ПРАКТИЧНЕ ЗАНЯТТЯ № 02

ТЕХНОЛОГІЯ СТВОРЕННЯ ПРОГРАМНИХ ПРОДУКТІВ

Тема: **ОБ'ЄКТНО-ОРІЄНТОВАНИЙ АНАЛІЗ ТА ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. ДІАГРАМА КЛАСІВ.**

Мета: дослідження предметної області, вивчення процесу побудови діаграми класів за допомогою інструмента об'єктного моделювання StarUML.

Програмне забезпечення: ОС Windows/Linux, StarUML (https://staruml.io/).

Теоретичний базис:

1. Загальна концепція

Діаграма класів (class diagram) використовуються для моделювання статичного виду системи з точки зору проектування. Діаграма класів - діаграма, на якій показано безліч класів, інтерфейсів, кооперацій і відносин між ними. Використовується в наступних цілях:

- для моделювання словника системи: передбачає прийняття рішення про те, які абстракції є частиною системи, а які – ні. За допомогою діаграм класів можна визначити ці абстракції і їх обов'язки;
- для моделювання простих кооперацій. Кооперація – це спільнота класів, інтерфейсів і інших елементів, що працюють спільно для забезпечення деякої кооперативного поведінки;
- для моделювання логічної схеми бази даних.

Згідно Мартіну Фаулеру існують три різні точки зору на побудову діаграм класів або будь-який іншій моделі:

- концептуальна точка зору – діаграми класів служать для формування та представлення понять досліджуваної предметної області. Ці поняття будуть відповідати класам, що їх реалізують, але пряма відповідність може бути нечіткою або відсутньою. Концептуальна модель може мати слабке відношення або взагалі не мати ніякого відношення до програмного забезпечення, що її реалізує, тому її можна розглядати без прив'язки до якоїсь конкретної мови програмування;
- точка зору специфікації – розглядається програмна система, при цьому розглядаються лише її інтерфейси, але не реалізація;
- точка зору реалізації – класи діаграми відповідають реальним класам програмної системи.

2. Основні графічні нотації діаграм класів на мові UML 2

Діаграма класів UML — це графічна нотація, яка використовується для побудови та візуалізації об’єктно-орієнтованих систем. Діаграма класів в уніфікованій мові моделювання (UML) — це тип статичної структурної діаграми, яка описує структуру системи, показуючи системні:

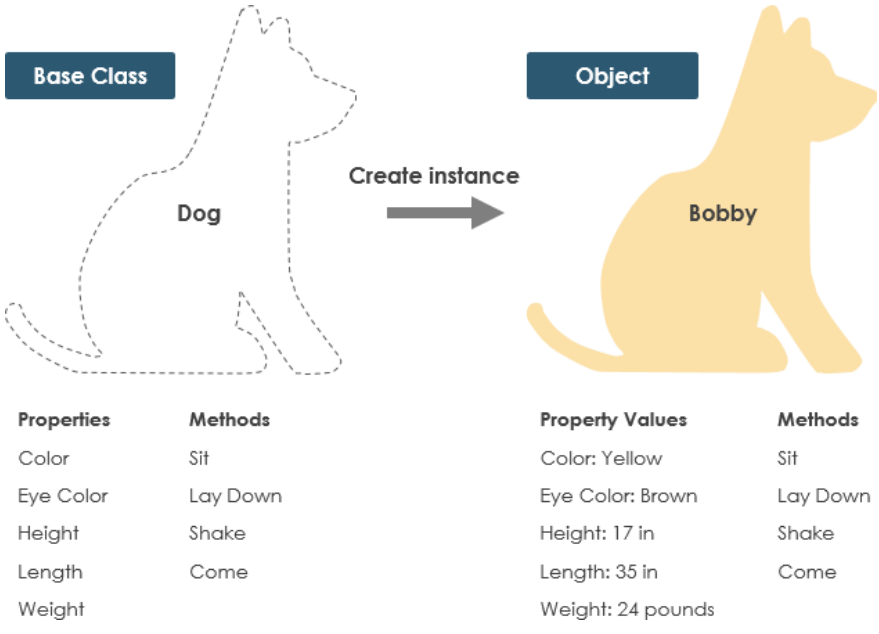
- класи,
- їхні атрибути,
- операції (або методи),
- і зв’язки між об’єктами.

