Питання до іспиту АППЗ

1. Визначити поняття системної розробки програмного забезпечення.

2. Визначити поняття детального проектування.

3. Визначити поняття вбудованого програмного забезпечення.

4. Обґрунтувати використанню метрик при оцінці якості архітектури програмного забезпечення.

6. Дати оцінку ролі і місцю детального проектування у життєвому циклі програмного забезпечення

7. Дати оцінку цілям вибору архітектури програмного забезпечення.

9. Визначити поняття декомпозиції програмного забезпечення.

10. Описати типову схему детального проектування.

13. Визначити поняття каркасу програмного забезпечення.

14. Дати оцінку використанню повторно використовуваним компонентам у детальному проектуванні.

20. Обґрунтувати специфікацію класів та функцій у детальному проектуванні.

21. Навести класифікацію зразків проектування.

22. Визначити поняття інваріантів класів.

23. Обґрунтувати використання компонентів при розробці архітектури програмного забезпечення.

24. Проаналізувати інваріанти передумови та післяумови.

25. Дати оцінку використання моделей при розробці архітектури програмного забезпечення.

26. Визначити прийоми специфікації алгоритмів.

27. Дати оцінку перевагам та недолікам використання програмної архітектури заснованої на потоках даних.

29. Дати оцінку перевагам та недолікам використання програмної архітектури незалежних компонентів.

31. Дати оцінку перевагам та недолікам використання програмної архітектури віртуальних машин.

32. Порівняти переваги та недоліки використання блок-схем та псевдокоду при детальному проектуванні.

33. Дати оцінку перевагам та недолікам використання репозиторної архітектури.

36. Проаналізувати використання креаційних зразків у детальному проектуванні.

38. Проаналізувати використання структурних зразків у детальному проектуванні.

39. Описати інструментальні засоби для розробки програмної архітектури.

40. Проаналізувати використання поведінкових зразків у детальному проектуванні.

41. Описати етапи контролю якості при виборі архітектури.

42. Визначити поняття зразків проектування заснованих на поведінці додатків.

43. Дати оцінку перевірці архітектури за допомогою варіантів використання.

45. Описати етапи інспектування вибору архітектури.

47. Перерахуйте основні стандарти для документування проекту програмного продукту.

48. Проаналізувати співвідношення варіантів використання архітектури та детального проектування.

49. Навести переваги використання рівневих архітектур.

50. Обґрунтувати використання детальних діаграм послідовності.

51. Визначити поняття створення архітектури програмного забезпечення.

52. Обґрунтувати використання блок-схем у детальному проектуванні.

53. Описати процедуру вибору архітектури.

54. Дати оцінку основним прийомам детального проектування.

55. Визначити поняття зразка проектування програмного забезпечення.

56. Обґрунтувати використання псевдокоду у детальному проектуванні.

57. Навести класифікацію архітектур програмного забезпечення.

58. Проаналізувати використання інтерфейсів у детальному проектуванні.

59. Дати оцінку використання каркасів у проектуванні програмного забезпечення

60. Обґрунтувати використання детальних діаграм потоків даних.

61. Подати за допомогою діаграми класів структуру патерна Спостерігач (Observer), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

62. Подати за допомогою діаграми класів структуру патерна Команда (Command), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

63. Подати за допомогою діаграми класів структуру патерна Ітератор (Iterator), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

64. Подати за допомогою діаграми класів структуру патерна Пристосуванець (Flyweight), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

65. Подати за допомогою діаграми класів структуру патерна Міст (Bridge), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

66. Подати за допомогою діаграми класів структуру патерна Замісник (Proxy), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

67. Подати за допомогою діаграми класів структуру патерна Декоратор (Decorator), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

68. Подати за допомогою діаграми класів структуру патерна Адаптер (Adapter), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

69. Подати за допомогою діаграми класів структуру патерна Інформаційний експерт (Information Expert) , коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

70. Подати за допомогою діаграми класів структуру патерна Низька зв’язність (Low Coupling), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

71. Подати за допомогою діаграми класів структуру патерна Фасад (Facade), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

72. Подати за допомогою діаграми класів структуру патерна Компоновщик (Composite), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

73. Подати за допомогою діаграми класів структуру патерна Інтерпретатор (Interpreter), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

