МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет біоресурсів і природокористування України

Методичні вказівки до виконання лабораторних робіт з дисципліни

«Об'єктне моделювання та проектування складних систем»

для студентів ОС «Магістр»,
що навчаються за спеціальністю
«Інформаційні управляючі системи і технології»

Київ 2024

УДК 004.94
ББК 32.973-018.2

Укладач: к.е.н., Ніколаєнко Д.В.

Рецензенти: к.т.н., доцент Голуб Б.Л., д.е.н., доцент Кравченко В.М.

Затверджено навчально-методичною комісією факультету інформаційних технологій Національним університетом біоресурсів і природокористування України (протокол №__ від «__» _____ 2024 р.)

Методичні вказівки до виконання лабораторних робіт з дисципліни «Об'єктне моделювання та проектування складних систем»/ уклад.: к.е.н. Ніколаєнко Д.В. – К.:НУБіП, 2024. – 37 с.

Містить загальні рекомендації до виконання лабораторних робіт. Для студентів, що навчаються за ОПП «Інформаційні управляючі системи і технології» ОС «Магістр»

ЗМІСТ

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ

Для здачі лабораторної роботи студенту потрібно ознайомитися з теоретичним матеріалом, що представлений в лекції, методичними вказівками до відповідної роботи, підготувати звіт у форматі “.doc” або “.pdf” та завантажити його в ЕНК, захистити роботу під час проведення заняття за розкладом. Лабораторна робота вважається зданою , якщо за нею студент отримав не менше, ніж 60% зазначеної кількості балів у лабораторній роботі.

Звіт повинен відповідати такій структурі:

- титульна сторінка, де зазначена тема лабораторної роботи та хто її виконав;
- формулювання завдання;
- хід виконання роботи.

ВСТУП

Метою викладання дисципліни “Об'єктне моделювання та проектування складних систем” є набуття студентами знань з об’єктно-орієнтованого підходу щодо аналізу та проектування інформаційних систем.

У результаті вивчення дисципліни “Об'єктне моделювання та проектування складних систем” студенти повинні мати знання з питань:

- оцінювання складності програмних систем;
- декомпозиції складних систем;
- концептуальних парадигм об’єктно-орієнтованого підходу щодо проектування інформаційних систем;
- формування структур класів і об’єктів інформаційної системи;

оволодіти практичними навичками:

- проведення об’єктно-орієнтованого аналізу інформаційної системи;
- проведення об’єктно-орієнтованого проектування інформаційної системи;
- розробки структури класів програмної системи;
- розробки структури об’єктів програмної системи.

Матеріал, який викладається у цій дисципліні, використовується студентами для написання дипломної роботи магістра.

ЛАБОРАТОРНА РОБОТА №1

Тема
Діаграма класів

Мета роботи визначити класи інформаційної системи та побудувати діаграму класів в режими «ескізу».

Теоретичні вказівки

Для виділення класів інформаційної системи потрібно керуватись підходом абстрагування.

Абстрагування – це спосіб бачення всього навкруги себе, спосіб мислення. Це спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми.

Приклади визначення об’єктів з використанням підходу абстрагування:

- Собака – це об’єкт
- Машина – це об’єкт
- …
- Користувач – це об’єкт
- Заказ – це об’єкт
- Елемент заказу – це об’єкт
- Продукт – це об’єкт
- …

• Касовий чек – це об’єкт

• …

• Всесвіт – це об’єкт

Класом називають множину об’єктів, що мають спільну структуру, властивості та поведінку. Ключові характеристики класу, які потрібно використовувати для визначення класу:

• Клас – це абстрактна сутність.

• Клас – не існує.

• Клас – це лише уява розробника.

Клас визначає абстрактні характеристики деякої сутності:

• Атрибути (властивості)

• Методи (поведінка)

• Події (повідомлення)

Об’єкт – це окремий екземпляр класу який створюється після запуску програми та ініціалізації полів класу.

Існують три способи використання мови UML:

Режим ескізу. Використовується як під час прямої розробки (forward - engineering) коли діаграми будуються до написання коду, так і під час зворотної розробки (reverse - engineering).

Режим проектування. Використовується як під час прямої розробки (forward - engineering) коли

діаграми будуються до написання коду, так і під час зворотної розробки (reverse - engineering).

Режим програмування. Чим довше дизайнер працює з UML тим точніші та детальніші моделі він створює і поступово процес програмування перетворюється на більш механічну роботу.