Що таке клас?

Клас — це схема об’єкта. Об’єкти та класи йдуть рука об руку. Ми не можемо говорити про одне, не говорячи про інше. І вся суть об’єктно-орієнтованого проектування полягає не в об’єктах, а в класах, тому що ми використовуємо класи для створення об’єктів. Отже, клас описує, чим буде об’єкт, але це не сам об’єкт.

Насправді класи описують тип об’єктів, тоді як об’єкти є придатними для використання екземплярами класів. Кожен об’єкт був створений з однакового набору креслень і тому містить однакові компоненти (властивості та методи). Стандартне значення полягає в тому, що об’єкт є екземпляром класу та об’єкта. Об’єкти мають стани та поведінку.

У собаки є стани - колір, ім'я, порода, а також поведінка - виляння, гавкіт, їжа. Об’єкт – це екземпляр класу.



Нотація класу UML

Клас представляє концепцію, яка інкапсулює стан (атрибути) і поведінку (операції). Кожен атрибут має тип. Кожна операція має підпис . *Назва класу є єдиною обов'язковою інформацією* .

Назва класу:

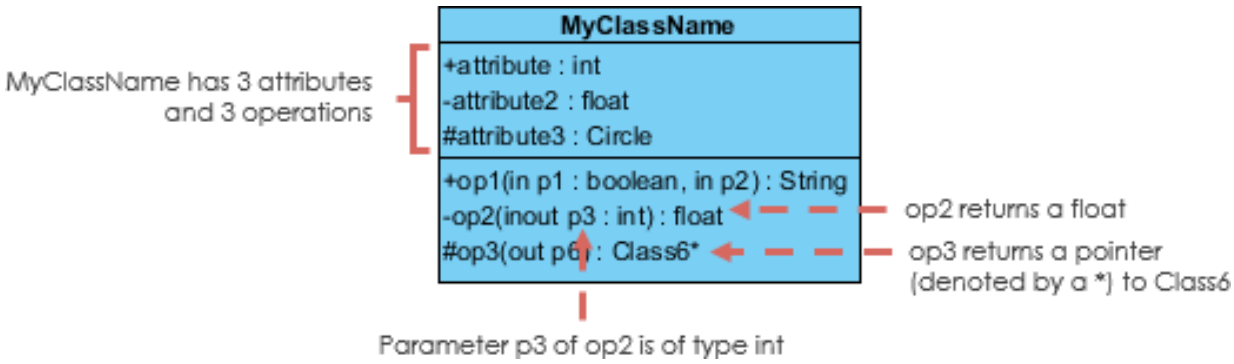
- Ім'я класу з'являється в першому розділі.

Атрибути класу:

- Атрибути показані у другому розділі.
- Тип атрибута вказується після двокрапки.
- Атрибути відображаються на змінних-членах (членах даних) у коді.

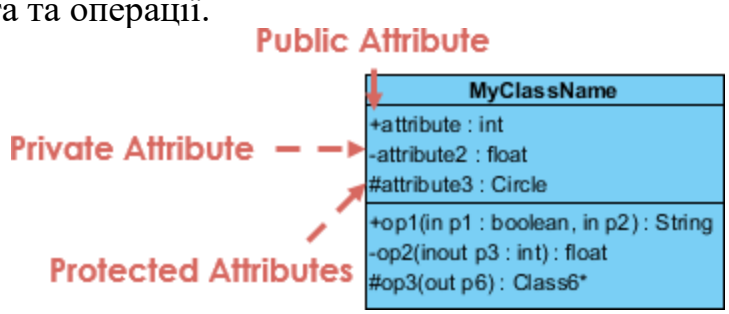
Операції класу (методи):

- Операції показані в третьому розділі. Це послуги, які надає клас.
- Тип повернення методу відображається після двокрапки в кінці підпису методу.
- Повернений тип параметрів методу відображається після двокрапки після назви параметра. Операції відображаються на методи класу в коді
-