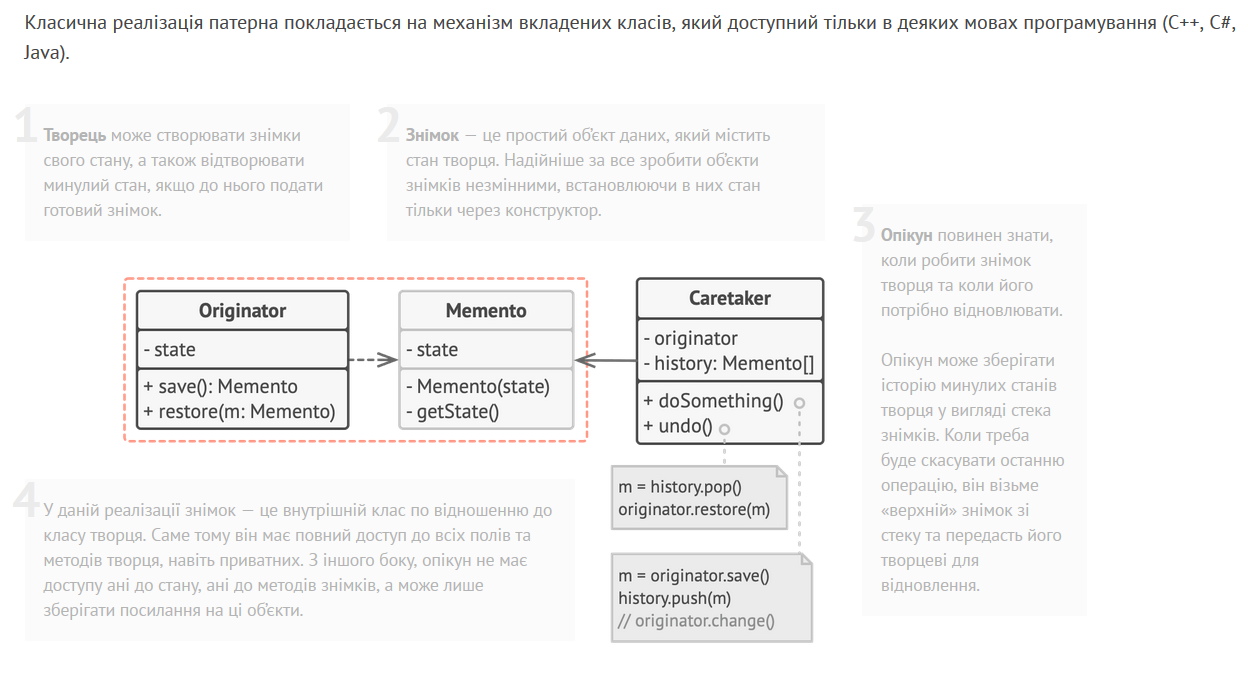
74. Подати за допомогою діаграми класів структуру патерна Відвідувач (Visitor), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

75. Подати за допомогою діаграми класів структуру патерна Посередник (Mediator), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

76. Подати за допомогою діаграми класів структуру патерна Стан (State), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

77. Подати за допомогою діаграми класів структуру патерна Стратегія (Strategy), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

78. Подати за допомогою діаграми класів структуру патерна Зберігач (Memento), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування



**Зберігач** — це поведінковий патерн проектування, що дає змогу зберігати та відновлювати минулий стан об’єктів, не розкриваючи подробиць їхньої реалізації.

**Застосування**

* Коли вам потрібно зберігати миттєві знімки стану об’єкта (або його частини) для того, щоб об’єкт можна було відновити в тому самому стані.
* Патерн Знімок дозволяє створювати будь-яку кількість знімків об’єкта і зберігати їх незалежно від об’єкта, з якого роблять знімок. Знімки часто використовують не тільки для реалізації операції скасування, але й для транзакцій, коли стан об’єкта потрібно «відкотити», якщо операція не була вдалою.
* Коли пряме отримання стану об’єкта розкриває приватні деталі його реалізації, порушуючи інкапсуляцію.
* Патерн пропонує виготовити знімок саме вихідному об’єкту, тому що йому доступні всі поля, навіть приватні.