В решті решт дизайн стає настільки повним та описує всю систему, таким чином UML перетворюється на мову програмування.

Приклад виконання роботи

Для виконання лабораторної роботи №1 потрібно визначити класи системи відповідно до обраної теми та відобразити їх на діаграмі класів використовуючи режим ескізу.

Даний режим має на увазі побудову загальної схеми класів з визначенням:

- Переліку класів та їх назви;
- Загальних властивостей та поведінки;
- Зв’язків між класами.

На даному етапі діаграма не повинна мати жодних деталей про класи, а саме:

- Всіх можливих властивостей та операцій;
- Визначення атрибутів властивостей чи поведінки, як то видимість, кратність, тип, значення за замовченням, тощо.

При визначенні класів слід користуватись наступними підказками:

- Назва має бути унікальною в межах моделі.
- Клас – це не таблиця в базі даних
- Клас – це сутність яка має поведінку (в більшості випадків)
- Клас – це сутність, яка поєднує інші об’єкти за спільними властивостями та поведінкою
- На початкових етапах аналізу достатньо вказувати лише назву класу
- З ходом доопрацювання моделі класи наповнюються атрибутами та методами

Завдання

1. Розробіть діаграму класів відповідно до обраного завдання використовуючи будь-який інструментарій створення діаграм UML. Діаграма має бути побудована в режимі ескізу (без технічних деталей) для розуміння загальної архітектури системи.
2. Надайте розгорнутий опис побудованої діаграми класів.
3. Надайте визначення зв'язків які можуть використовуватись на діаграмі класів.
4. Підготуйте звіт у вигляді документу

ЛАБОРАТОРНА РОБОТА №2

Тема

Діаграма прецедентів

Мета роботи Отримання діаграм прецедентів для вибраного проєкту.

Теоретичні вказівки

Діаграма прецедентів – відображає можливості системи (або її частини) та варіанти взаємодії актора з системою. Вона відображає функціональність системи в межах окремого модулю або бізнес-процесу.

Прецедент – це функціональність системи (підсистеми, модуля).

Це може бути:

- Подача заявки;
- Зміна пін коду;
- Завантаження даних;
- Перегляд результату тощо.

Актор – це користувач системи в межах окремого прецеденту.

Це може бути:

- Користувач
- Адміністратор
- Менеджер
- Інша система

Типи **зв'язків**:

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

Відношення асоціації відображає простий зв’язок актора з прецедентом

Відношення включення відображає зв’язок двох прецедентів коли один є частиною другого

Має сенс використовувати при декомпозиції прецедента на декілька

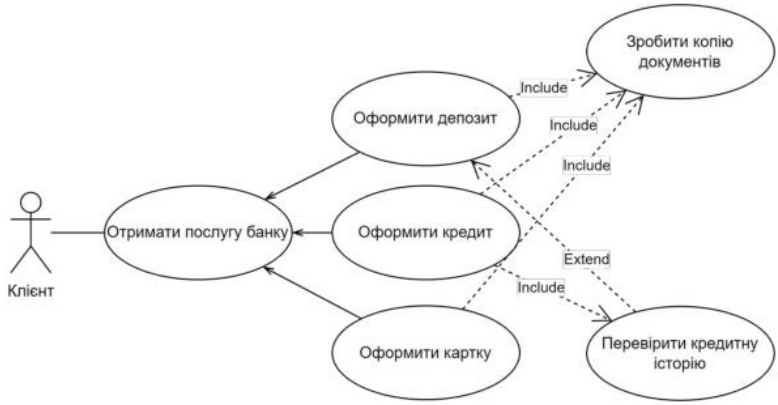
Направлене від прецедента А до прецедента Б і показує, що прецедент А включає в себе виконання прецеденту Б

Відношення розширення Відображає зв’язок двох прецедентів коли один доповнює інший

Має сенс використовувати при наявності альтернатив

Направлене від прецедента А до прецедента Б і показує, що прецедент А доповнює функціонал прецедента Б

Відношення узагальнення відображає зв'язок двох прецедентів коли один є спеціалізацією іншого



Має сенс використовувати при наявності складних бізнес-процесів

Направлене від прецедента А до прецедента Б і показує, що прецедент А є спеціалізацією (спадкуванням) прецедента Б

Специфікація прецедента.