Видимість класу

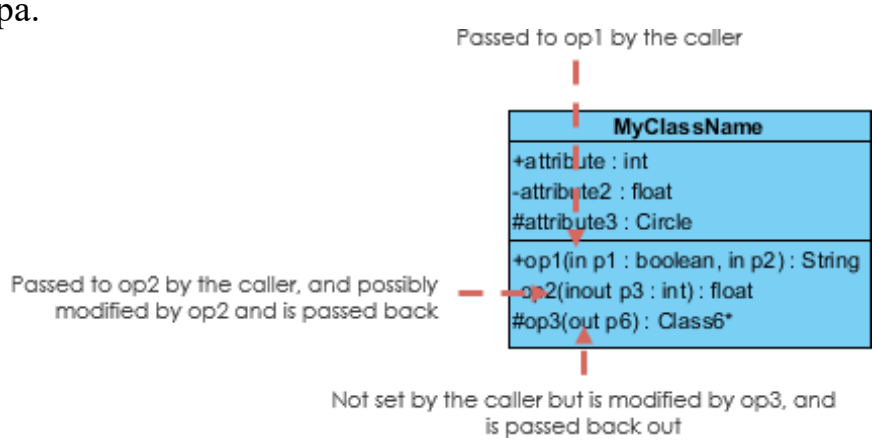
Символи +, - і # перед назвою атрибута та операції в класі позначають видимість атрибута та операції.



- \+ позначає публічні атрибути або операції
- \- позначає приватні атрибути або операції
- # позначає захищені атрибути або операції

Параметр Спрямованість

Кожен параметр в операції (методі) може бути позначений як in, out або inout , що визначає його напрямок щодо викликаючого. Ця спрямованість показана перед назвою параметра.



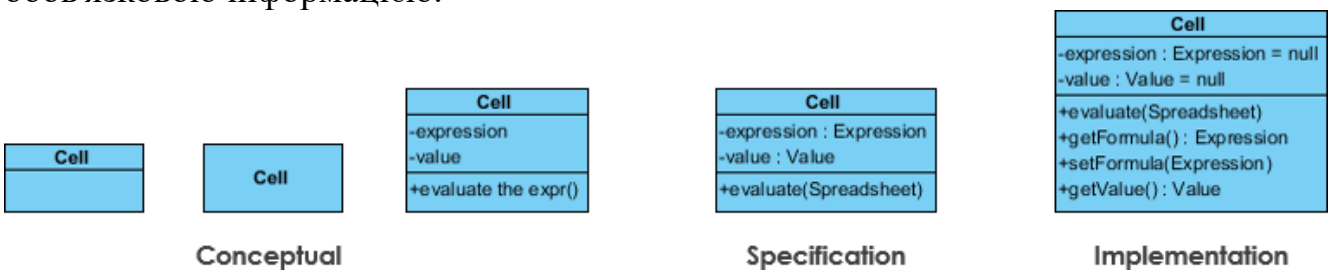
Перспективи діаграми класів

Вибір точки зору залежить від того, наскільки далеко ви просунулися в процесі розвитку. Під час формулювання моделі домену , наприклад, ви рідко проходите далі концептуальної точки зору . Моделі аналізу зазвичай містять поєднання концептуальних і специфікаційних точок зору . Розробка моделі дизайну зазвичай починається з великого акценту на перспективі специфікації та розвивається в перспективі впровадження .

Діаграму можна інтерпретувати з різних точок зору:

- Концептуальний : представляє концепції в домені
- Специфікація : фокус зосереджений на інтерфейсах абстрактного типу даних (ADT) у програмному забезпеченні
- Реалізація : описує, як класи реалізовуватимуть свої інтерфейси

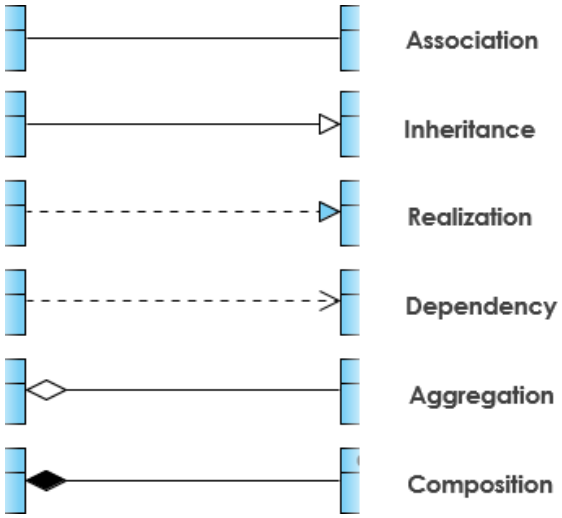
Перспектива впливає на кількість деталей, які потрібно надати, і на види стосунків, які варто представити. Як ми вже згадували вище, ім'я класу є єдиною обов'язковою інформацією.



