Як показано на мал. 8.2, процедурний потік рисується стрілками із заповненими наконечниками.

Рисунок 2 - Потік синхронних повідомлень

Тут повідомлення 2.1 : Напій : = Виготовити(Смесь№3) визначене як перше повідомлення, вкладене в друге повідомлення 2 : Замовити(Смесь№3) послідовності, а повідомлення 2.2 : Принести(Напій) - як друге вкладене повідомлення. Усі повідомлення процедурної послідовності вважаються синхронними. Робота із синхронним повідомленням підкоряється наступному правилу: передавач чекає доти , поки одержувач не прийме й не обробить

повідомлення. У нашому прикладі це означає, що третє повідомлення буде послано тільки після обробки повідомлень 2.1 і 2.2. Відзначимо, що ступінь вкладеності повідомлень може бути кожний. Головне, щоб дотримувалося правило: послідовність повідомлень зовнішнього рівня відновляється тільки після завершення вкладеної послідовності.

Менш загальну форму керування задає асинхронний потік керування. Як показано на мал. 18.3, асинхронний потік рисується звичайними стрілками. Тут усі повідомлення вважаються асинхронними, при яких передавач не чекає реакції від одержувача повідомлення. Такий вид комунікації має семантику поштової скриньки - одержувач ухвалює повідомлення в міру готовності. Іншими словами, передавач і одержувач не синхронізують свою роботу, скоріше, один об'єкт "позбувається" від повідомлення для іншого об'єкта. У нашому прикладі повідомлення Подтверждениевызова визначене як друге повідомлення в послідовності.

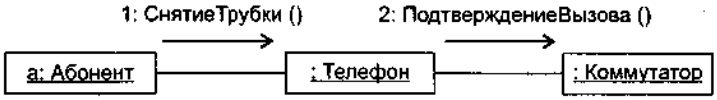


Рисунок 3 - Потік асинхронних повідомлень

Крім розглянутих лінійних потоків керування, можна моделювати й більш складні форми - ітерації й розгалуження.

Ітерація представляє повторювану послідовність повідомлень. Після номера повідомлення ітерації додається вираження

*[i := 1 .. n].

Воно означає, що повідомлення ітерації буде повторюватися задана кількість раз. Наприклад, чотириразове повторення першого повідомлення Рисоватьсторонупрямоугольника можна задати вираженням

1*[1 := 1 .. 4] : РисоватьСторонуПрямоугольника(i)

Для моделювання розгалуження після номера повідомлення додається вираження умови, наприклад: [x>0]. Повідомлення альтернативної галузей позначається таким же номером, але з іншою умовою: [x<=0]. Приклад ітераційного потоку, що й розгалужується, повідомлень наведений на рис. 8.4.

Тут перше повідомлення повторюється 4 рази, а в якості другого вибирається одне із двох повідомлень (залежно від значення змінної x). У підсумку екземпляр

рисователя намалює на екрані прямокутне вікно, а екземпляр співрозмовника виведе в нього відповідне повідомлення.

Таким чином, для формування діаграми співробітництва виконуються наступні дії:

1) відображаються об'єкти, які беруть участь у взаємодії;

2) рисуються зв'язки, що з'єднують ці об'єкти;

3) зв'язки позначаються повідомленнями, які посилають і одержують виділені об'єкти.

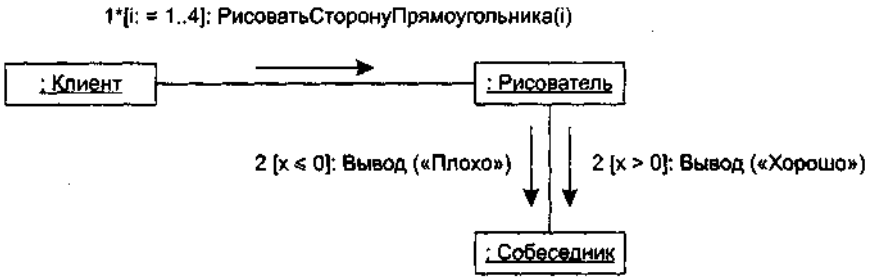


Рисунок 4 - Ітерація й розгалуження

У підсумку формується ясна візуальна вистава потоку керування (у контексті структурної організації об'єктів, що співробітничають).

Як приклад на рис. 5 наведена діаграма співробітництва системи керування польотом літального апарата.

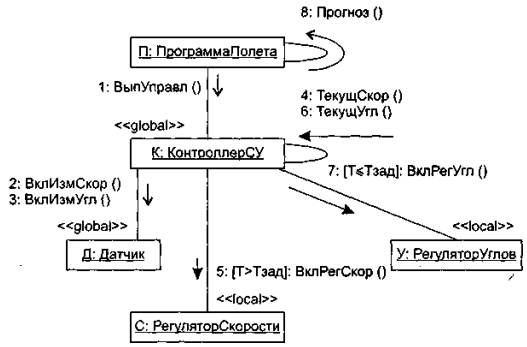


Рисунок 5 - Діаграма співробітництва системи керування польотом

На даній діаграмі представлено п'ять об'єктів, явно показані характеристики видимості всіх зв'язків системи. Потік керування в системі включає вісім повідомлень: чотири асинхронні й чотири синхронні повідомлення. Екземпляр Контролера СУ чекає приймання й обробки повідомлень:

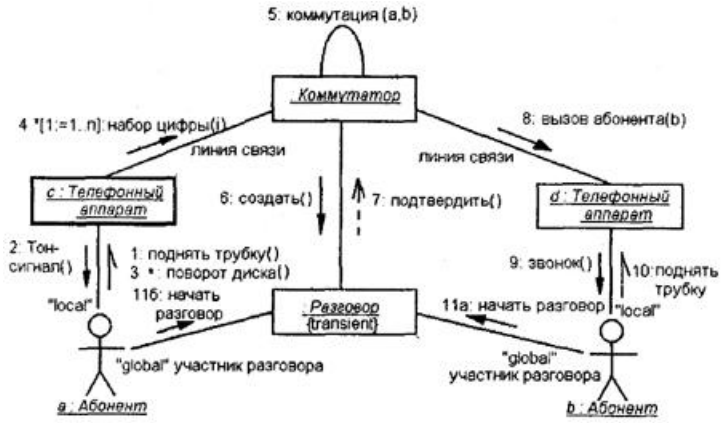
- ВКЛРЕГСКОР( );
- Вррегугл();
- Текущскор();

- Текущугл( ).

Порядок проходження повідомлень заданий їхніми номерами. Для п'ятого й сьомого повідомлень зазначені умови:

- включення Регулятора Швидкості відбувається, якщо відносний час польоту Т більше заданого періоду $T_{зад}$;
- включення Регулятора Кутів забезпечується, якщо відносний час польоту менше або дорівнює заданому періоду.

Приклад діаграми:



Діаграма послідовності - другий різновид діаграм взаємодії. Відбиваючи сценарій поведінки в системі, ця діаграма забезпечує більш наочна вистава порядку передачі повідомлень. Правда, вона не дозволяє показати такі деталі, які видні на діаграмі співробітництва (структурні характеристики об'єктів і зв'язків).

Об'єкти

На діаграмі послідовності зображуються винятково ті об'єкти, які безпосередньо беруть участь у взаємодії й не показуються можливі статичні асоціації з іншими об'єктами. Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об'єктів у часі. При цьому діаграма послідовності має як би два виміри. Одне - ліворуч праворуч у вигляді вертикальних ліній, кожна з яких зображує *лінію життя* окремого об'єкта, що брало участь у взаємодії. Графічно кожний об'єкт зображується прямокутником і розташовується у верхній частині своєї лінії життя (рис. 6). Усередині прямокутника записуються ім'я об'єкта й ім'я класу, розділені двокрапкою. При цьому весь запис підкреслюється, що є ознакою об'єкта, яка, як відомо, являє собою екземпляр класу.

Рисунок 6 - Різні графічні примітиви діаграми послідовності

Крайнім ліворуч на діаграмі зображується об'єкт, який є ініціатором взаємодії (об'єкт 1 на мал. 6). Праворуч зображується інший об'єкт, який безпосередньо взаємодіє з першим. Таким чином, усі об'єкти на діаграмі послідовності утворюють деякий порядок, обумовлений 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 посилають одними об'єктами іншим. Повідомлення зображуються у вигляді горизонтальних стрілок з іменем повідомлення й також утворюють порядок за часом свого виникнення. Інакше кажучи, повідомлення, розташовані на діаграмі послідовності вище, ініціюються раніше тих, які розташовані нижче. При цьому масштаб на осі часу не вказується, оскільки діаграма послідовності моделює лише тимчасову впорядкованість взаємодій типу " раніше-пізніше".

Лінія життя об'єкта

Лінія життя об'єкта (object lifeline) зображується пунктирною вертикальною лінією, асоційованої з єдиним об'єктом на діаграмі послідовності. Лінія життя служить для позначення періоду часу, протягом якого об'єкт існує в системі й, отже, 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 (об'єкти 1 і 2 на мал. 7).

Окремі об'єкти, виконавши свою роль у системі, можуть бути знищені (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 спеціальний символ у формі латинської букви "X" (об'єкт 3 на мал. 7). Нижче цього символу пунктирна лінія не зображується, оскільки відповідного об'єкта в системі вже ні, і цей об'єкт повинен бути виключений із усіх наступних взаємодій.

Зовсім не обов'язково створювати всі об'єкти в початковий момент часу. Окремі об'єкти в системі можуть створюватися в міру необхідності, суттєво заощаджуючи ресурси системи й підвищуючи її продуктивність. У цьому випадку прямокутник такого об'єкта зображується не у верхній частині діаграми послідовності, а в тій її частині, яка відповідає моменту створення об'єкта (об'єкт 6 на мал. 7). 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 й, можливо, з фокусом керування.

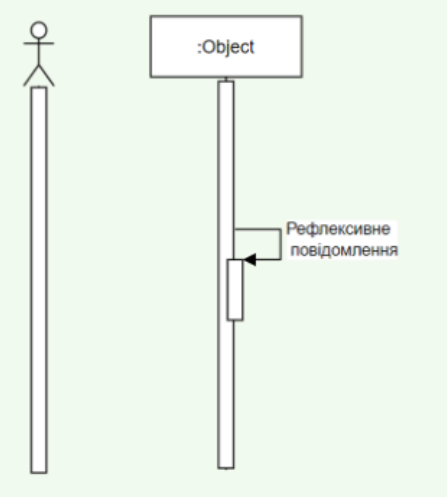


Рисунок 7 - Графічне зображення різних варіантів ліній життя й фокусів керування об'єктів

Фокус керування

У процесі функціонування об'єктно-орієнтованих систем одні об'єкти можуть перебувати в активному стані, безпосередньо виконуючи певні дії або в стані пасивного очікування повідомлень від інших об'єктів. Щоб явно виділити подібну активність об'єктів, у мові UML застосовується спеціальне поняття, що

одержала назва фокуса керування (focus of control). Фокус керування зображується у формі витягнутого вузького прямокутника, верхня сторона якого позначає початок одержання фокуса управління об'єкта (початок активності), а її нижня сторона - закінчення фокуса керування (закінчення активності). Цей прямокутник розташовується нижче позначення відповідного об'єкта й може заміняти його лінію життя, якщо на всьому її протязі він є активним.

З іншого боку, періоди активності об'єкта можуть чергуватися з періодами його пасивності або очікування. У цьому випадку в такого об'єкта є кілька фокусів керування. Важливо розуміти, що одержати фокус керування може тільки існуючий об'єкт, у якого в цей момент є лінія життя. Якщо ж деякий об'єкт був знищений, то знову виникнути в системі він уже не може. Замість нього лише може бути створений інший екземпляр цього ж класу, який, строго говорячи, буде іншим об'єктом.

В окремих випадках ініціатором взаємодії в системі може бути актор або зовнішній користувач. У цьому випадку актор зображується на діаграмі послідовності найпершим об'єктом ліворуч зі своїм фокусом керування. Найчастіше актор і його фокус керування будуть існувати в системі постійно, відзначаючи характерну для користувача активність в ініціюванні взаємодій із системою. При цьому сам актор може мати власне ім'я або залишатися анонімним.

Іноді деякий об'єкт може ініціювати рекурсивна взаємодія із самим собою. Мова йде про те, що наявність у багатьох мовах програмування спеціальних засобів побудови рекурсивних процедур вимагає візуалізації відповідних понять у формі графічних примітивів. На діаграмі послідовності рекурсія позначається невеликим прямокутником, приєднаним до правої сторони фокуса керування того об'єкта, для якого зображується ця рекурсивна взаємодія.