Специфікація прецедента – це детальний опис кожної окремої функціональності системи у наступному вигляді:

Element	Detail
Name	The use case name. Typically the name is of the format <action> + <object>.
ID	An identifier that is unique to each Use Case.
Description	A brief description that states what the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish a task. Actors are identified by role name.
Organization Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often this Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Preconditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Postconditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or Alternate Courses are followed. Some exceptions may result in failure to meet the Postconditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Course	Alternate paths through the system. AC1:<condition for the alternate to be called> 1. Step 1 2. Step 2 AC2:<condition for the alternate to be called> 1. Step 1
Exception Courses	Exception handling by the system EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2:<condition for the exception to be called> 1. Step 1

Найбільш поширеною та використовуваною є наступна структура прецеденту:

- Номер та назва
- Мета або опис
- Передумова
- Тригер
- Основний сценарій
- Альтернативні сценарії

Приклад специфікації прецеденту:

NVUC001 Оформити кредит

Цей юзкейс описує процес відкриття кредиту клієнтом банку

Передумова: клієнт має намір відкрити кредит та є зареєстрованим клієнтом банку

Тригер: Клієнт відкриває форму «оформити кредит»

Основний сценарій:

1. Клієнт вносить особисті дані
2. Система перевіряє кредитну історію (A1)
3. Система видає запит на завантаження копії документів
4. …
5. Система створює кредитний рахунок
6. Система видає повідомлення про успішне відкриття кредиту

Альтернативний сценарій (A1)

1. Система відхиляє в укладанні кредиту
2. Система видає повідомлення про відмову

Завдання

1. Побудуйте діаграму(и) прецедентів до обраної системи.
2. Складіть специфікацію (текстовий опис Use-case) побудованих прецедентів (2-3 прецеденти).

3. Надайте визначення зв'язків які можуть використовуватись на діаграмі прецедентів.
4. Підготуйте звіт.

ЛАБОРАТОРНА РОБОТА №3

Тема
Діаграми послідовностей

Мета роботи побудова діаграм послідовностей для визначених прецедентів системи.

Теоретичні вказівки

Діаграма послідовності – опис взаємодії груп об’єктів в різних умовах їхньої поведінки.

На діаграмі послідовності зображуються виключно ті об'єкти, які безпосередньо беруть участь у взаємодії, і не показуються можливі статичні асоціації з іншими об'єктами.

Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об'єктів у часі.

Діаграма послідовностей показує саме відносини між об’єктами (учасниками), а не між класами.

На діаграмі послідовностей не обов’язково представляються лише об’єкти, тому можна називати їх учасники

Об’єкт (учасник)

Об’єкт – це створений екземпляр класу.

Для заощадження ресурсів системи зовсім не обов'язково створювати всі об'єкти в початковий момент часу. У цьому разі прямокутник такого об'єкта

зображується не у верхній частині діаграми послідовності, а в тій її частині, яка відповідає моменту створення об'єкта

Лінія життя об’єкта

Лінія життя об'єкта (object lifeline) зображується пунктирною вертикальною лінією, асоційованою з єдиним об'єктом на діаграмі послідовності.

Лінія життя слугує для позначення періоду часу, протягом якого об'єкт існує в системі і, отже, може потенційно брати участь у всіх її взаємодіях.

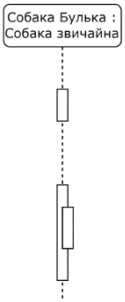
Якщо об'єкт існує в системі постійно, то і його лінія життя має продовжуватися по всій площині діаграми.

Окремі об'єкти, виконавши свою роль у системі, можуть бути знищені (зруйновані), щоб звільнити ресурси, які вони займають.

Для позначення моменту знищення об'єкта в мові UML використовується спеціальний символ у формі латинської літери "X"

Фокус управління

Об'єкти можуть перебувати в активному стані, безпосередньо виконуючи певні дії або в стані пасивного очікування повідомлень від інших об'єктів.



Щоб явно виокремити подібну активність об'єктів, у мові UML застосовують спеціальне поняття, що отримало назву фокусу керування (focus of control).

Фокус управління зображується у формі витягнутого вузького прямокутника, верхня сторона якого позначає початок отримання фокуса управління об'єкта (початок активності), а її нижня сторона - закінчення фокуса управління (закінчення активності).

Періоди активності об'єкта можуть чергуватися з періодами його пасивності або очікування. У цьому разі в такого об'єкта є кілька фокусів управління

Повідомлення

Повідомлення (message) являє собою закінчений фрагмент інформації, який надсилається одним об'єктом іншому.