Відносини між класами

UML — це не лише красиві картинки. При правильному використанні UML точно передає, як код має бути реалізований із діаграм. За умови точної інтерпретації реалізований код правильно відображатиме наміри дизайнера. Чи можете ви описати, що означають кожне зі зв’язків відносно вашої цільової мови програмування, зображеної на малюнку нижче?

Якщо ви ще не можете їх розпізнати, не біда, цей розділ має на меті допомогти вам зрозуміти зв’язки класів UML. Клас може бути залучений до одного або кількох відносин з іншими класами. Відносини можуть бути одного з таких типів:

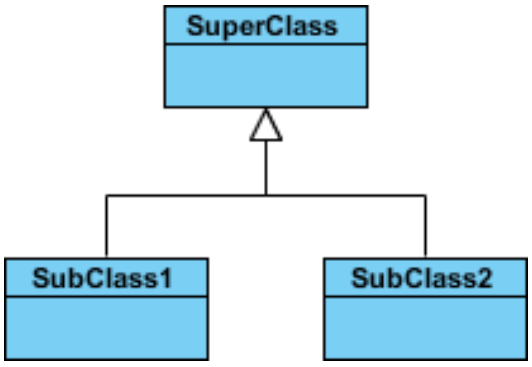


Успадкування (або узагальнення):

Узагальнення — це таксономічний зв’язок між більш загальним класифікатором і більш конкретним класифікатором. Кожен екземпляр конкретного класифікатора також є непрямим екземпляром загального класифікатора. Таким чином, специфічний класифікатор успадковує ознаки більш загального класифікатора.

- Представляє зв'язок "є-є".
- Назва абстрактного класу виділено курсивом.
- SubClass1 і SubClass2 є спеціалізаціями SuperClass.

На малюнку нижче показано приклад ієрархії успадкування. SubClass1 і SubClass2 є похідними від SuperClass. Зв’язок відображається як суцільна лінія з порожнистою стрілкою, яка вказує від дочірнього елемента до батьківського.



Приклад успадкування - фігури

Асоціація

Асоціації — це зв’язки між класами на діаграмі класів UML. Вони представлені суцільною лінією між класами. Асоціації, як правило, називаються дієсловом або дієслівною фразою, яка відображає сферу реальної проблеми.

Проста асоціація

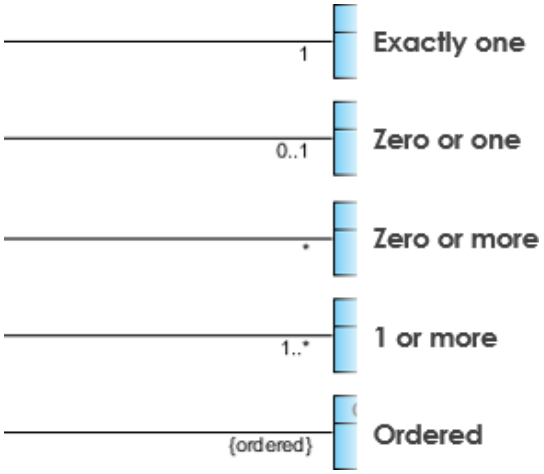
- Структурний зв'язок між двома однолітками.
- Існує асоціація між Class1 і Class2

На малюнку нижче показано приклад простої асоціації. Існує асоціація, яка з'єднує <<контрольний>> клас Class1 і <<граничний>> клас Class2. Відношення відображається суцільною лінією, що з'єднує два класи.

Кардинальність

Кардинальність виражається в термінах:

- один до одного
- один до багатьох
- багато багатьом



Агрегація

Особливий вид асоціації.