Рисунок 8 - Графічне зображення актора, рекурсії й рефлексивного повідомлення на діаграмі послідовності

Повідомлення

Як було відзначено вище, ціль взаємодії в контексті мови UML полягає в тому, щоб специфицировать комунікацію між безліччю взаємодіючих об'єктів. Кожна взаємодія описується сукупністю повідомлень, якими що брати участь у ньому об'єкти обмінюються між собою. У цьому змісті повідомлення (message) являє собою закінчений фрагмент інформації, який відправляється одним об'єктом іншому. При цьому приймання повідомлення ініціює виконання певних дій, спрямованих на розв'язок окремого завдання тем об'єктом, якому це повідомлення відправлене.

Таким чином, повідомлення не тільки передають деяку інформацію, але й вимагають або припускають від об'єкта, що ухвалює, виконання очікуваних дій. Повідомлення можуть ініціювати виконання операцій об'єктом відповідного класу, а параметри цих операцій передаються разом з повідомленням. На діаграмі послідовності всі повідомлення впорядковані за часом свого виникнення в моделируемой системі.

У такому контексті кожне повідомлення має напрямок від об'єкта, який ініціює й відправляє повідомлення, до об'єкта, який його одержує. Іноді відправника повідомлення називають клієнтом, а одержувача - сервером. При цьому повідомлення від клієнта має форму запиту деякого сервісу, а реакція сервера на запит після одержання повідомлення може бути пов'язана з виконанням певних дій або передачі клієнтові необхідної інформації теж у формі повідомлення.

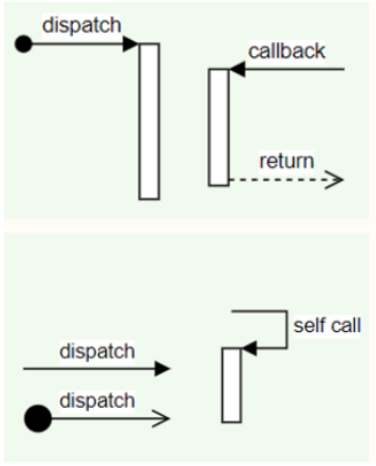


Рисунок 9 - Графічне зображення різних видів повідомлень між об'єктами на діаграмі послідовності

У мові UML можуть зустрічатися кілька різновидів повідомлень, кожне з яких має своє графічне зображення.

Перший різновид повідомлення є найпоширенішою й використовується для виклику процедур, виконання операцій або позначення окремих вкладених потоків керування. Початок цієї стрілки завжди стикається з фокусом керування або лінією життя того об'єкта- клієнта, який ініціює це повідомлення. Кінець стрілки стикається з лінією життя того об'єкта, який ухвалює це повідомлення й виконує у відповідь певні дії. При цьому, що ухвалює об'єкт найчастіше одержує й фокус керування, стаючи активним.

Другий різновид повідомлення використовується для позначення простого (не вкладеного) потоку керування. Кожна така стрілка вказує на прогрес одного кроку потоку. При цьому відповідні повідомлення звичайно є асинхронними, тобто можуть виникати в довільні моменти часу. Передача такого повідомлення звичайно супроводжується одержанням фокуса керування об'єктом, його, що прийняли.

Третій різновид явно позначає асинхронне повідомлення між двома об'єктами в деякій процедурній послідовності. Прикладом такого повідомлення може служити переривання операції при виникненні виняткової ситуації. У цьому випадку інформація про таку ситуацію передається зухвалому об'єкту для продовження процесу подальшої взаємодії.

Нарешті, останній різновид повідомлення використовується для повернення з виклику процедури. Прикладом може служити просте повідомлення про завершення деяких обчислень без надання результату розрахунків об'єкту-клієнтові. У процедурних потоках керування ця стрілка може бути опущена, оскільки її наявність неявно передбачається наприкінці активізації об'єкта. У той же час уважається, що кожний виклик процедури має свою пару - повернення виклику. Для непроцедурних потоків керування, включаючи паралельні й асинхронні повідомлення, стрільця повернення повинна вказуватися явно .

Звичайно повідомлення зображуються горизонтальними стрілками, що з'єднують лінії життя або фокуси керування двох об'єктів на діаграмі послідовності. При цьому неявно передбачається, що час передачі повідомлення досить мало в порівнянні із процесами виконання дій об'єктами. Уважається також, що за час передачі повідомлення з відповідними об'єктами не може відбутися

ніяких подій. Інакше кажучи, стану об'єктів залишаються без зміни. Якщо ж це припущення не може бути визнане слушним, то стрілка повідомлення зображується під деяким нахилом, так щоб кінець стрілки розташовувався нижче її початку.

В окремих випадках об'єкт може посилати повідомлення самому собі, ініціюючи так звані рефлексивні повідомлення. Такі повідомлення зображуються прямокутником зі стрілкою, початок і кінець якої збігаються. Подібні ситуації виникають, наприклад, при обробці натискань на клавіші клавіатури при введенні тексту в документ, що редагується, при наборі цифр номера телефону абонента.

Таким чином, у мові UML кожне повідомлення асоціюється з деякою дією, яка повинна є виконане його об'єктом, що прийняв. При цьому дія може мати деякі аргументи або параметри, залежно від конкретних значень яких може бути отриманий різний результат. Відповідні параметри буде мати й зухвала ця дія повідомлення. Більше того, значення параметрів окремих повідомлень можуть містити умовні вираження, утворюючи розгалуження або альтернативні шляхи основного потоку керування.

Розгалуження потоку керування

Для зображення розгалуження рисуються дві або більш стрілки, що виходять із однієї крапки фокуса управління об'єкта. При цьому відповідні умови повинні бути явно зазначені поруч із кожної зі стрілок у формі сторожової умови. Як неважко представити, якщо умова записана у формі булевского вираження, то розгалуження буде містити тільки дві галузі. У кожному разі умови повинні взаємно виключати одночасну передачу альтернативних повідомлень.

За допомогою розгалуження можна зобразити й більш складну логіку взаємодії об'єктів між собою. Якщо умов більш двох, то для кожного з них необхідно передбачити ситуацію єдиного виконання. Розглянутий приклад ставиться до моделювання взаємодії програмної системи обслуговування клієнтів у банку. На цьому прикладі діаграми послідовності об'єкт 1 передає керування одному із трьох інших об'єктів.

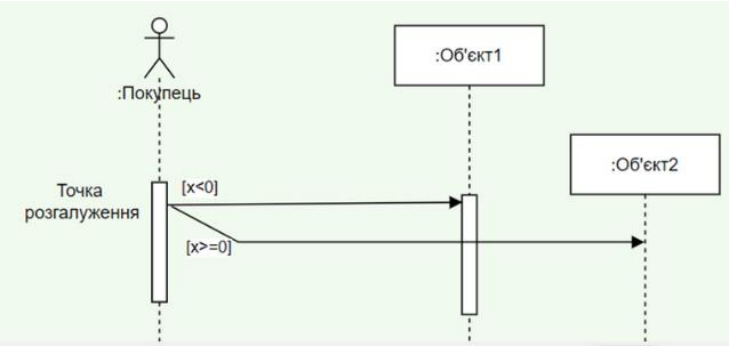
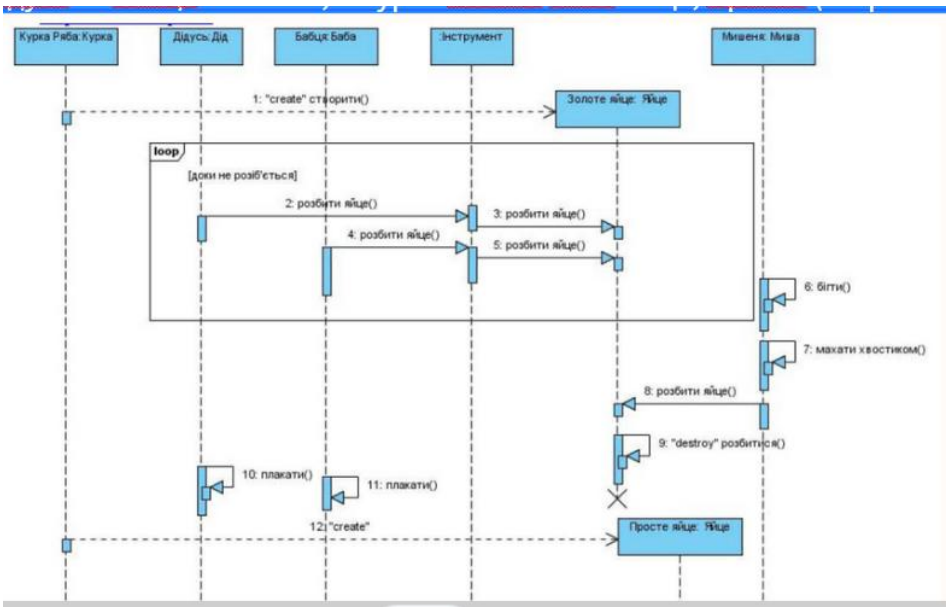


Рисунок 10 - Графічне зображення бінарного розгалуження потоку керування на діаграмі послідовності

Приклад діаграми:

Курка Ряба знесла (стереотип <<create>>) Золоте яйце. Дідусь та Бабця намагались розбити яйце (циклічний фрагмент), проте їхні спроби не були вдалими. Бішло Мишеня (рекурсивне повідомлення Бігти()) , махало хвостиком (рекусивне повідомлення Михати хвостиком()) та розбило яйце (стериотип <<destroy>>). Дідусь та Бабця плакали, а Курка знесла нове Яйце, Просте (стереотип <<create>>).



Тема 2.1.7 Моделювання поведінки програмної системи. Діаграма схем станів

Лекція №10

(4 години)

Мета: Розглянути основні поняття діаграми станів. Ознайомитись з графічною нотацією діаграми.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Моделювання поведінки програмної системи;
2. Діаграми схем станів;
   a. Автомати;
   b. Стан;
   c. Список внутрішніх дій;
   d. Початковий стан;
   e. Кінцевий стан;
   f. Перехід;
   g. Подія;
   h. Сторожова умова.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:
   1. Ознайомитись з теоретичним матеріалом теми 2.1.7.
   2. Вивчити основні поняття лекції.
   3. Відповісти на контрольні питання.

Контрольні питання:

1. Що використовують для моделювання поведінки системи?
2. Дайте визначення поняттю автомат?
3. За допомогою чого можна відобразити автомат?
4. Що відображає діаграма схем станів?

5. На виконанні яких обов'язкових умов заснований формалізм діаграм станів?
6. Що таке стан?
7. Дайте визначення поняттю список внутрішніх подій.
8. Які види псевдо-станів Вам відомі?
9. Що таке подія?
10. Для чого використовується сторожова умова?

Динамічні моделі забезпечують вистава поведінки систем. "Динамізм" цих моделей полягає в тому, що в них відбивається зміна станів у процесі роботи системи (залежно від часу). Засобу мови UML для створення динамічних моделей численні й різноманітні. Ці засоби орієнтовані не тільки на властиво програмні системи, але й на відображення вимог замовника до поведінки таких систем.

Моделювання поведінки програмної системи

Для моделювання поведінки системи використовують:

- автомати;
- взаємодії.

Автомат (State machine) описує поведінка в термінах послідовності станів, через які проходить об'єкт протягом свого життя. *Взаємодія* (Interaction) описує поведінку в термінах обміну повідомленнями між об'єктами.

Таким чином, автомат задає поведінку системи як цільної, єдиної сутності; моделює життєвий цикл єдиного об'єкта. У силу цього автоматний підхід зручно застосовувати для формалізації динаміки окремого важкого для розуміння блоку системи.

Взаємодії визначають поведінку системи у вигляді комунікацій між його частинами (об'єктами), представляючи систему як співтовариство спільно працюючих об'єктів. Саме тому взаємодії вважають основним апаратом для фіксації повної динаміки системи.

Автомати відображають за допомогою:

- діаграм схем станів;
- діаграм діяльності.

Взаємодії відображають за допомогою:

- діаграм співробітництва (кооперації);
- діаграм послідовності.

Діаграми схем станів

Діаграма схем станів - одна з п'яти діаграм UML, що моделюють динаміку систем. Діаграма схем станів відображає кінцевий автомат, виділяючи потік керування, що випливає від стану до стану. *Кінцевий автомат* - поведінка, яка визначає послідовність станів у ході існування об'єкта. Ця послідовність розглядається як відповідь на події й включає реакції на ці події.