Повідомлення не тільки передають деяку інформацію, а й вимагають або припускають від об'єкта, що приймає, виконання очікуваних дій.

Повідомлення можуть ініціювати виконання операцій об'єктом відповідного класу, а параметри цих операцій передаються разом із повідомленням.

На діаграмі послідовності всі повідомлення впорядковані за часом свого виникнення.

Зазвичай над повідомленнями вказується яке саме повідомлення відправляється

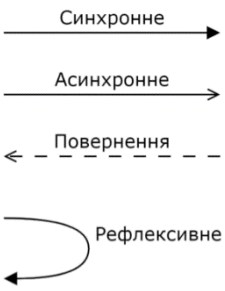
• getPrice

• getQuantity(quantity: number)

• calculateBasePrice

UML не має засобів відображення типу даних тому використовується параметр в повідомленні.

Типи повідомлень



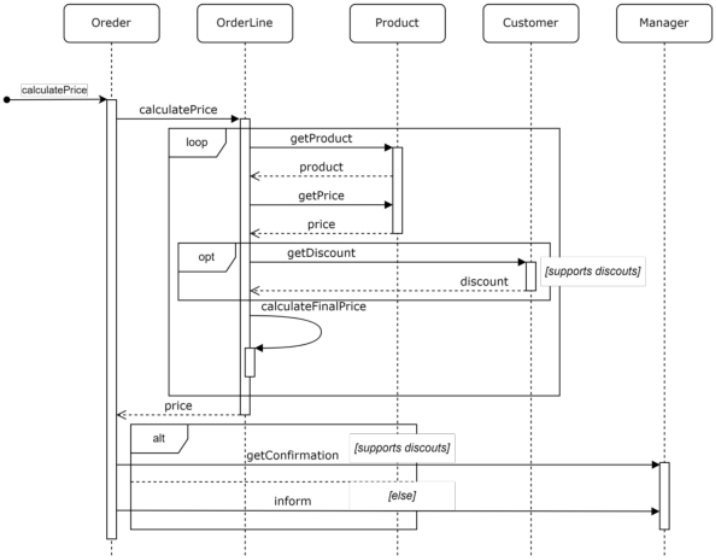
Синхронне повідомлення є найпоширенішим і використовується для виклику процедур, виконання операцій або позначення окремих вкладених потоків управління.

Асинхронне повідомлення викликає дію іншого об’єкта, проте поточний об’єкт може продовжувати роботу та не чекати на відповідь

Повернення результатів роботи процедури

Рефлексивне - повідомлення самому собі.

Фрейми на діаграмі послідовностей



loop – цикл

alt – альтернативні фрагменти

alt – необов’язковий фрагмент

par – паралельний фрагмент

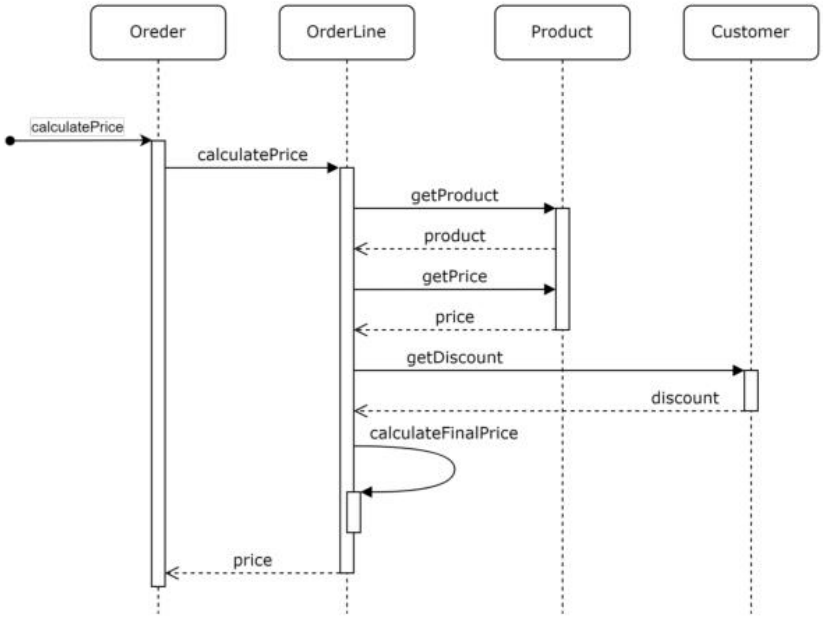
region – критична область, може мати тільки один потік

neg – невірна взаємодія

ref – посилання на взаємодію (інша діаграма)

sd – інша діаграма послідовностей

Приклад виконання роботи



Завдання

1. Побудуйте діаграму(и) послідовностей до обраної системи.
2. Опишіть обрані об'єкти та побудовані послідовності.
3. Підготуйте звіт.