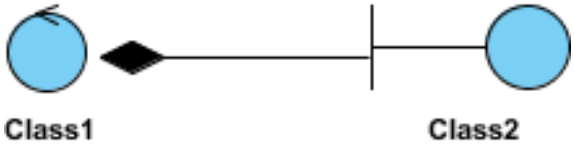
- Він представляє «частину» відносин.
- Class2 є частиною Class1.
- Багато екземплярів (позначених *) Class2 можуть бути пов'язані з Class1.
- Об'єкти Class1 і Class2 мають різні терміни життя.

На малюнку нижче показано приклад агрегації. Зв'язок відображається суцільною лінією з незаповненим ромбом на кінці асоціації, яка пов'язана з класом, який представляє сукупність.

Композиція

- Особливий тип агрегації, де частини руйнуються, коли руйнується ціле.
- Об'єкти Class2 живуть і вмирають з Class1.
- Class2 не може стояти сам по собі.

На малюнку нижче показано приклад композиції. Зв’язок відображається суцільною лінією із зафарбованим ромбом на кінці асоціації, яка пов’язана з класом, який представляє ціле або складене.

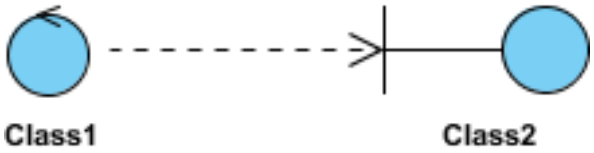


Залежність

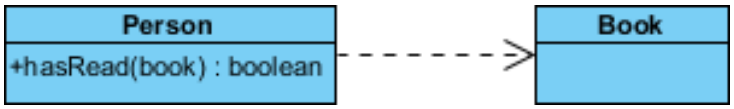
Об’єкт одного класу може використовувати об’єкт іншого класу в коді методу. Якщо об’єкт не зберігається в жодному полі, це моделюється як зв’язок залежності.

- Особливий вид асоціації.
- Існує між двома класами, якщо зміни у визначенні одного можуть спричинити зміни в іншому (але не навпаки).
- Class1 залежить від Class2

На малюнку нижче показано приклад залежності. Зв'язок відображається пунктирною лінією з відкритою стрілкою.



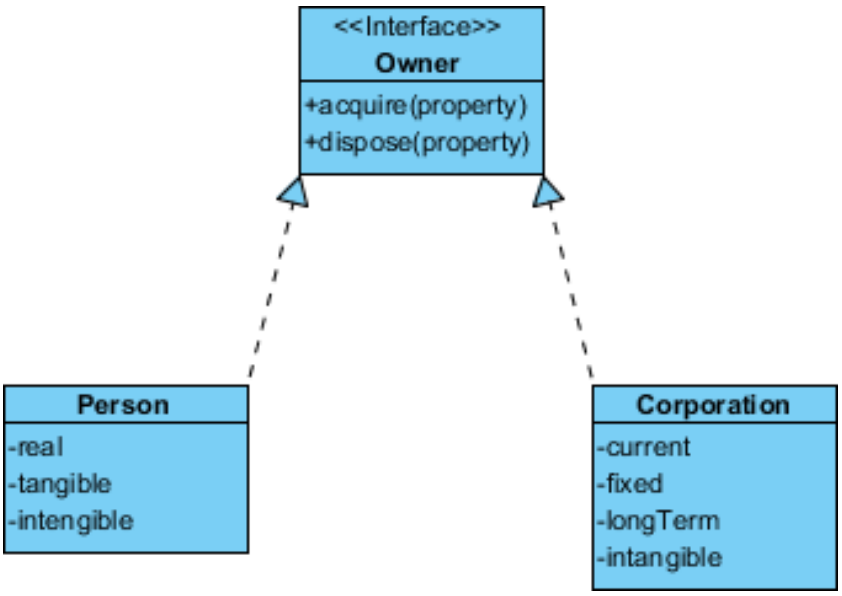
На малюнку нижче показано інший приклад залежності. Клас Person може мати метод hasRead з параметром Book, який повертає true, якщо особа прочитала книгу (можливо, перевіривши якусь базу даних).