Діаграма схем станів показує:

1) набір станів системи;

2) події, які викликають перехід з одного стану в інше;

3) дії, які відбуваються в результаті зміни стану.

Головне призначення цієї діаграми - описати можливі послідовності станів і переходів, які в сукупності характеризують поведінка елемента моделі протягом його життєвого циклу. Діаграма станів представляє динамічну поведінку сутностей, на основі специфікації їх реакції на сприйняття деяких конкретних подій. Системи, які реагують на зовнішні дії від інших систем або від користувачів, іноді називають реактивними. Якщо такі дії ініціюються в довільні випадкові моменти часу, то говорять про асинхронну поведінку моделі.

Хоча діаграми станів найчастіше використовуються для опису поведінки окремих екземплярів класів (об'єктів), але вони також можуть бути застосовані для специфікації функціональності інших компонентів моделей, таких як варіанти використання, актори, підсистеми, операції й методи.

Діаграма станів по суті є графом спеціального виду, який представляє деякий автомат. Поняття автомата в контексті UML має досить специфічну семантику, засновану на теорії автоматів. Вершинами цього графа є стани й деякі інші типи елементів автомата (псевдо-стани), які зображуються відповідними графічними символами. Дуги графа служать для позначення переходів зі стану в стан. Діаграми станів можуть бути вкладені друг у друга, створюючи вкладені діаграми більш детального представлення окремих елементів моделі.

Автомати

Автомат (state machine) у мові UML являє собою деякий формалізм для моделювання поведінки елементів моделі й системи в цілому. У метамоделі UML автомат є пакетом, у якому визначена безліч понять, необхідних для представлення поведінки сутності у вигляді дискретного простору з кінцевим числом станів і переходів. З іншого боку, автомат описує поведінку окремого об'єкта у формі послідовності станів, які охоплюють усі етапи його життєвого циклу, починаючи від створення об'єкта й закінчуючи його знищенням. Кожна діаграма станів представляє деякий автомат.

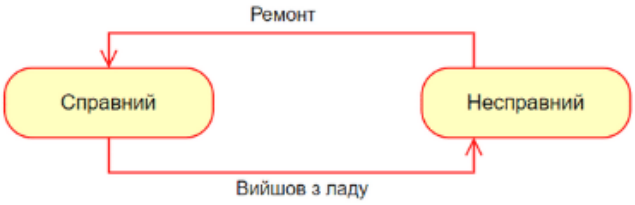


Рисунок 1 - Найпростіший приклад діаграми станів для технічного обладнання типу комп'ютер

Основними поняттями, що входять у формалізм автомата, є стан і перехід. Головна відмінність між ними полягає в тому, що тривалість знаходження системи в окремому стані суттєво перевищує час, який затрачається на перехід з одного стану в інше. Передбачається, що в межі час переходу з одного стану в інше дорівнює нулю (якщо додатково нічого не сказане). Інакше кажучи, перехід об'єкта зі стану в стан відбувається миттєво.

Поведінка моделюється як послідовне переміщення по графові станів від вершини до вершини по єднальних їхніх дугах з обліком їх орієнтації. Для графа станів системи можна ввести в розгляд спеціальні властивості.

Одним з таких властивостей є виділення із усієї сукупності станів двох спеціальні: *початкового й кінцевого*. Хоча ні в графові станів, ні на діаграмі станів час знаходження системи в тому або іншому стані явно не враховується, передбачається, що послідовність зміни станів упорядкована в часі. Інакше кажучи, кожний наступний стан завжди наступає пізніше попереднього йому стану.

Формалізм автоматів допускає вкладення одних автоматів в інші для уточнення внутрішньої структури окремих більш загальних станів (макростанів). У цьому випадку вкладені автомати одержали назву *під-автоматів*. Під-автомати можуть використовуватися для внутрішньої специфікації процедур і функцій, що

утворюють поведінку вихідного об'єкта. Наприклад, стан несправності технічного обладнання (мал. 1) може бути деталізований на окремі під-стани, кожне з яких може характеризувати несправність окремих підсистем, що входять до складу даного обладнання.

У мові UML поняття автомата доповнене спеціальною семантикою вхідних у відповідний пакет елементів.

Формалізм звичайного автомата заснований на виконанні наступних обов'язкових умов:

1. Автомат не запам'ятовує історію переміщення зі стану в стан. З погляду моделируемої поведінки визначальним є сам факт знаходження об'єкта в тому або іншому стані, але ніяк не послідовність станів, у результаті якої об'єкт перейшов у поточний стан. Інакше кажучи, автомат "забуває" усі стани, які передували поточному в цей момент часу.
2. У кожний момент часу автомат може перебувати в одному й тільки в одному зі своїх станів. Це означає, що формалізм автомата призначений для моделювання послідовної поведінки, коли об'єкт протягом свого життєвого циклу послідовно проходить через усі свої стани. При цьому автомат може перебувати в окремому стані як завгодно довго, якщо не відбувається ніяких подій.
3. Хоча процес зміни станів автомата відбувається в часі, явно концепція часу не входить у формалізм автомата. Це означає, що тривалість знаходження автомата в тому або іншому стані, а також час досягнення того або іншого стану ніяк не специфікуються. Інакше кажучи, час на діаграмі станів присутній в неявному виді, хоча для окремих подій може бути зазначений інтервал часу й у явному виді.
4. Кількість станів автомата повинне бути обов'язково кінцевою (у мові UML розглядаються тільки кінцеві автомати), і всі вони повинні бути специфіковані явно. При цьому окремі псевдо-стани можуть не мати специфікацій (початковий і кінцевий стану). У цьому випадку їх призначення й семантика повністю визначаються з контексту моделі й розглянутої діаграми станів.
5. Граф автомата не повинен містити ізольованих станів і переходів. Ця умова означає, що для кожного зі станів, крім початкового, повинне бути визначений

попередній стан. Кожний перехід повинен обов'язково з'єднувати два стани автомата. Допускається перехід зі стану в себе, такий перехід ще називають "петлею".

6. Автомат не повинен містити конфліктуючих переходів, тобто таких переходів з того самого стану, коли об'єкт одночасно може перейти у два й більш наступні стани (крім випадку паралельних під-автоматів). У мові UML виключення конфліктів можливо на основі введення так званих *сторожових умов*.

Таким чином, правила поведінки об'єкта, що моделюється деяким автоматом, визначаються, з одного боку, загальним формалізмом автомата, а з іншого - його графічним зображенням у мові UML у формі конкретної діаграми станів.

Стан

У мові UML під *станом* розуміється абстрактний мета клас, використовуваний для моделювання окремої ситуації, протягом якої має місце виконання деякої умови. Стан може бути заданий у вигляді набору конкретних значень атрибутів класу або об'єкта, при цьому зміна їх окремих значень буде відбивати зміну стану моделюємого класу або об'єкта.

Не кожний атрибут класу може характеризувати його стан. Як правило, мають значення тільки такі властивості елементів системи, які відбивають динамічний або функціональний аспект її поведінки. У цьому випадку стан буде характеризуватися деякою інваріантною умовою, що включають у себе тільки значимі для поведінки класу атрибути і їх значення.

Елемент переходить у стан в момент початку відповідної діяльності й залишає даний стан у момент її завершення.

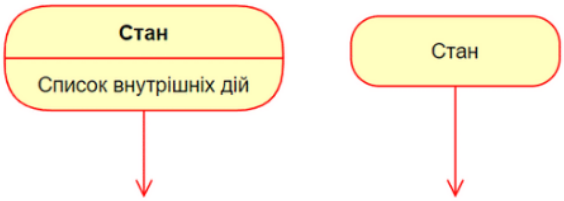


Рисунок 2 - Графічне зображення станів на діаграмі станів

Стан на діаграмі зображується прямокутником з округленими вершинами (рис. 2). Цей прямокутник, у свою чергу, може бути розділений на дві секції горизонтальною лінією. Якщо зазначена лише одна секція, то в ній записується тільки ім'я стану. А якщо ні, то в першій з них записується ім'я стану, а в другий - список деяких *внутрішніх дій* або *переходів* у даному стані. При цьому під *дією* в мові UML розуміють деяку атомарну операцію, виконання якої приводить до зміни стану або поверненню деякого значення (наприклад, "істина" або "неправда").

Лекція №11

Список внутрішніх дій

Ця секція містить перелік внутрішніх дій або *діяльностей*, які виконуються в процесі знаходження елемента в даному стані. Кожне з дій записується у вигляді окремого рядка й має наступний формат:

< мітка-дії '/' вираження - дії>

Мітка дії вказує на обставини або умови, при яких буде виконуватися діяльність, певна вираженням дії. При цьому *вираження дії* може використовувати будь-які атрибути й зв'язки, які належать області імен або контексту об'єкта.

Перелік міток дії має фіксовані значення в мові UML, які не можуть бути використані в якості імен подій. Ці значення наступні:

- entry - ця мітка вказує на дію, специфікована наступним за нею вираженням дії, яка виконується в момент входу в даний стан (вхідна дія);
- exit - ця мітка вказує на дію, специфікована наступним за нею вираженням дії, яка виконується в момент виходу з даного стану (вихідна дія);
- do - ця мітка специфицирует діяльність, що виконується ("do activity"), яка виконується протягом усього часу, поки об'єкт перебуває в даному стані, або доти , поки не закінчиться обчислення, специфіковане наступним за нею вираженням дії.

В останньому випадку при завершенні події генерується відповідний результат;

- include - ця мітка використовується для звертання до під-автомату, при цьому наступне за нею вираження дії містить ім'я цього під-автомату.

Як приклад стану розглянемо ситуацію введення пароля користувача при аутентифікації входу в деяку програмну систему. У цьому випадку список

внутрішніх дій у даному стані не порожній і включає 4 окремих дії, перші два з яких стандартні й описані вище, а два останні визначаються своєю специфікацією.

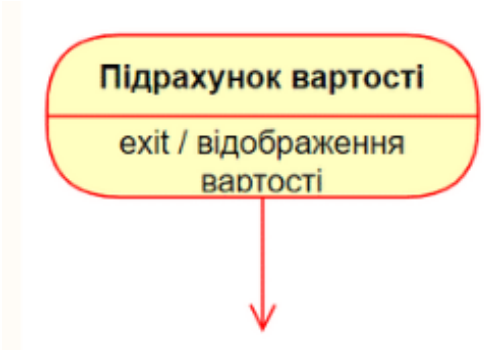


Рисунок 3 - Приклад стану з непустою секцією внутрішніх дій

Початковий стан

Початковий стан являє собою окремий випадок стану, який не містить ніяких внутрішніх дій (псевдостан). У цьому стані перебуває об'єкт за замовчуванням у початковий момент часу. Воно служить для вказівки на діаграмі станів графічної області, від якої починається процес зміни станів.

Рисунок 4 – Початковий та кінцевий стани

Кінцевий стан

Кінцевий (фінальне) стан являє собою окремий випадок стану, який також не містить ніяких внутрішніх дій (псевдостан). У цьому стані буде перебувати об'єкт за замовчуванням після завершення роботи автомата в кінцевий момент часу. Воно служить для вказівки на діаграмі станів графічної області, у якій завершується процес зміни станів або життєвий цикл даного об'єкта.

Перехід

Простий перехід (simple transition) являє собою відношення між двома послідовними станами, яке вказує на факт зміни одного стану іншим. Перебування об'єкта в першому стані може супроводжуватися виконанням деяких дій, а перехід у другий стан буде можливий після завершення цих дій, а також після задоволення деяких додаткових умов. У цьому випадку говорять, що перехід спрацьовує, Або відбувається спрацьовування переходу. До спрацьовування переходу об'єкт перебуває в попередньому від нього стані, називаним вихідним станом, або в джерелі, а після його спрацьовування об'єкт перебуває наступному від нього стані (цільовому стані).

Перехід здійснюється при настанні деякої події: закінчення виконання діяльності (do activity), одержанні об'єктом повідомлення або прийманням сигналу. Спрацьовування переходу може залежати не тільки від настання деякої події, але й від виконання певного умови, називаної *сторожовою умовою*. Об'єкт перейде з одного стану в інше в тому випадку, якщо відбулася зазначена подія й сторожова умова прийняла значення "істина".

На діаграмі станів перехід зображується суцільною лінією зі стрілкою, яка спрямована в цільовий стан (наприклад, "вихід з ладу" на мал. 1). Кожний перехід може бути позначений рядком тексту, який має наступний загальний формат:

<сигнатура події>'['<сторожова умова>']' <вираження дії>.