ЛАБОРАТОРНА РОБОТА №4

Тема

Діаграма класів (детальна)

Мета роботи побудувати детальну діаграму класів в режимі «проектування».

Теоретичні вказівки

Атрибути класів

Атрибут описує властивості у вигляді тексту в першому блоці класу.

Повна форма виглядає так:

```
Видимість назва: тип [кратність] = значення за замовченням {властивості}
```

Наприклад:

```
- username: String [1] = “Noname” {readOnly}
```

Видимість атрибутів

Характеризує доступність цього атрибуту з інших класів системи.

«+» - відкритий (public) – такий атрибут доступний з будь-якого іншого класу, пакета або модуля

«-» - закритий (private) – такий атрибут доступний тільки з цього класу.

«#» - захищений (protected) – такий атрибут доступний тільки з підкласів цього суперкласу (при використанні спадкування).

Видимість атрибуту не є обов'язковою і може бути пропущена.

Кратність атрибутів

Характеризує загальну кількість конкретних атрибутів даного типу, що входять до складу окремого класу.

[0..1] - кратність атрибута може набувати значення 0 або 1. При цьому 0 означає відсутність значення для даного атрибута.

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

[1.:*] - кратність атрибута може приймати будь-яке позитивне ціле значення більше або рівне 1.

За замовченням приймає значення [1]

Тип атрибутів

Тип атрибута являє собою вираз, семантика якого визначається мовою специфікації відповідної моделі. У найпростішому випадку тип атрибута вказується рядком тексту, що має осмислене значення в межах пакета або моделі, до яких належить розглянутий клас.

Наприклад:

```
: Color
: String
: Boolean
```

Тип атрибуту не є обов’язковим і може бути пропущене.

Значення за замовченням

Початкове значення слугує для завдання деякого початкового значення для відповідного атрибута в момент створення окремого екземпляра класу.

Тут необхідно дотримуватися правила належності значення типу конкретного атрибута.

Наприклад:

```
- color: Color = (255, 0, 0)
- name[1]: String = “Іван Іванович”
- visible: Boolean = True
```

Якщо початкове значення не вказано, то значення відповідного атрибута не визначено 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.

Операції

Операція описує поведінку класу і записується у другому блоці класу.

Повна форма виглядає так:

`видимість назва (список параметрів): тип {властивості}`

Наприклад:

`+ checkBalanceOn (date: Date): Currency`

Список параметрів операції

Визначає перелік параметрів що передаються операції або повертаються Повна форма виглядає так:

`напрямок назва: тип = значення за замовченням`

Наприклад:

`in date: DateTime`

`inOut itemNumber: Integer`

- `in` - вхідний параметр
- `out` - вихідний параметр
- `inOut` - вхідний та вихідний параметр

За замовченням `in`

Видимість операції

Характеризує доступність цього операції з інших класів системи

«+» - відкритий (public) – така операція доступна з будь-якого іншого класу, пакета або модуля

«-» - закритий (private) – така операція доступна тільки з цього класу

«#» - захищений (protected) – така операція доступна тільки з підкласів цього суперкласу (при використанні спадкування).

Видимість не є обов’язковою і може бути пропущена.

Операції та Методи

Часто операції та методи вважають одним і тим самим, проте слід вміти їх розрізняти.

Операція представляє собою те, що викликається об’єктом під час створення процедури

Метод – тіло процедури.

Їх має сенс розлічати в межах терміну поліморфізм.

Якщо є суперклас з якоюсь поведінкою та 2 підкласи, що перевизначають цю поведінку, то це одна операція та два методи, що її реалізують.

Приклади операцій

`+створити()`

може позначати абстрактну операцію зі створення окремого об'єкта класу, яка є загальнодоступною і не містить формальних параметрів. Ця операція не повертає жодного значення після свого виконання.