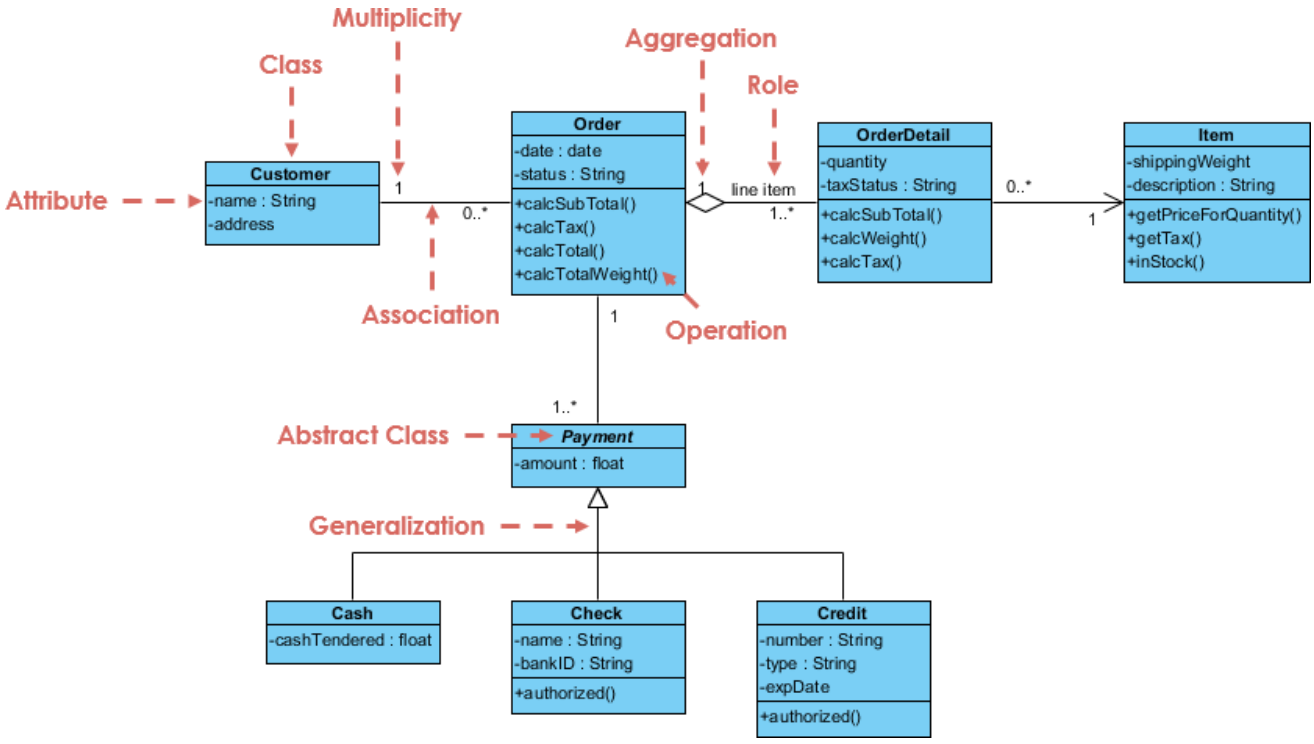
Реалізація

Реалізація - це зв'язок між класом плану та об'єктом, що містить відповідні деталі рівня реалізації. Кажуть, що цей об’єкт реалізує клас blueprint. Іншими словами, ви можете зрозуміти це як зв'язок між інтерфейсом і класом реалізації.

Наприклад, інтерфейс власника може визначати методи придбання власності та розпорядження нею. Класи Person і Corporation повинні реалізувати ці методи, можливо, дуже різними способами.