При цьому сигнатура події описує деяка подія з необхідними аргументами:

<ім'я події>'('<список параметрів, розділених комами>')'.

Подія

Подія являє собою специфікацію деякого факту, що має місце в просторі й у часі. Про події говорять, що вони "відбуваються", при цьому окремі події повинні бути впорядковані в часі. Після настання деякої події не можна вже повернутися до попередніх подій, якщо така можливість не передбачена явно в моделі.

Семантика поняття події фіксує увагу на зовнішніх проявах якісних змін, що відбуваються при переході моделируемого об'єкта зі стану в стан. Наприклад, при включенні електричного перемикача відбувається деяка подію, у результаті якого кімната стає освітленою. Після успішного ремонту комп'ютера також відбувається немаловажна подія - відновлення його працездатності. Якщо підняти трубку звичайного телефону, те, у випадку його справності, ми очікуємо почути тоновий сигнал. І цей факт теж є подією.

У мові UML події відіграють роль стимулів, які ініціюють переходи з одних станів в інші. У якості подій можна розглядати сигнали, виклики, закінчення фіксованих проміжків часу або моменти закінчення виконання певних дій. Перехід *тригерний*, тобто специфікован подією-тригером.

Якщо поруч зі стрілкою переходу не зазначений ніякий рядок тексту, то відповідний перехід є не тригерним, і в цьому випадку з контексту діаграми станів повинне бути ясно, після закінчення якої діяльності він спрацьовує.

Сторожова умова

Сторожова умова (guard condition), якщо вона є, завжди записується в прямих дужках після події-тригера і являє собою деякий булевський вираз. Нагадаємо, що булевское вираження повинне ухвалювати одне їх двох значень, що взаємно виключають: "істина" або "неправда".

Рисунок 5 – Діаграма станів для прикладу включення ПК

Тема 2.1.8 Діаграми діяльності

Лекція №12

(4 години)

Мета: Розглянути основні поняття діаграми діяльності. Ознайомитись з графічною нотацією діаграми.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Актуалізація опорних знань студентів:
   a. Повідомлення теми та мети заняття;
   b. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Стан дії;
2. Переходи;
3. Доріжки;
4. Об’єкти;

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичним матеріалом теми 2.1.8.
2. Вивчити основні поняття лекції.
3. Відповісти на контрольні питання.

Контрольні питання:

1. В чому різниця діаграми діяльності від діаграми станів?
2. Які елементи використовуються на діаграмі діяльності?
3. Що таке діяльність?
4. Що таке стан дії?
5. Які переходи використовуються на цих діаграмах?
6. Для чого необхідні доріжки?
7. Що відображають об’єкти?
8. Як відобразити паралельні дії?

При моделюванні поведінки проектованої або аналізованої системи виникає необхідність не тільки представити процес зміни її станів, але й деталізувати особливості алгоритмічної й логічної реалізації виконуваних системою операцій. Традиційно для цієї мети використовувалися блок-схеми або структурні схеми алгоритмів. Кожна така схема акцентує увагу на послідовності виконання певних дій або елементарних операцій, які в сукупності приводять до одержання бажаного результату.

Важливо підкреслити та обставина, що зі збільшенням складності системи строге дотримання послідовності виконуваних операцій здобуває усе більш важливе значення. Якщо спробувати заварити кава холодною водою, то ми можемо тільки зіпсувати одну порцію напою. Порушення послідовності операцій при ремонті двигуна може привести до його поломки або виходу з ладу.

Для моделювання процесу виконання операцій у мові UML використовуються так звані *діаграми діяльності*. Застосовувана в них графічна нотація багато в чому схожа на нотацію діаграми станів, оскільки на діаграмах діяльності також присутні позначення станів і переходів. Відмінність полягає в семантиці станів, які використовуються для вистави не діяльностей, а дій, і у

відсутності на переходах сигнатури подій. Кожний стан на діаграмі діяльності відповідає виконанню деякої елементарної операції, а перехід у наступний стан спрацьовує тільки при завершенні цієї, операції в попередньому стані. Графічно діаграма діяльності представляється у формі графа діяльності, вершинами якого є стани дії, а дугами - переходи від одного стану дії до іншого.

Таким чином, діаграми діяльності можна вважати часткам случаємо діаграм станів. Саме вони дозволяють реалізувати в мові UML особливості процедурного й синхронного керування, обумовленого завершенням внутрішніх діяльностей і дій. Метамодель UML надає для цього необхідні терміни й семантику. Основним напрямком використання діаграм діяльності є візуалізація особливостей реалізації операцій класів, коли необхідно представити алгоритми їх виконання. При цьому кожний стан може бути виконанням операції деякого класу або її частини, дозволяючи використовувати діаграми діяльності для опису реакцій на внутрішні події системи.

У контексті мови UML *діяльність (activity)* являє собою деяку сукупність окремих обчислень, виконуваних автоматом. При цьому окремі елементарні обчислення можуть приводити до деякого результату або дії (action). На діаграмі діяльності відображається логіка або послідовність переходу від однієї діяльності до іншої, при цьому увага фіксується на результаті діяльності. Сам же результат може привести до зміни стану системи або поверненню деякого значення.

Стан дії

Стан дії (action state) є спеціальним випадком стану з деякою вхідною дією й принаймні одним вихідним зі стану переходом. Цей перехід неявно припускає, що вхідна дія вже завершилася. Стан дії не може мати внутрішніх переходів, оскільки воно є елементарним. Звичайне використання стану дії полягає в моделюванні одного кроку виконання алгоритму (процедури) або потоку керування.

Графічно стан дії зображується фігурою, що нагадує прямокутник, бічні сторони якого замінені опуклими дугами (рис. 1). Усередині цієї фігури записується вираження дії (action-expression), яка повинна є унікальним у межах однієї діаграми діяльності.

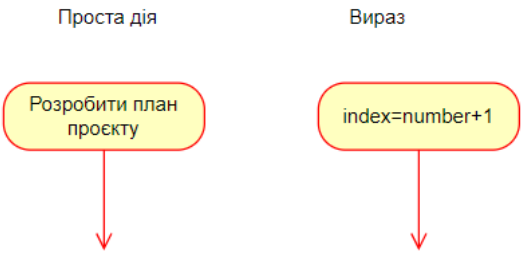


Рисунок 1 – Графічне зображення стану дії

Дія може бути записане природньою мовою, деякому псевдокоді або мові програмування. Ніяких додаткових або неявних обмежень при записі дій не накладається.

Іноді виникає необхідність представити на діаграмі діяльності деяка складна дія, яка, у свою чергу, складається з декількох більш простих дій. У цьому випадку можна використовувати спеціальне позначення так званого стану під-діяльності (subactivity state). Такий стан є графом діяльності й позначається спеціальною піктограмою в правому нижньому куті символу стану дії (рис. 2). Ця конструкція може застосовуватися до будь-якого елемента мови UML, який підтримує "вкладеність" своєї структури. При цьому піктограма може бути додатково позначена типом вкладеної структури.

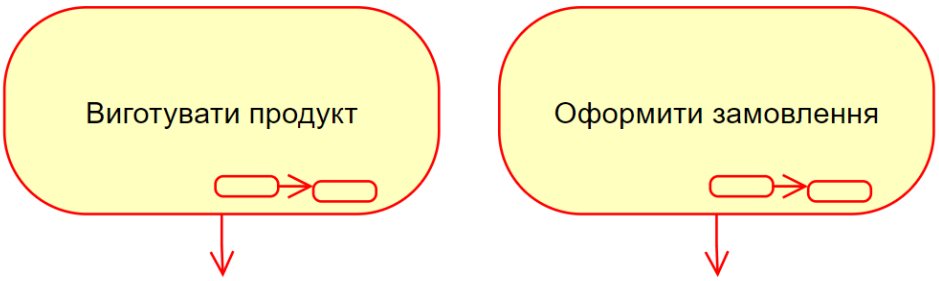


Рисунок 2 - Графічне зображення стану під-діяльності

Кожна діаграма діяльності повинна мати єдиний початковий і єдиний кінцевий стану. Вони мають такі ж позначення, як і на діаграмі станів. При цьому кожна діяльність починається в початковому стані й закінчується в кінцевому стані. Саму діаграму діяльності прийнято мати у своєму розпорядженні такий образ, щоб дії випливали зверху вниз. У цьому випадку початковий стан буде зображуватися у верхній частині діаграми, а кінцеве - у її нижній частині.

Переходи

При побудові діаграми діяльності використовуються тільки *не тригерні переходи,* тобто такі, які спрацьовують відразу після завершення діяльності або виконання відповідного дії. Цей перехід переводить діяльність у наступний стан відразу, як тільки закінчиться дія в попередньому стані. На діаграмі такий перехід зображується суцільною лінією зі стрілкою.

Якщо зі стану дії виходить єдиний перехід, то він може бути ніяк не позначений. Якщо ж таких переходів кілька, то спрацювати може тільки один з них. Саме в цьому випадку для кожного з таких переходів повинне бути явно записана *сторожова умова* в прямих дужках. При цьому для всіх вихідних з деякого стану переходів повинне виконуватися вимога істинності тільки одного з них. Подібний випадок зустрічається тоді, коли послідовно виконувана діяльність повинна розділитися на альтернативні галузі залежно від значення деякого проміжного результату. Така ситуація одержала назву *розгалуження*, а для її позначення застосовується спеціальний символ.

Графічно розгалуження на діаграмі діяльності позначається невеликим ромбом, усередині якого немає ніякого тексту (рис. 3). У цей ромб може входити тільки одна стрілка від того стану дії, після виконання якого потік керування повинен бути продовжений по одній з галузей, що взаємно виключають. Прийняте вхідну стрілку приєднувати до верхньої або лівій вершині символу розгалуження. Вихідних стрілок може бути дві або більше, але для кожної з них явно вказується відповідна сторожова умова у формі булевского вираження.

Як приклад розглянемо фрагмент відомого алгоритму знаходження корнів квадратного рівняння. У загальному випадку після приведення рівняння другому ступеня до канонічного виду: a*x*x+в*x+c=0 необхідно обчислити його дискримінант. Причому, у випадку негативного дискримінанта рівняння не має розв'язку на безлічі дійсних чисел, і подальші обчислення повинні бути припинені. При ненегативному дискримінанті рівняння має розв'язок, коріння якого можуть бути отримані на основі конкретної розрахункової формули. .

Графічно фрагмент процедури обчислення корінь квадратного рівняння може бути представлений у вигляді діаграми діяльності із трьома станами дії й розгалуженням (рис. 10.3). Кожний з переходів, що виходять зі стану "Обчислити дискримінант", має сторожову умову, що визначає єдину галузі, по якій може бути

продовжений процес обчислення корінь залежно від знака дискримінанта. Очевидно, що у випадку його заперечності, ми відразу попадаємо в кінцевий стан, тим самим завершуючи виконання алгоритму в цілому.

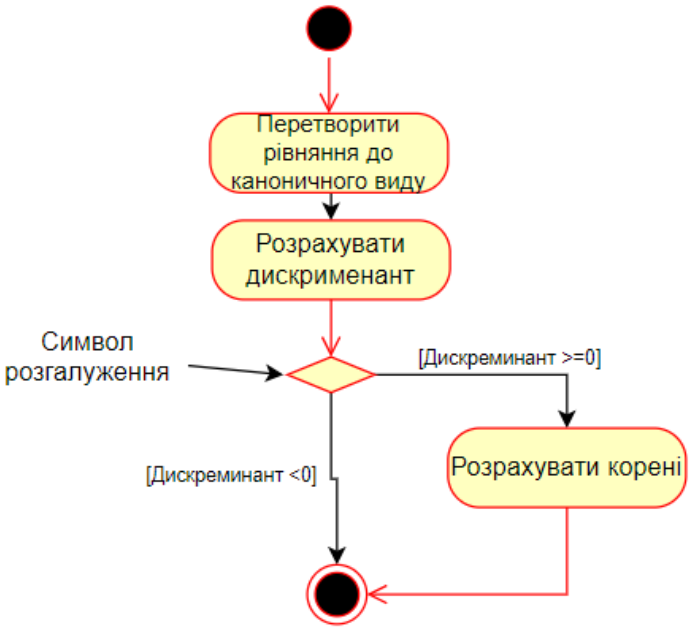


Рисунок 3 - Фрагмент діаграми діяльності для алгоритму знаходження корінь квадратного рівняння

У розглянутому прикладі, як видне з мал. 3, виконувані дії з'єднуються в кінцевому стані. Однак це зовсім не є обов'язковим. Можна зобразити ще один символ розгалуження, який буде мати кілька вхідних переходів і один вихідний.