`+намалювати(форма: Багатокутник = прямокутник, колір_заливки: Color = (0, 0, 255))`

може позначати операцію по зображенню на екрані монітора прямокутної області синього кольору, якщо не вказуються інші значення як аргументи даної операції.

`запросити_рахунок_клієнта(номер_рахунку:integer = 123456):Currency`

позначає операцію зі встановлення наявності коштів на поточному рахунку клієнта банку. При цьому аргументом цієї операції є номер рахунку клієнта, який записується у вигляді цілого числа (наприклад, "123456"). Результатом виконання цієї операції є деяке число, записане в прийнятому грошовому форматі (наприклад, $1,500.00).

`видати_повідомлення():{"Помилка ділення на нуль"}`

сенс цієї операції не потребує пояснення, оскільки воно міститься в рядку-властивості операції. Це повідомлення може з'явитися на екрані монітора в разі спроби ділення числа на нуль, що неприпустимо.

Відношення між класами

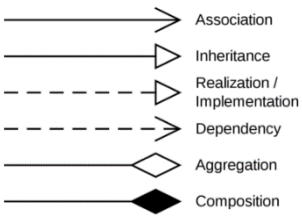
Відношення між класами показує що один клас звертається до іншого:

- Викликає метод;

- Встановлює властивість;
- Читає властивість;
- Спадкує;
- Реалізує функціональність, тощо.

Існують наступні відношення:

- Відношення асоціації (association)
- Відношення залежності (dependency)
- Відношення агрегації (aggregation)
- Відношення композиції (composition)
- Відношення узагальнення (generalization) або Відношення спадкування (inheritance)
- Відношення реалізації (realization)



Відношення асоціації

Показує, що об'єкти однієї сутності (класу) пов'язані з об'єктами іншої сутності.

Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносин.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою

одного примірника асоціації. Це число називається кратністю (Multiplicity)

Відношення залежності

Відношення залежності використовується в такій ситуації, коли деяка зміна одного елемента моделі може вимагати зміни іншого залежного від нього елемента моделі.

Позначається у вигляді стрілки від головного до залежного класу.

Відношення агрегації

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

За потреби можна вказувати кратність агрегації.

Агрегація не накладає жорстких умов на термін існування об'єктів. Наприклад, «частини» можуть існувати тоді, коли «ціле» зникає.

Графічно агрегація представлена порожнім ромбом на блоці «цілого», і лінією, яка проведена від цього ромба до класу, що міститься в ньому («частин»).

Відношення композиції

Більш строгий варіант агрегації

Композиція має жорстку залежність часу існування екземплярів класу контейнера та

екземплярів класів що містяться в ньому. Якщо контейнер буде знищений, то весь його вміст буде також знищено.

Графічно представляється як і агрегація, але з зафарбованим ромбиком.

Відношення узагальнення (спадкування)

Це означає, що один з двох споріднених класів (підклас, підтип, нащадок) вважається спеціалізованою формою іншого (суперклас, супертип, батько), а суперклас вважається узагальненням підкласу.

Графічне представлення узагальнення – це порожнистий трикутник на кінці лінії (або дерева ліній) суперкласу, який з'єднує його з одним або декількома підкласами.

Відношення узагальнення також відоме як спадкування (наслідування) або відношення «є».

Відношення реалізації

Зв'язок між двома елементами моделі, в якому один елемент моделі (клієнт) реалізує (впроваджує або виконує) поведінку, яку визначає інший елемент моделі (постачальник).

Графічне представлення – порожнистий трикутник на «інтерфейсному» кінці пунктирної лінії.

Реалізації можуть бути показані тільки на діаграмах класів або компонентів.

Зв'язок реалізації між класами/компонентами та інтерфейсами показує, що клас/компонент реалізує операції.

Завдання

1. Розробіть діаграму класів відповідно до обраного завдання використовуючи будь-який інструментарій створення діаграм UML. діаграма має містити коректні зав'язки, деталі про властивості та операції (тип видимості, тип, параметри, тощо).
2. Надайте розгорнутий опис побудованої діаграми класів.
3. Надайте визначення зв'язків які було використано на діаграмі класів.
4. Підготуйте звіт у вигляді документу.

ЛАБОРАТОРНА РОБОТА №5

Тема

Діаграма діяльності

Мета роботи Отримати навички у побудові діаграми діяльності.

Теоретичні вказівки

Діаграма діяльності

Є базовою з точки зору подання поведінки. Діаграми діяльності мають ряд переваг для користувачів.