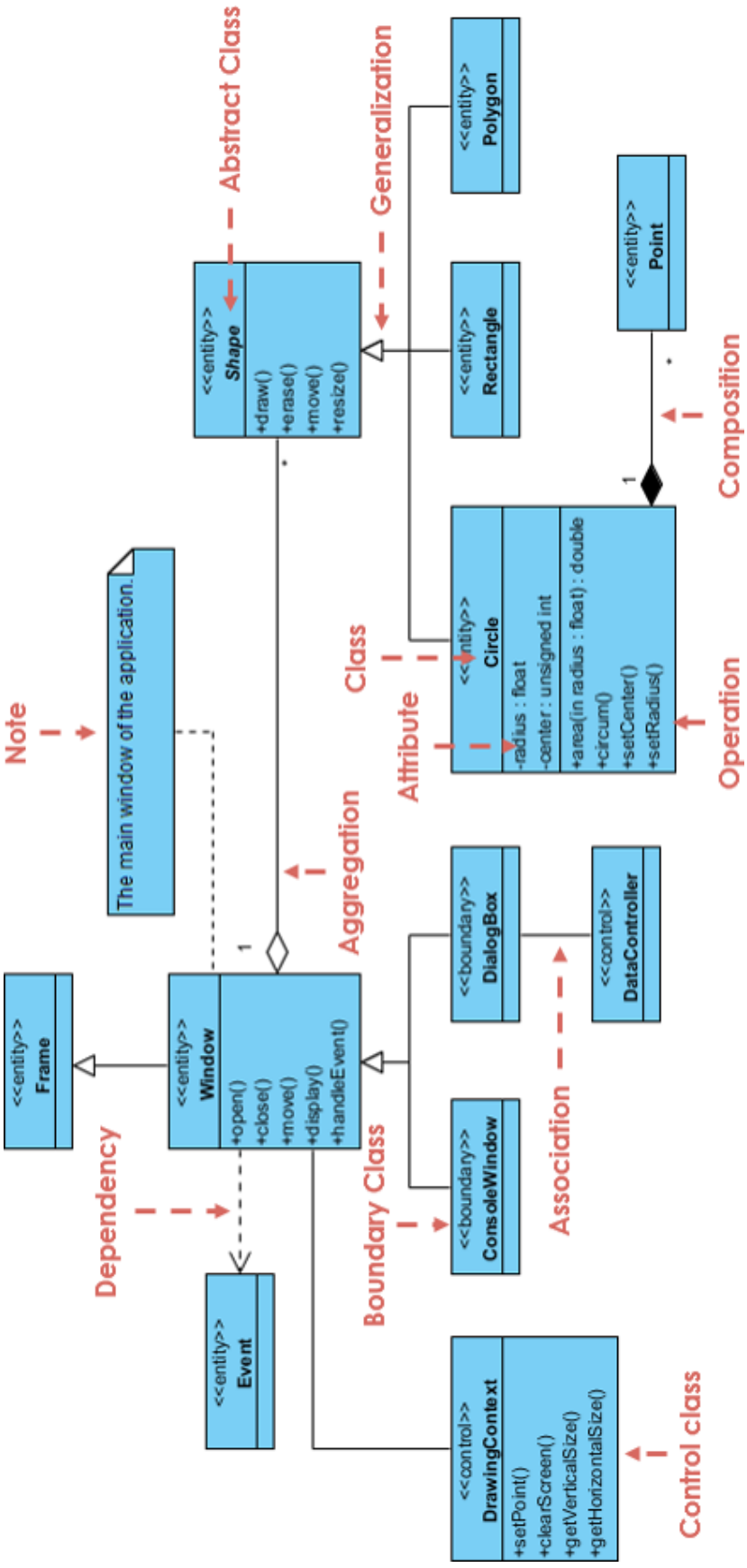


Приклад діаграми класів: Система порядку



Приклад діаграми класів: GUI

Діаграма класів також може містити примітки до класів або зв’язків.



3. Основні правила роботи із діаграмами класів у середовищі StarUml
https://docs.staruml.io/working-with-uml-diagrams/class-diagram#create-class-diagram

Порядок виконання роботи:

1. Опрацюйте теоретичний матеріал та засвойте основи, концепції роботи із «class diagrams» (на основі наданої презентації).

2. *В разі якщо ви здійснюєте побудову діаграми класів для проектуємого проекту і на даному етапі відсутні напрацювання у вигляді коду, то:*

- розглядати діаграму класів рекомендується ***з концептуальної точки зору***, яка використовується на початкових етапах моделювання та розробки.
- створити діаграму класів для одного/декількох сценаріїв діаграми прецедентів, що була розроблена у попередній лабораторній роботі. Для кожного класу необхідно задати атрибути і операції.
- кожен клас повинен бути детально задокументовано - необхідно задати текстовий опис самого класу, опису його атрибутів і операцій;
- провести етап об'єктно-орієнтованого проектування та встановити необхіді зв'язки та їх види (асоціації, агрегації, композиції тощо);
- при проектуванні врахувати SOLID принципи (https://metanit.com/sharp/patterns/) та можливість використання GOF-паттернів (https://metanit.com/sharp/patterns/);
- здійснити кодову генерацію для побудованої діаграми класів.