У наступному прикладі (рис. 4) розраховується загальна вартість товарів, що купуються по кредитній картці в супермаркеті. Якщо ця вартість перевищує $50, то виконується аутентифікація особистості власника картки. У випадку позитивної перевірки (картка дійсна) або якщо вартість товарів не перевищує $50, відбувається зняття суми з рахунку й оплата вартості товарів. При негативному результаті (картка недійсна) оплати не відбувається, і товар залишається в продавця.

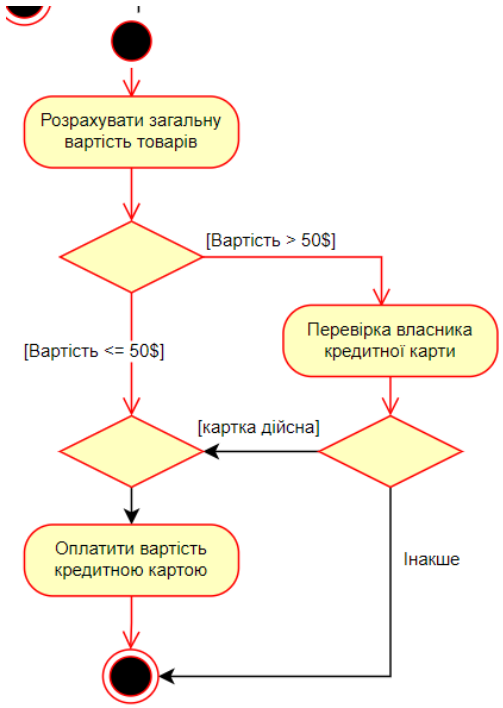


Рисунок 4 - Різні варіанти розгалужень на діаграмі діяльності

Один з найбільш значимих недоліків звичайних блок-схем або структурних схем алгоритмів пов'язаний із проблемою зображення паралельних галузей окремих обчислень. Оскільки распараллеливание обчислень суттєво підвищує загальна швидкодія програмних систем, необхідні графічні примітиви для вистави паралельних процесів. У мові UML для цієї мети використовується спеціальний символ для поділу й злиття паралельних обчислень або потоків керування. Таким символом є пряма риска, аналогічно позначенню переходу у формалізмі мереж Петри.

Як правило, така риска зображується відрізком горизонтальної лінії, товщина якої трохи ширше основних суцільних ліній діаграми діяльності. При цьому поділ (concurrent fork) має один вхідний перехід і кілька вихідних (рис. 5, а). Злиття (concurrent join), навпаки, має кілька вхідних переходів і один вихідний (рис. 5, б).

Рисунок 10.5 - Графічне зображення поділу й злиття паралельних потоків керування

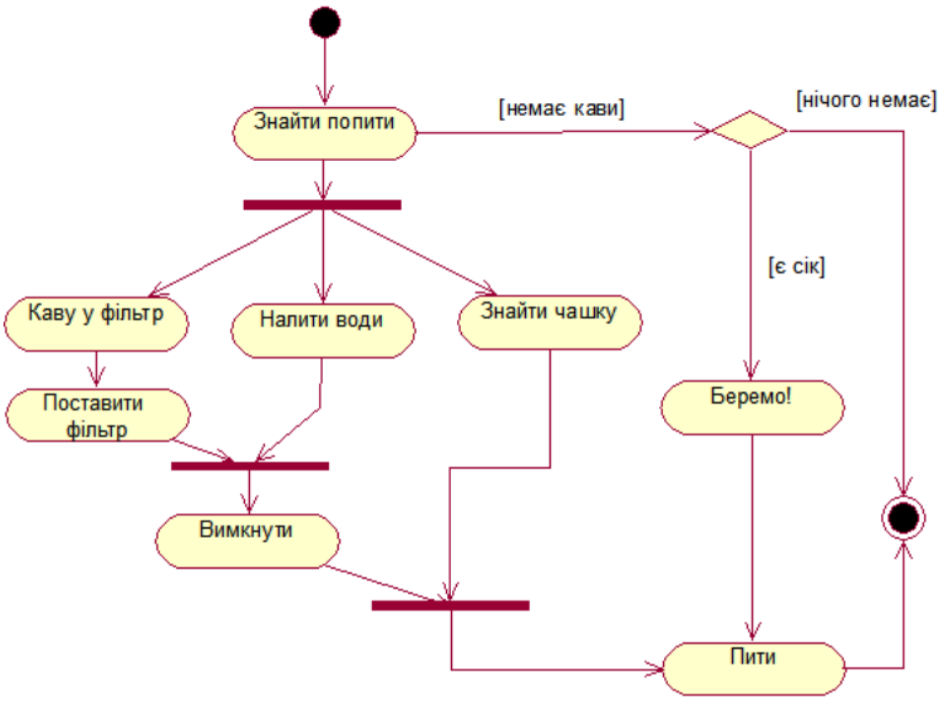


Рисунок 6 - Діаграма діяльності для прикладу з готуванням напою

Для ілюстрації особливостей паралельних процесів виконання дій розглянемо вже класичним приклад, що став, з готуванням напою. Гідність цього прикладу полягає в тому, що він практично не вимагає ніяких додаткових пояснень у силу своєї очевидності (рис. 6).

Таким чином, діаграма діяльності є не що інше, як спеціальний випадок діаграми станів, у якій усі або більшість станів є діями або станами під-діяльності. А всі або більшість переходів є не тригерні переходами, які спрацьовують по завершенню дій або під-діяльностей у станах-джерелах.

Лекція №13

Доріжки

Діаграми діяльності можуть бути використані не тільки для специфікації алгоритмів обчислень або потоків керування в програмних системах. Не менш важлива область їх застосування пов'язана з моделюванням бізнес-процесів. Дійсно, діяльність будь-якої компанії (фірми) також являє собою не що інше, як сукупність окремих дій, спрямованих на досягнення необхідного результату. Однак стосовно до бізнес-процесам бажане виконання кожної дії асоціювати з конкретним підрозділом компанії. У цьому випадку підрозділ відповідає за реалізацію окремих дій, а сам бізнес-процес представляється у вигляді переходів дій з одного підрозділу до іншого.

Для моделювання цих особливостей у мові UML використовується спеціальна конструкція назва, що одержала, доріжки (swimlanes). Мається на увазі візуальна аналогія із плавальними доріжками в басейні, якщо дивитися на відповідну діаграму. При цьому всі стани дії на діаграмі діяльності діляться на окремі групи, які відділяються друг від друга вертикальними лініями. Дві сусідні лінії й утворюють доріжку, а група станів між цими лініями виконується окремим підрозділом (відділом, групою, відділенням, філією) компанії (рис. 7).

Назви підрозділів явно вказуються у верхній частині доріжки. Перетинати лінію доріжки можуть тільки переходи, які в цьому випадку позначають вихід або вхід потоку керування у відповідний підрозділ компанії. Порядок проходження доріжок не несе якої-небудь семантичної інформації й визначається міркуваннями зручності.

Як приклад розглянемо фрагмент діаграми діяльності торговельної компанії, що обслуговує клієнтів по телефону. Підрозділами компанії є відділ приймання й оформлення замовлень, відділ продажів і склад.

Цим підрозділам будуть відповідати три доріжки на діаграмі діяльності, кожна з яких специфицирует зону відповідальності підрозділу. У цьому випадку діаграма діяльності містить у собі не тільки інформацію про послідовність виконання робочих дій, але й про те, яке з підрозділів торговельної компанії повинне виконувати те або інша дія (рис. 7).

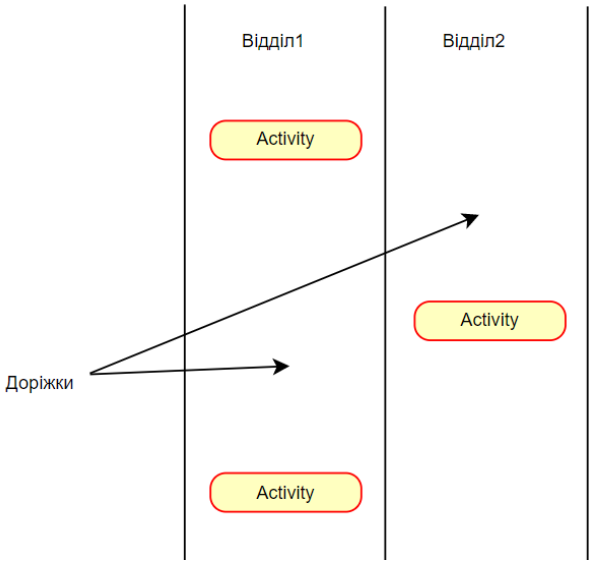


Рисунок 7 - Варіант діаграми діяльності з доріжками

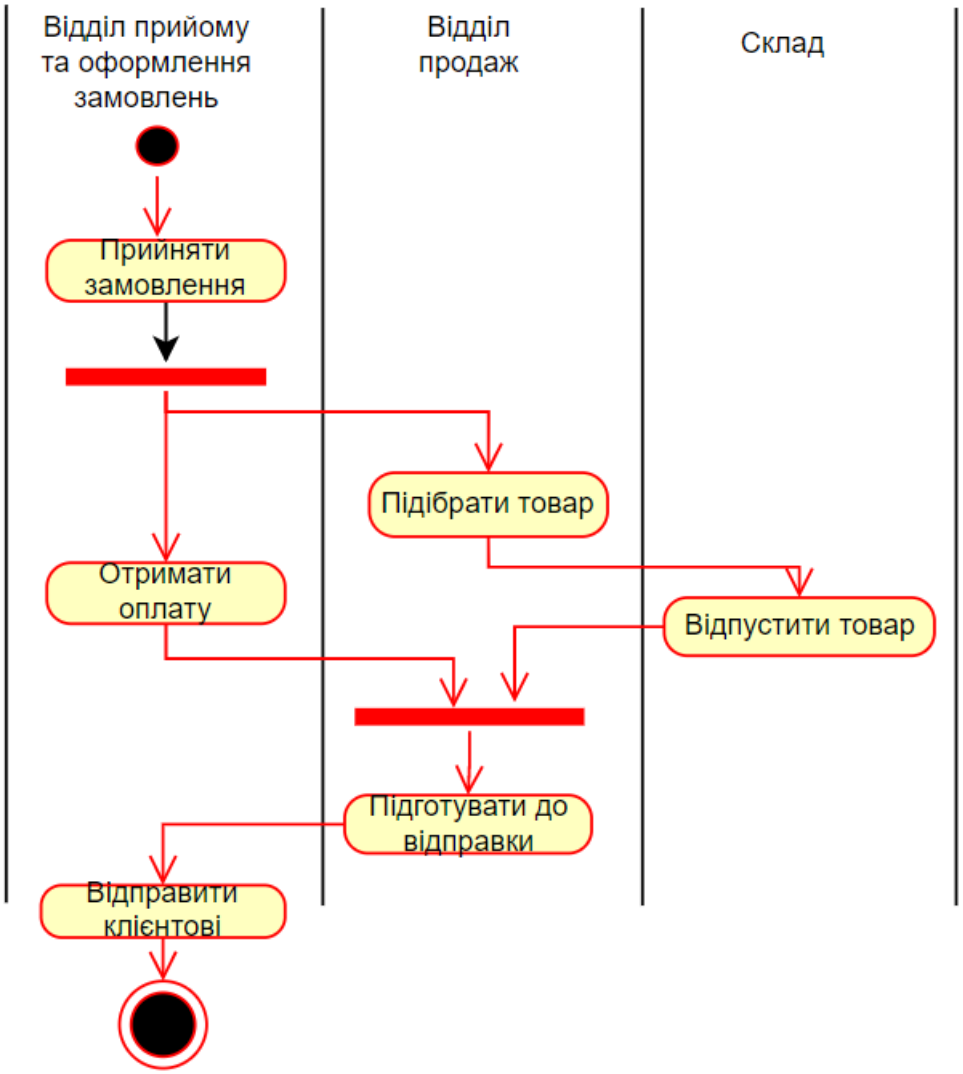


Рисунок 8 - Фрагмент діаграми діяльності для торговельної компанії

Із зазначеної діаграми діяльності відразу видне, що після прийняття замовлення від клієнта відділом приймання й оформлення замовлень здійснюється распараллеливание діяльності на два потоки (перехід-поділ). Перший з них залишається в цьому ж відділі й пов'язаний з одержанням оплати від клієнта за замовлений товар. Другий ініціює виконання дії по добору товару у відділі продажів (модель товару, розміри, колір, рік випуску та ін.). По закінченню цієї роботи ініціюється дія по відпустці товару зі складу. Однак підготовка товару до відправлення в торговельному відділі починається тільки після того, як буде отримана оплата за товар від клієнта й товар буде відпущений зі складу (перехід-з'єднання). Тільки після цього товар відправляється клієнтові, переходячи в його власність.