Демонструє логіку алгоритму.

Описує кроки, що виконуються у варіанті використання UML.

Ілюструє бізнес-процес або робочий процес між користувачами та системою.

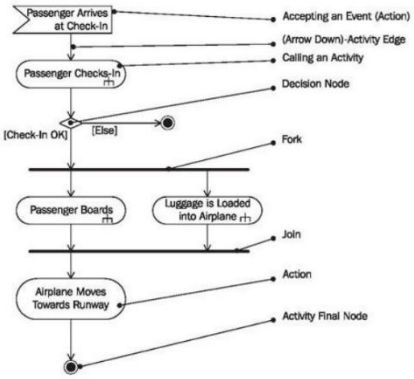
Спрощує та покращує будь-який процес шляхом пояснення складних варіантів використання.

Моделює елементи архітектури програмного забезпечення, такі як метод, функція та операція.

Складові діаграми діяльності

- Початковий вузол або подія
- Потік/ребро

- Рішення
- Розгалуження
- Об’єднання
- Операція
- Вкладена операція
- Фінальний вузол

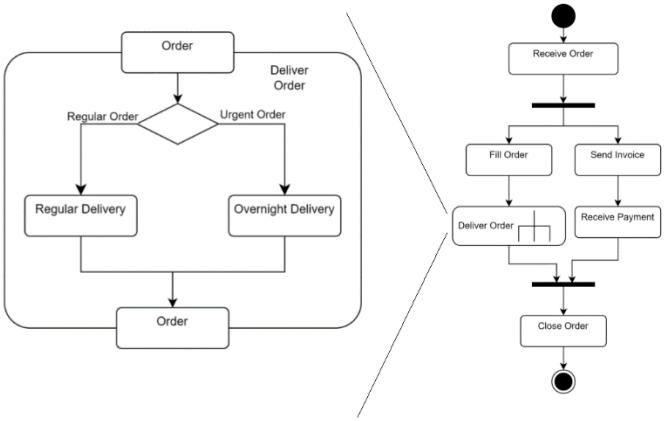


Розділи

Використовуються для відображення інформації про учасника діяльності використовуються розділи (swim lane).

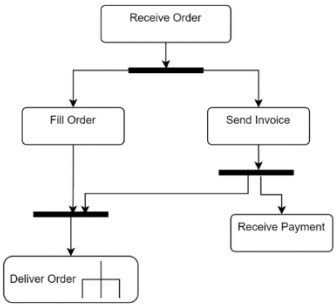
Декомпозиція операцій

Для спрощення діаграми використовується принцип декомпозиції операцій, коли окремі операції поєднуються в одну та подаються на діаграмі діяльності у вигляді вкладеної операції.

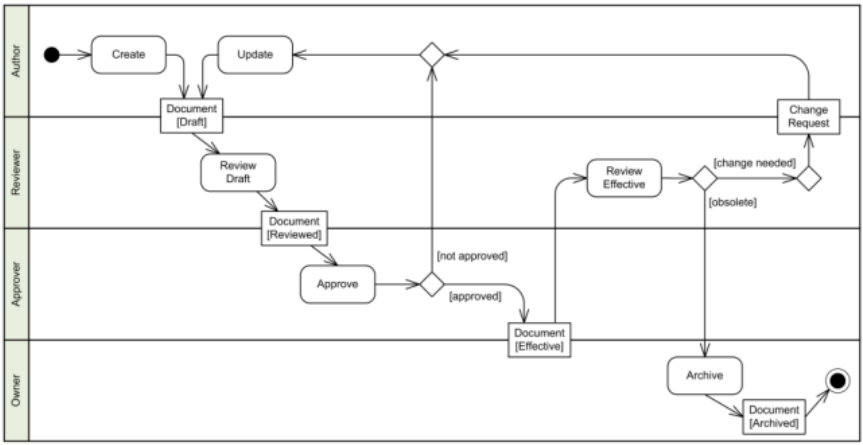


Синхронізація

Діаграма діяльності чудово підходить для відображення синхронізації потоків та операцій коли паралельні операції впливають одна на одну та потребують узгодження.



Приклад виконання роботи



Завдання

1. Побудуйте діаграму(и) діяльності до обраної системи.
2. Складіть детальний опис побудованої діаграми діяльності.
3. Підготуйте звіт.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Design Patterns [Електронний ресурс] – Режим доступу до сайту: https://refactoring.guru/design-patterns