3. *В разі якщо ви здійснюєте побудову діаграми класів для розробленого проекту або використовуєте наданий тестовий навчальний проект (eShopOnWeb), то*

- розглядати діаграму класів рекомендується ***з точки зору реалізації та специфікації (реальні класи + інтерфейси).*** Відповідна синергенція дозволяє краще аналізувати архітектурні рішення та виявляти «слабкі місця» проекту;
- провести деталізовану побудову діаграми класів в розрізі 1 логічного ланцюжка «окремо взятий контролер – сервіс – UOW – конкретний репозиторій – Generic Repository – entities» з врахуванням всіх допоміжних класів та сторонніх бібліотек (інтерфейси, файли конфігурацій, DI, DTO, FluentValidation, Automapper тощо);
- оцінити коректність реалізації паттернів/рішень в розрізі розробленого проекту на основі побудованої діаграми класів. Провести аналіз отриманої UML діаграми, оцінити слабкі місця та можливість покращення структури коду і відповідні шляхи цього;
- в разі, якщо за основу береться тестовий проект то здійснити побудову діаграми класів для наступного зв'язку з врахуванням всіх допоміжних класів та сторонніх бібліотек (інтерфейси, файли конфігурацій, DI, DTO, FluentValidation, Automapper тощо):

```
CatalogController ->                                                    ->
Index.cshtml.cs   -> CachedCatalogViewModelService -> CatalogViewModelService ->
                                                                        ->
->                                    -> CatalogItem
-> EfRepository ->  CatalogContext    -> CatalogBrand    ->    BaseEntity
->                                    -> CatalogType
```

Контрольні питання:

1. Призначення діаграми класів?
2. Як позначається клас і що включає?
3. Що таке атрибут у діаграмі класів?
4. Як позначаються області видимості у атрибута?
5. Що таке операція в діаграмі класів?
6. Перелічити види відносин між класами, охарактеризувати кожне з них?
7. Як позначається об'єкт (примірник) класу?

Література:

1. Grady Booch, James Rumbaugh, Ivar Jacobson. Unified Modeling Language User Guide. Addison-Wesley, 1999. 496 p. ISBN: 0-201-57168-4.
2. Martin Fowler. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley, 2004. 208 p. ISBN: 0-321-19368-7.
3. Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Prentice Hall, 2004. 736 p. ISBN: 0-13-148906-2.
4. Grady Booch, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language Reference Manual. Addison-Wesley, 1999. 720 p. ISBN: 0-201-30998-X.
5. Doug Rosenberg, Kendall Scott. Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example. Addison-Wesley, 2001. 512 p. ISBN: 0-201-60489-7.
6. Jean-Marc Nerson. Object-Oriented Software Construction. Prentice Hall, 2002. 704 p. ISBN: 0-13-629155-4.
7. Jim Arlow, Ila Neustadt. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design. Addison-Wesley, 2005. 624 p. ISBN: 0-321-26779-8.
8. Michael Blaha, James Rumbaugh. Object-Oriented Modeling and Design with UML. Prentice Hall, 2004. 416 p. ISBN: 0-13-196845-0.
9. Alan Dennis, Barbara Haley Wixom, David Tegarden. Systems Analysis and Design: An Object-Oriented Approach with UML. Wiley, 2014. 544 p. ISBN: 978-1-118-56497-6.
10. Craig Larman. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design. Prentice Hall, 2005. 736 p. ISBN: 0-13-148906-2.
11. Richard F. Schmidt. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Pearson, 2019. 480 p. ISBN: 978-0131489066.
12. Dan Pilone, Neil Pitman. UML 2.0 in a Nutshell. O'Reilly Media, 2005. 258 p. ISBN: 0-596-00795-7.
13. Doug Rosenberg, Matt Stephens. Use Case Driven Object Modeling with UML: A Practical Approach. Apress, 2007. 648 p. ISBN: 978-1-59059-774-3.
14. Grady Booch, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language User Guide (2nd Edition). Addison-Wesley, 2005. 704 p. ISBN: 0-321-26777-1.