Об'єкти

У загальному випадку дії на діаграмі діяльності виконуються над тими або іншими об'єктами. Ці об'єкти або ініціюють виконання дій, або визначають деякий результат цих дій. При цьому дії специфицируют виклики, які передаються від одного об'єкта графа діяльності до іншого. Оскільки в такому ракурсі об'єкти відіграють певну роль у розумінні процесу діяльності, іноді виникає необхідність явно вказати їх на діаграмі діяльності.

Для графічної вистави об'єктів використовуються прямокутник класу, з тим відмінністю, що ім'я об'єкта підкреслюється. Далі після імені може вказуватися характеристика стану об'єкта в прямих дужках. Такі прямокутники об'єктів приєднуються до станів дії відношенням залежності пунктирною лінією зі стрілкою. Відповідна залежність визначає стан конкретного об'єкта після виконання попереднього дії.

На діаграмі діяльності з доріжками розташування об'єкта може мати деякий додатковий зміст. А саме, якщо об'єкт розташовано на границі двох доріжок, те це може означати, що перехід до наступного стану дії в сусідній доріжці асоційований з готовністю деякого документа (об'єкт у деякому стані). Якщо ж об'єкт цілком розташований усередині доріжки, то й стан цього об'єкта цілком визначається діями даної доріжки.

Вертаючись до попереднього прикладу з торговельною компанією, можна помітити, що центральним об'єктом процесу продажу є замовлення або вірніше стан його виконання. Спочатку до дзвінка від клієнта замовлення як об'єкт відсутній і виникає лише після такого дзвінка. Однак це замовлення ще не заповнене до кінця, оскільки потрібно ще підібрати конкретний товар у відділі продажів. Після його підготовки він передається на склад, де разом з відпусткою товару замовлення остаточно дооформляется. Нарешті, після одержання підтвердження про оплату товару ця інформація заноситься в замовлення, і він уважається виконаним і закритим. Дана інформація може бути представлена графічно у вигляді модифікованого варіанта діаграми діяльності цієї ж торговельної компанії (рис. 9).

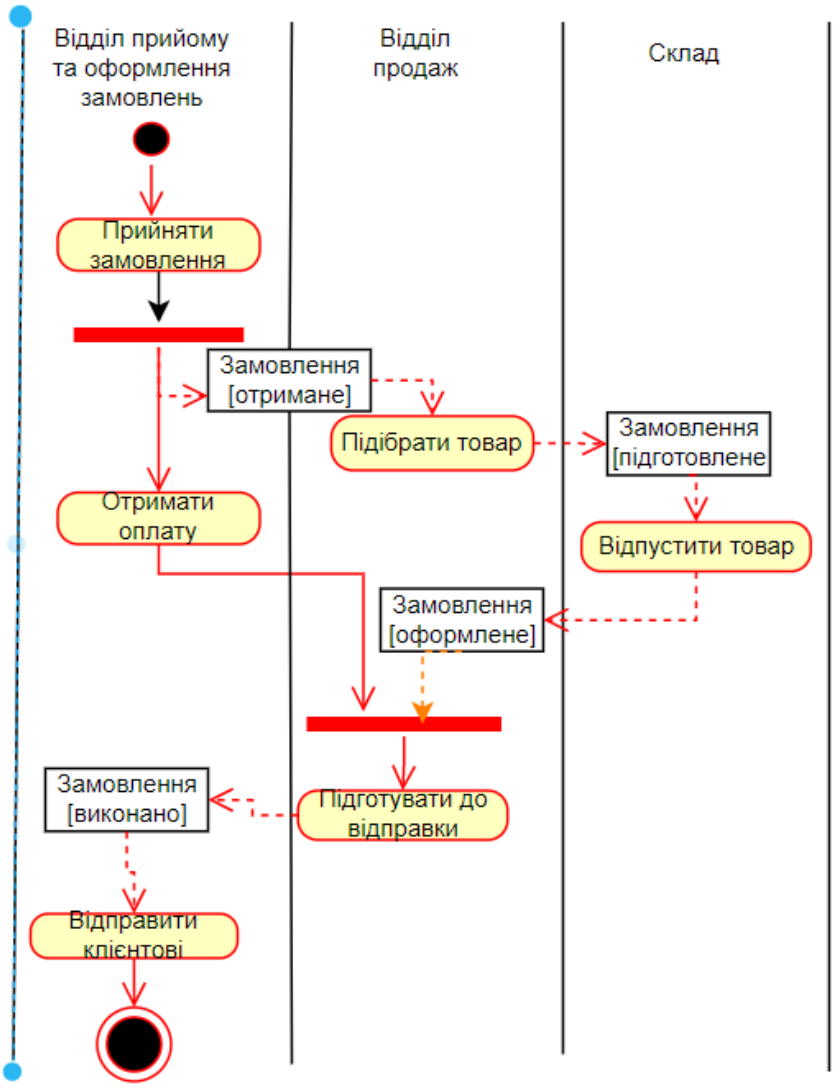


Рисунок 9 - Фрагмент діаграми діяльності торговельної компанії з об'єктом-замовленням

На закінчення слід зупинитися на необхідності синхронізації окремих дій на діаграмі діяльності. Така необхідність виникає щораз , коли паралельно виконувані дії впливають один на одного. Синхронізація паралельних процесів може бути реалізована за допомогою переходів "поділ-злиття".

Як приклад розглянемо спрощену ситуацію з моделюванням процесу будівлі будинку. В цьому прикладі будівля будинку містить у собі будівельні роботи (зведення фундаменту й стін, зведення даху й опоряджувальні роботи) і роботи з електрифікації будинку (підведення електричної лінії, прокладка схованої електропроводки й установка освітлювальних ламп). Синхронізація паралельного виконання цього комплексу робіт може бути явно зазначена на діаграмі діяльності (рис. 10).

Рисунок 10 - Діаграма діяльності із синхронізацією паралельних дій

У розглянутому прикладі всі стани є станами під-діяльності. Це означає, що кожне з них можна деталізувати у вигляді окремого графа діяльності з відповідною діаграмою. Дійсно, стан під-діяльності "Підготовка ділянки" може містити в собі такі дії, як очищення ділянки від дерев, вивіз цих дерев за межі ділянки, риття котловану під фундамент, установка тимчасових будов для складування будівельних матеріалів і інші роботи.

Тема 2.1.9 Компонентні діаграми

Лекція №14

(2 години)

Мета: Розглянути основні поняття компонентної діаграми. Ознайомитись з графічною нотацією діаграми.

Структура заняття:

I. Організаційний момент:

1. Готовність групи до заняття;

2. Психоемоційний настрій;
3. Перевірка присутніх.

II. Актуалізація опорних знань студентів:

1. Повідомлення теми та мети заняття;
2. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Компонентні діаграми;
2. Компоненти;
3. Інтерфейси;
4. Компонування системи;
5. Різновиди компонентів;
6. Використання компонентних діаграм;
7. Моделювання програмного тексту системи.

IV. Підведення підсумків занять;

V. Домашнє завдання:

1. Ознайомитись з теоретичним матеріалом теми 2.1.9.
2. Вивчити основні поняття лекції.
3. Відповісти на контрольні питання.

Контрольні питання:

1. У чому основне призначення моделей реалізації?
2. Які вершини й дуги утворюють компонентну діаграму?
3. Що таке компонент? Чим він відрізняється від класу?
4. Що таке інтерфейс?
5. Які форми вистави інтерфейсу ви знаєте?
6. Чим корисний інтерфейс?
7. Які різновиди компонентів ви знаєте?
8. Для чого використовують компонентні діаграми?

Статичні й динамічні моделі описують логічну організацію системи, відбивають логічний мир програмного додатка. Моделі реалізації забезпечують

вистава системи у фізичному світі, розглядаючи питання впакування логічних елементів у компоненти й розміщення компонентів в апаратних вузлах.

Компонентні діаграми

Компонентна діаграма - перша із двох різновидів діаграм реалізації, що моделюють фізичні аспекти об'єктно-орієнтованих систем. Компонентна діаграма показує організацію набору компонентів і залежності між компонентами.

Елементами компонентних діаграм є компоненти й інтерфейси, а також відносини залежності й реалізації. Як і інші діаграми, компонентні діаграми можуть включати примітки й обмеження. Крім того, компонентні діаграми можуть містити пакети або підсистеми, використовувані для угруповання елементів моделі у великі фрагменти.

Компоненти

По своїй суті компонентів є фізичним фрагментом реалізації системи, який містить у собі програмний код (вихідний, двійковий, що виконується), сценарні описи або набори команд операційної системи (маються на увазі командні файли). Мова UML дає наступне визначення.

Компонент - фізична й замінна частина системи, яка відповідає набору інтерфейсів і забезпечує реалізацію цього набору інтерфейсів.

Інтерфейс - дуже важлива частина поняття "компонент", його ми обговоримо в наступному підрозділі. Графічно компонент зображується як прямокутник із вкладками, що звичайно включає ім'я (рис. 1).

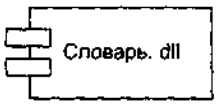


Рисунок 1 - Позначення компонента

Компонент - базисний будівельний блок фізичного представлення ПЗ, тому цікаво зрівняти його з базисним будівельним блоком логічного представлення ПЗ - класом.

Подібні характеристики компонента й класу:

- наявність імені;
- реалізація набору інтерфейсів;
- участь у відносинах залежності;
- можливість бути вкладеним;

 наявність екземплярів (екземпляри компонентів можна використовувати тільки в діаграмах розміщення).

Ви скажете - багато загального. І проте між компонентами й класами є істотна різниця, її характеризує табл. 1.

Таблиця 1. Відмінності компонентів і класів

№	Опис
1	Класи - логічні абстракції, компоненти - фізичні предмети, які живуть у світі битов. Зокрема , компоненти можуть "жити" у фізичних вузлах, а класи позбавлені такої можливості
2	Компоненти є фізичними впакуваннями, контейнерами, инкапсулирующими в собі різні логічні елементи. Вони - елементи абстракцій іншого рівня
3	Класи мають властивості й операції. Компоненти мають тільки операції, які доступні через їхні інтерфейси

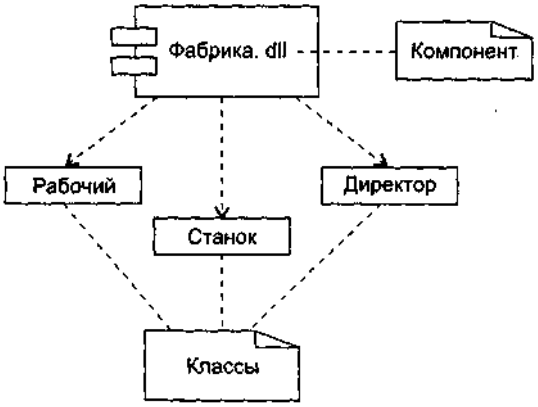


Рисунок 2 - Класи в компоненті

Про що говорять ці відмінності? По-перше, клас не може "дихати" повітрям фізичного миру реалізації. Йому потрібний скафандр. Таким скафандром є компонент.

По-друге, їм не жити друг без друга - порожні скафандри нікому не потрібні. Причому в скафандрі-компоненті може перебувати кілька класів і кооперацій. Отже, у скафандрі - фізичної реалізації - розташовується набір логіки. Як показано на мал. 2, за допомогою відносини залежності можна явно відобразити відношення між компонентом і класами, які він реалізує. Правда, найчастіше такі відносини не відображаються. Їх зручно представляти в компонентній специфікації.

По-третє, клас - душа нараспашку (він може навіть показати свої властивості). Компонент завжди застебнуть на всі ґудзики (правда, з нього стирчать интерфейсні рознімання операцій).

Тепер доречно перейти до обговорення інтерфейсів.

Інтерфейси

Інтерфейс - список операцій, які визначають послуги класу або компонента. Образно говорячи, інтерфейс - це рознімання, яке стирчить із шухлядки компонента. За допомогою интерфейсних рознімань компонента стикуються один з одним, поєднуючись у систему.

Ще одна аналогія. Інтерфейс подібний абстрактного класу, у якого відсутні властивості й працююча операції, а є тільки абстрактні операції (щоне мають тіл). Якщо прагнете, інтерфейс схожий на посмішку чеширского кота із правдивої історії про Алісу, де кіт окремо й посмішка окремо. Усі операції інтерфейсу відкриті й видимі клієнтові (а якщо ні, то вони втратили б усякий зміст). Отже, операції інтерфейсу тільки йменують пропоновані послуги, не більш того.

Дуже важливий взаємозв'язок між компонентом і інтерфейсом. Можливі два способи відображення взаємозв'язки між компонентом і його інтерфейсами. У першому, згорнутому способі, як показано на мал. 3, інтерфейс зображується у формі піктограми. Компонент Образ.java, який реалізує інтерфейс, з'єднується зі значком інтерфейсу (кружком) НаблюдательОбраза простий лінією. Компонент РыцарьПечальногоОбраза.jaya, який використовує інтерфейс, пов'язаний з ним відношенням залежності.

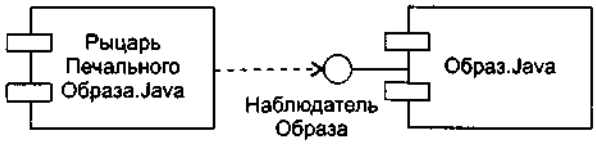


Рисунок 3 - Вистава інтерфейсу у формі піктограми

Другий спосіб вистави інтерфейсу ілюструє мал. 22.4. Тут використовується розгорнута форма зображення інтерфейсу, у якій можуть показуватися його операції. Компонент, який реалізує інтерфейс, підключається до нього відношенням реалізації. Компонент, який одержує доступ до послуг іншого компонента через інтерфейс, як і раніше підключається до інтерфейсу відношенням залежності.

Рисунок 4 - Розгорнута форма представлення інтерфейсу

По способу зв'язку компонента з інтерфейсом розрізняють:

- експортований інтерфейс - той, який компонент реалізує й пропонує як послугу клієнтам;
- імпортований інтерфейс - той, який компонент використовує як послугу іншого компонента.

В одного компонента може бути трохи експортованих і кілька імпортованих інтерфейсів.

Той факт, що між двома компонентами завжди перебуває інтерфейс, усуває їхню пряму залежність. Компонент, що використовує інтерфейс, буде функціонувати правильно незалежно від того, який компонент реалізує цей інтерфейс. Це дуже важливо й забезпечує гнучку заміну компонентів в інтересах розвитку системи.

Компонування системи

За останні піввіку розроблювачі апаратури пройшли шлях від комп'ютерів розміром з кімнату до малюсіньких "ноутбуків", що забезпечили зрослі функціональні можливості. За ті ж піввіку розроблювачі програмного забезпечення пройшли шлях від великих систем на Асемблері й Фортрані до ще більших систем на C++ і Java. На жаль, але програмний інструментарій розбудовується повільніше, чим апаратний інструментарій.

Цей секрет - компонента. Розроблювач апаратури створює систему з готових апаратних компонентів (мікросхем), що виконують певні функції, що й надають набір послуг через ясні інтерфейси. Завдання конструкторів спрощується за рахунок повторного використання результатів, отриманих іншими.

Повторне використання - магістральний шлях розвитку програмного інструментарію. Створення нового ПЗ з існуючих, працездатних програмних компонентів приводить до більш надійного й дешевого коду. При цьому строки розробки суттєво скорочуються.

Основна мета програмних компонентів - допускати складання системи із двійкових замінних частин. Вони повинні забезпечити початкове створення системи з компонентів, а потім і її розвиток - додавання нових компонентів і заміну деяких старих компонентів без перебудови системи в цілому. Ключ до втілення такої можливості - інтерфейси. Після того як інтерфейс визначений, до виконуваної системи можна підключити будь-який компонент, який задовольняє йому або забезпечує цей інтерфейс. Для розширення системи проводяться компоненти, які забезпечують додаткові послуги через нові інтерфейси. Такий підхід ґрунтується на особливостях компонента:

- Компонент фізичний. Він живе у світі бітів, а не логічних понять і не залежить від мови програмування
- Компонент - замінний елемент. Властивість заменяемості дозволяє замінити один компонент іншим компонентом, який задовольняє тим же інтерфейсам. Механізм заміни застережений сучасними компонентними моделями (COM, COM+, CORBA, Java Beans), що вимагають незначних перетворень або, що надають утиліти, які автоматизують механізм
- Компонент є частиною системи, він рідко автономний. Частіше компонент співробітничає з іншими компонентами й існує в архітектурній або технологічному середовищі, призначеної для його використання. Компонент зв'язаний і фізично, і логічно, він позначає фрагмент великої системи
- Компонент відповідає набору інтерфейсів і забезпечує реалізацію цього набору інтерфейсів

Різновиди компонентів

Мир сучасних компонентів досить широкий і різноманітний. У мові UML для позначення нових різновидів компонентів використовують механізм стереотипів. Стандартні стереотипи, передбачені в UML для компонентів:

"executable" - Компонент, який може виконуватися у фізичному вузлі (має розширення .exe)

"library" - Статична або динамічна об'єктна бібліотека (має розширення .dll)

"file" - Компонент, який представляє файл, що містить вихідний код або дані (має розширення .ini)

"table" - Компонент, який представляє таблицю бази даних (має розширення .tbl)

"document" - Компонент, який представляє документ (має розширення .hlp)

У мові UML не визначені піктограми для перерахованих стереотипів, застосовувані на практиці піктограми компонентів показані на мал5-9.

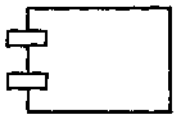


Рисунок 5 – Піктограма елемента що виконується.

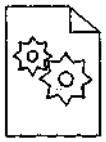


Рисунок 6 – Піктограма об'єктної Бібліотеки

Рисунок 7 – Піктограма документу з ісходним кодом

Рисунок 8– Піктограма таблиці даних БД

Рисунок 9 – Піктограма документу

Використання компонентних діаграм

Компонентні діаграми використовують для моделювання статичної вистави реалізації системи. Ця вистава підтримує керування конфігурацією системи, що складається з компонентів. Мається на увазі, що для одержання працюючої системи існують різні способи складання компонентів.

Компонентні діаграми показують відносини:

- періоду компіляції (серед текстових компонентів);
- періоду складання, линковки (серед об'єктних двійкових компонентів);
- періоду виконання (серед машинних компонентів).

Розглянемо типові варіанти застосування компонентних діаграм.

Моделювання програмного тексту системи

При розробці складних систем програмний текст (вихідний код) розкиданий по багатьом файлам вихідного коду. При використанні Java вихідний код зберігається в. java- файлах, при використанні C++ - у заголовних файлах (.h-файлах) і тілах (.cpp-файлах), при використанні Ada 95 - у специфікаціях (.ads-файлах) і реалізаціях (.adb-файлах).

Між файлами існують численні залежності компіляції. Якщо до цього додати, що в міру розробки народжуються нові версії файлів, то стає очевидною необхідність керування конфігурацією системи, візуалізації компіляційних залежностей.

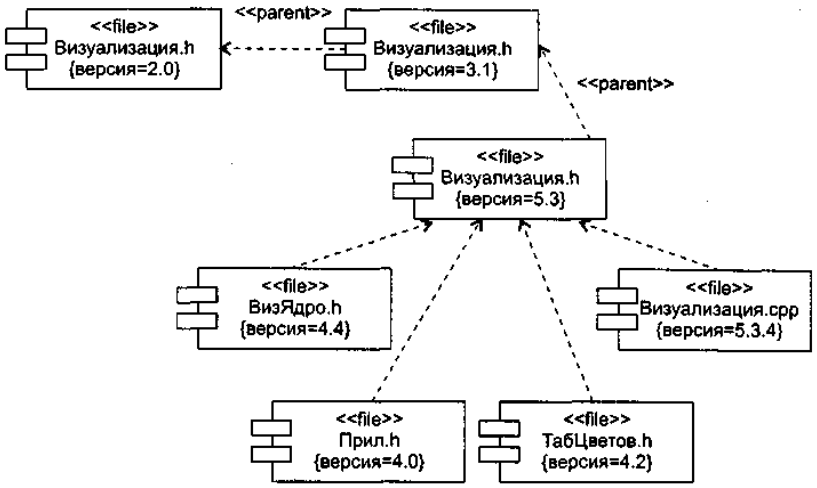


Рисунок 10 - Моделювання вихідного коду

Рисунок 11 - Моделювання вихідного коду з використанням піктограм

Як приклад на мал. 11 наведена компонентна діаграма, де зображені файли вихідного коду, використовувані для побудови бібліотеки Візуалізація.dll. Є чотири заголовні файли (Визуализация.h, ВизЯдро.h, Прил.h, ТабЦветов.h), які

представляють вихідний код для специфікації певних класів. Файл реалізації тут один (Візуалізація.cpp), він є реалізацією одного із заголовків. Відзначимо, що для кожного файлу явно зазначена його версія, причому для файлу Візуалізація.h показано три версії й історія їх появи. На мал. 11 повторюється та ж діаграма, але тут для позначення компонентів використані спеціальні піктограми.

Моделювання реалізації системи

Реалізація системи може включати велика кількість різноманітних компонентів:

- елементів, що виконуються;
- динамічних бібліотек;
- файлів даних;
- довідкових документів;
- файлів ініціалізації;
- файлів реєстрації;
- сценаріїв;
- файлів установки.

Моделювання цих компонентів, відносин між ними - важлива частина керування конфігурацією системи.

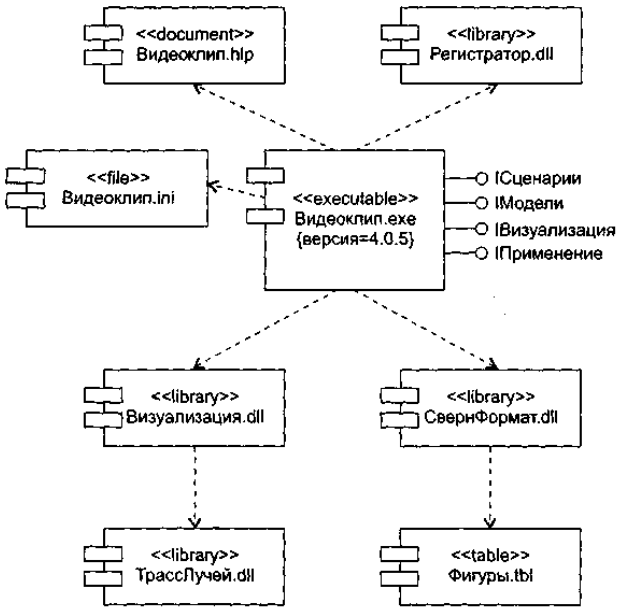


Рисунок 12- Моделювання реалізації системи

Наприклад, на мал. 12 показана частина реалізації системи, группируемая навколо елемента, що виконується, Відеокліп.exe. Тут зображено чотири

бібліотеки (Реєстратор.dll, Свернформат.dll, Візуалізація.dll, Трасслучей.dll), один документ (Відеокліп.hlp), один простий файл (Відеокліп.ini), а також таблиця бази даних (Фігури.tbl). У діаграмі зазначені відносини залежності, що існують між компонентами.

Для компонента, що виконується, Відеокліп.exe зазначений номер версії (за допомогою пгеговой величини), представлені його експортовані інтерфейси (Ісценарии, Івизуализация, Імодели, Іприменение). Ці інтерфейси утворюють API компонента "інтерфейс прикладного програмування).

На мал. 13 повторюється та ж діаграма, що моделює реалізацію, але тут для позначення компонентів використані спеціальні піктограми.

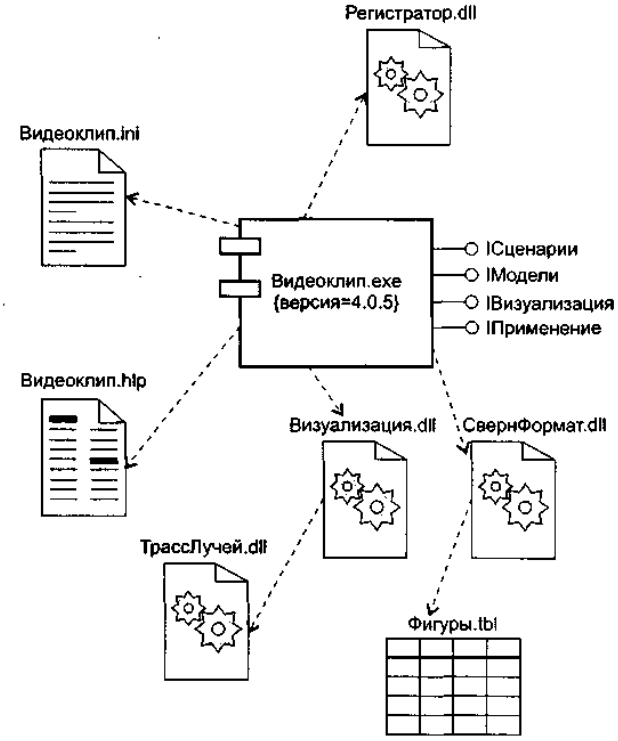


Рисунок 13 - Моделювання реалізації з використанням піктограм

Тема 2.1.10 Діаграми розміщення

Лекція №15

(2 години)

Мета: Розглянути основні поняття діаграми розміщення. Ознайомитись з графічною нотацією діаграми.

Структура заняття:

I. Організаційний момент:

1. Готовність групи до заняття;
2. Психоемоційний настрій;
3. Перевірка присутніх.

II. Актуалізація опорних знань студентів:

1. Повідомлення теми та мети заняття;
2. Відповіді та запитання.

III. Виклад нового матеріалу:

План:

1. Діаграма розміщення;
2. Вузол;
3. Використання діаграм розміщення.

IV. Підведення підсумків занять;

V. Домашнє завдання:

1. Ознайомитись з теоретичним матеріалом теми 2.1.10
2. Вивчити основні поняття лекції.
3. Відповісти на контрольні питання.

Контрольні питання:

1. Які вершини й ребра утворюють діаграму розміщення?
2. Чим відрізняється вузол від компонента?
3. Де можна використовувати й де не можна використовувати екземпляри компонентів?
4. Як застосовують діаграми розміщення?

Діаграма розміщення (розгортання) - друга із двох різновидів діаграм реалізації UML, що моделюють фізичні аспекти об'єктно-орієнтованих систем. Діаграма розміщення показує конфігурацію обробних вузлів у період роботи системи, а також компоненти, "живучі" у них.

Елементами діаграм розміщення є вузли, а також відносини залежності й асоціації. Як і інші діаграми, діаграми розміщення можуть включати примітки й обмеження. Крім того, діаграми розміщення можуть включати компоненти, кожний з яких повинен жити в деякому вузлі, а також містити пакети або підсистеми, використовувані для угруповання елементів моделі у великі

фрагменти. При необхідності візуалізації конкретного варіанта апаратної топології в діаграми розміщення можуть міститися об'єкти.

Вузли

Вузол - фізичний елемент, який існує в період роботи системи й представляє комп'ютерний ресурс, що має пам'ять, а можливо, і здатність обробки. Графічно вузол зображується як куб з іменем (рис. 1).

Рисунок 1 - Позначення вузла

Як і клас, вузол може мати додаткову секцію, що відображає розташовувані в ньому елементи (рис. 2).

Рисунок 2 - Розміщення компонентів у вузлі

Зрівняємо вузли з компонентами. Звичайно, у них є подібні характеристики:

- наявність імені;
- можливість бути вкладеним;
- наявність екземплярів.

Остання характеристика говорить про те, що на попередньому малюнку зображений тип Контролера. Зображення конкретного екземпляра, що належить цьому типу, презентовано на рис. 3.

Тепер обговоримо відмінності вузлів від компонентів. По-перше, вони належать до різних рівнів ієрархії у фізичній реалізації системи. Фізично система складається з вузлів, а вузли - з компонентів. По-друге, у кожного з них своє призначення. Компонент призначений для фізичного впакування й матеріалізації набору логічних елементів (класів і кооперацій). Вузол же є тем місцем, де фізично розміщаються компоненти, тобто відіграє роль "квартири" для компонентів.

Рисунок 3 - Екземпляр вузла

Відношення між вузлом і компонентами, які він розміщає, можна відобразити явно. Відношення залежності між вузлом Контролер і компонентами Введення.exe, Висновки.exe ілюструє рис. 4. Правда, найчастіше такі відносини не відображаються. Їх зручно представляти в окремій специфікації вузла.

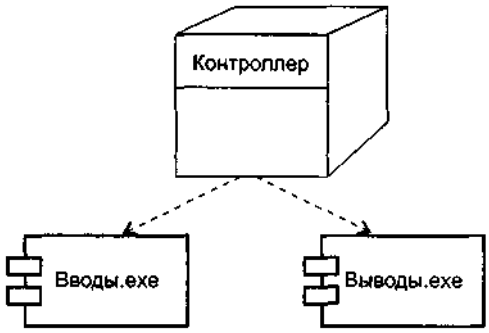


Рисунок 4 - Залежність вузла від компонентів

Угруповання набору об'єктів або компонентів, розташовуваних у вузлі, звичайно називають розповсюджуваним модулем.

Для вузла, як і для класу, можна задати властивості й операції. Наприклад, можна визначити властивості БыстродействиеПроцессора, ЕмкостьПамяти, а також операції Запустити, Виключити.

Використання діаграм розміщення

Діаграми розміщення використовують для моделювання статичної вистави того, як розміщається система. Ця вистава підтримує поширення, поставку й інсталяцію частин, що утворюють фізичну систему.

Графічно діаграма розміщення - це граф з вузлів (або екземплярів вузлів), з'єднаних асоціаціями, які показують існуючі комунікації. Екземпляри вузлів можуть містити екземпляри компонентів, що живуть або запускаються у вузлах. Екземпляри компонентів можуть містити об'єкти. Як показано на мал. 5, компоненти з'єднуються один з одним пунктирними стрілками залежностей (прямо або через інтерфейси).

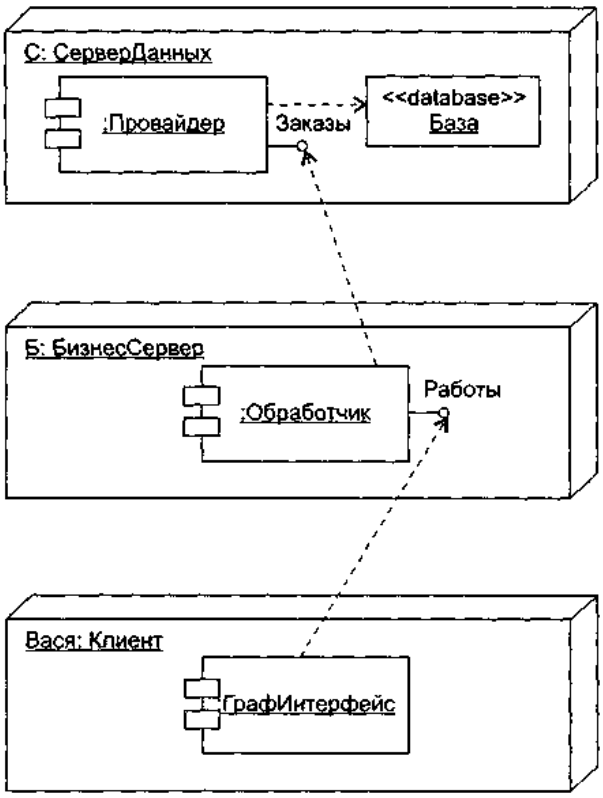


Рисунок 5 - Моделювання розміщення компонентів

На цій діаграмі зображена типова трехуровневая система:

- рівень бази даних реалізований екземпляром С вузла СервердДанных;
- рівень бізнес-логіки представлений екземпляром Б вузла БизнесСервер;
- рівень графічного інтерфейсу користувача утворений екземпляром Вася вузла Клієнт.

У вузлі сервера даних показане розміщення анонімного екземпляра компонента Провайдер і об'єкта База зі стереотипом <<database>>. Вузол бізнес-сервера містить анонімний екземпляр компонента Оброблювач, а вузол клієнта - анонімний екземпляр компонента ГрафИнтерфейс. Крім того, тут явно відображені інтерфейси компонентів Провайдер і Оброблювач, що мають, відповідно, імена Замовлення й Роботи.

Як презентовано на мал. 6, переміщення компонентів від вузла до вузла (або об'єктів від компонента до компонента) відзначається стереотипом <<becomes>> на відношенні залежності. У цьому випадку вважають, що компонент (об'єкт) резидентен у вузлі (компоненті) тільки в межах деякого кванта часу. На малюнку бачимо, що можливість міграції надана об'єктам X і Y.

Іноді корисно визначити фізичний розподіл компонентів по процесорах і іншим обладнанням системи. Є три способи моделювання розподілу:

- графічно розподіл не показувати, а документувати його в текстових специфікаціях вузлів;
- з'єднувати кожний вузол з розташовуваними компонентами відносинами залежності;
- у додатковій секції вузла вказувати список розташовуваних компонентів.

Діаграма розміщення, що ілюструє третій спосіб моделювання, показана на мал. 6.

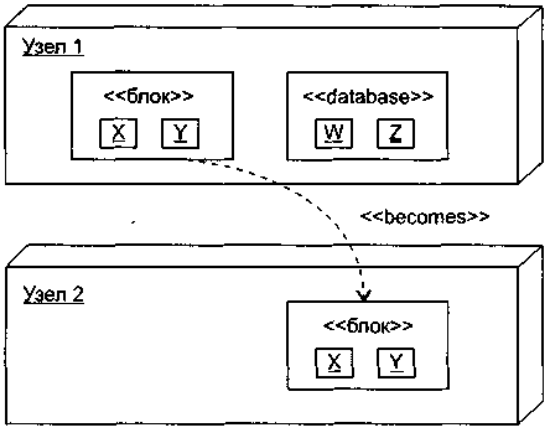


Рисунок 6 - Моделювання переміщення компонентів і об'єктів

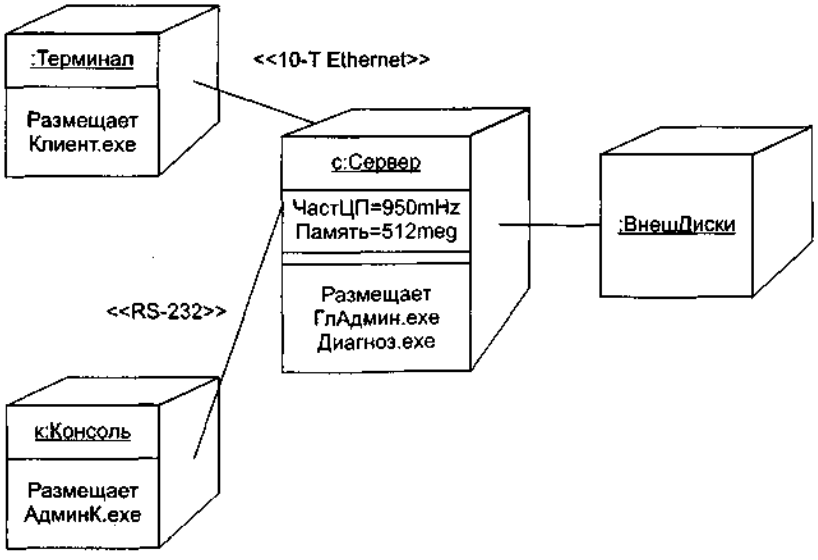


Рисунок 7 - Розподіл компонентів у системі

На малюнку показано два анонімні екземпляри вузлів (:ВНЕШДИСКИ, :Термінал) і два екземпляри вузлів з іменем (із для Сервера й до для Консолі). Кожний процесор намальований з додатковою секцією, у якій показані розміщені

компоненти. В екземплярі Сервера, крім того, відображені його властивості (ЧАСТЦП, Пам'ять) і їх значення.

За допомогою стереотипів задані характеристики фізичних з'єднань між процесорами: одне з них визначене як Ethernet- З'єднання, інше - як послідовне Rs-232- З'єднання.

Список використаних джерел

1. Крег Ларман «Книга Застосування UML 2.0 і шаблонів проєктування. 3-тє вид», Диалектика, 736 с. ISBN - 978-5-907144-36-1
2. Martin Fowler «UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition Addison-Wesley Professional, 2016 p., 208 c.
3. Martin Fowler «UML Distilled: A Brief Guide to the Standard Object Modeling Language, 2 Edition Addison-Wesley Professional, 2013 p., 192 c.
4. http://www.znannya.org – електноний ресурс.
5. Шенталер Ф. "Бізнес-процеси. Мови моделювання, методи, інструменти", Альпіна Паблішер, 2019 р., 264 с.