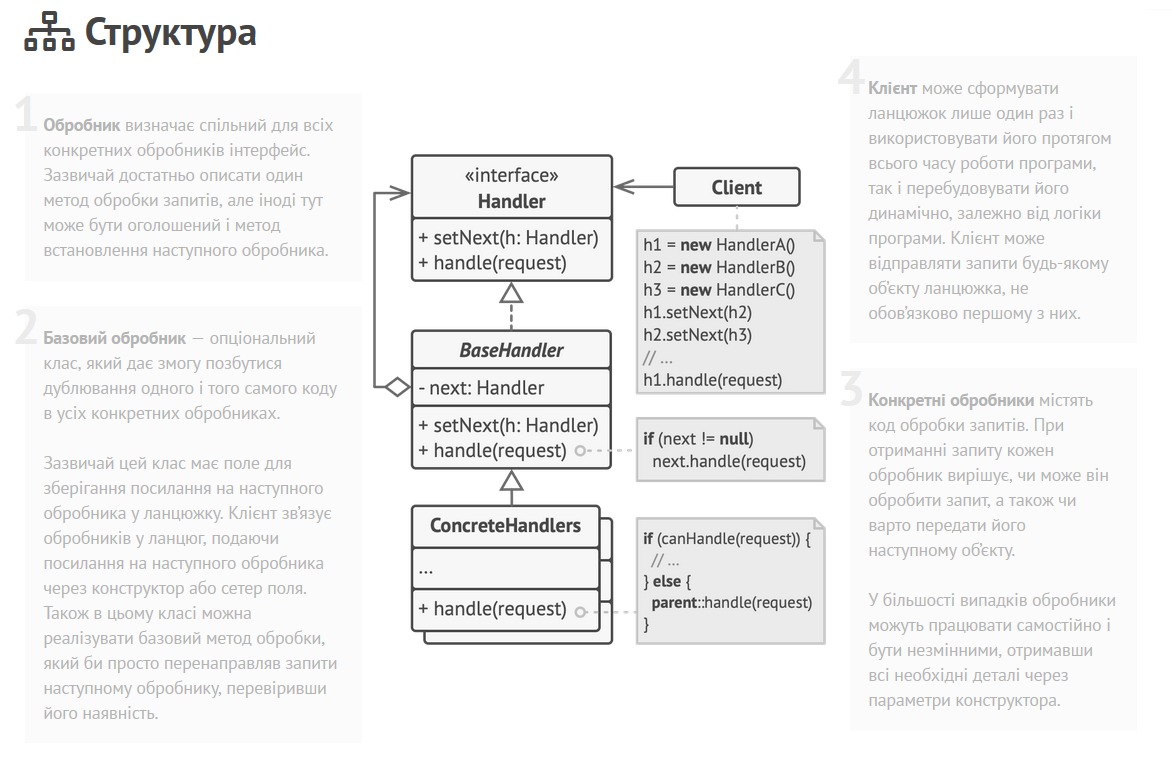
**Переваги:**

* Не порушує інкапсуляцію вихідного об’єкта.
* Спрощує структуру вихідного об’єкта. Йому не потрібно зберігати історію версій свого стану.

**Недоліки:**

* Вимагаєбагатопам’яті, якщоклієнтидужечастостворюютьзнімки.
* Можеспричинитидодатковівитратипам’яті, якщооб’єкти, щозберігаютьісторію, незвільняютьресурси, зайнятізастарілимизнімками.
* Вдеякихмовах (наприклад, PHP, Python, JavaScript) складногарантувати, щоблишевихіднийоб’єктмавдоступдостанузнімка.

79. Подати за допомогою діаграми класів структуру патерна Ланцюжок обов’язків (Chain of Responsibility), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування



**Ланцюжок обов’язків** — це поведінковий патерн проектування, що дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком.

## Застосування

* Якщо програма має обробляти різноманітні запити багатьма способами, але заздалегідь невідомо, які конкретно запити надходитимуть і які обробники для них знадобляться.
* За допомогою Ланцюжка обов’язків ви можете зв’язати потенційних обробників в один ланцюг і по отриманню запита по черзі питати кожного з них, чи не хоче він обробити даний запит.
* Якщо важливо, щоб обробники виконувалися один за іншим у суворому порядку.
* Ланцюжок обов’язків дозволяє запускати обробників один за одним у тій послідовності, в якій вони стоять в ланцюзі.
* Якщо набір об’єктів, здатних обробити запит, повинен задаватися динамічно.
* У будь-який момент ви можете втрутитися в існуючий ланцюжок і перевизначити зв’язки так, щоби прибрати або додати нову ланку.

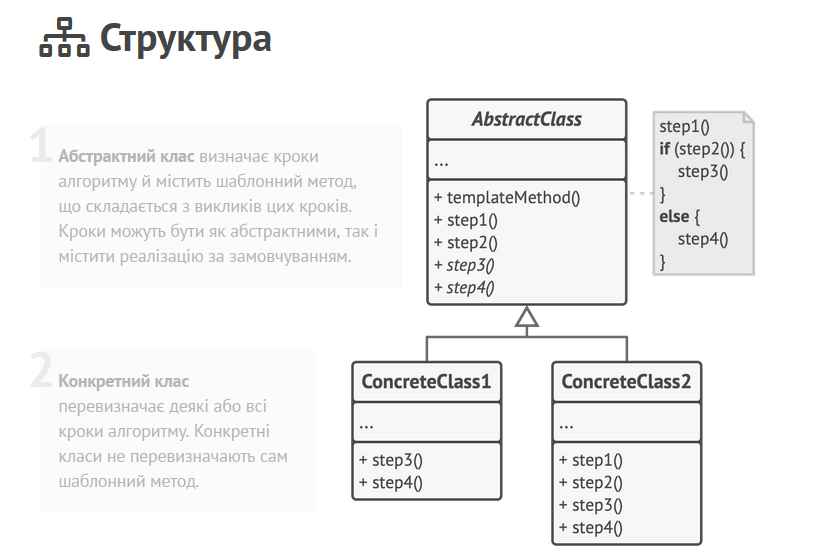
Переваги:

* Зменшує залежність між клієнтом та обробниками.
* Реалізує *принцип єдиного обов’язку*.
* Реалізує *принцип відкритості/закритості*.

Недоліки:

* Запит може залишитися ніким не опрацьованим.

80. Подати за допомогою діаграми класів структуру патерна Шаблонний метод (Template Method), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування



**Шаблонний метод** — це поведінковий патерн проектування, який визначає кістяк алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи. Патерн дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури.

## Застосування

* Якщо підкласи повинні розширювати базовий алгоритм, не змінюючи його структури.
* Шаблонний метод дозволяє підкласами розширювати певні кроки алгоритму через спадкування, не змінюючи при цьому структуру алгоритмів, оголошену в базовому класі.
* Якщо у вас є кілька класів, які роблять одне й те саме з незначними відмінностями. Якщо ви редагуєте один клас, тоді доводиться вносити такі ж виправлення до інших класів.
* Патерн шаблонний метод пропонує створити для схожих класів спільний суперклас та оформити в ньому головний алгоритм у вигляді кроків. Кроки, які відрізняються, можна перевизначити у підкласах.
* Це дозволить прибрати дублювання коду в кількох класах, які відрізняються деталями, але мають схожу поведінку.

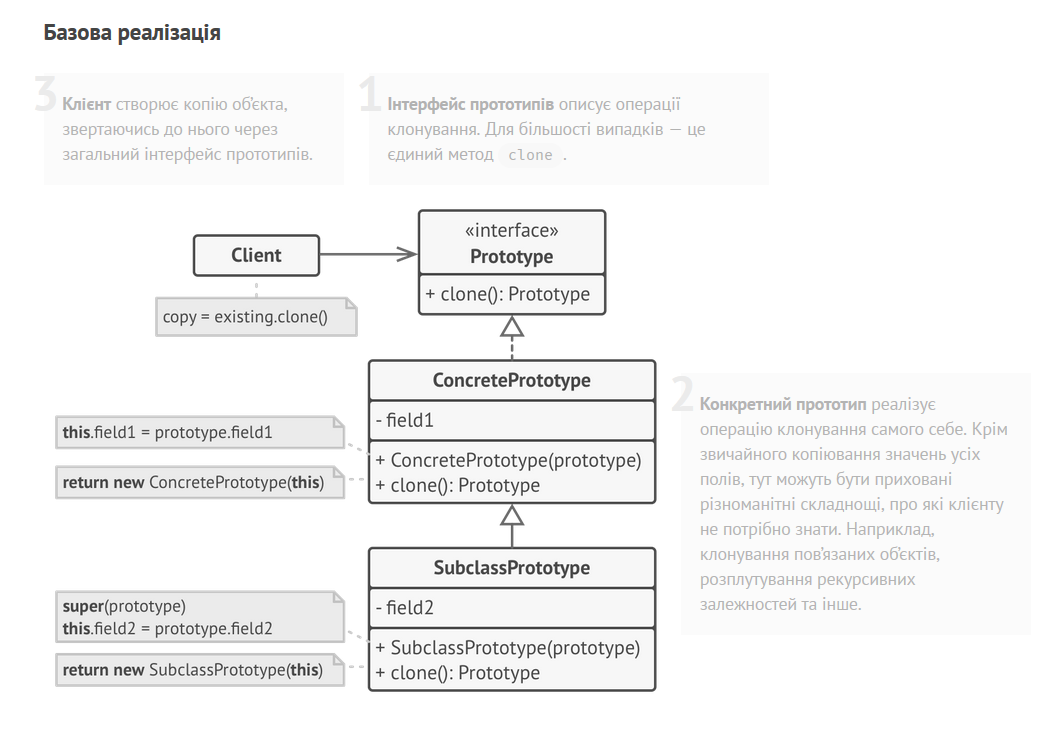
Переваги :

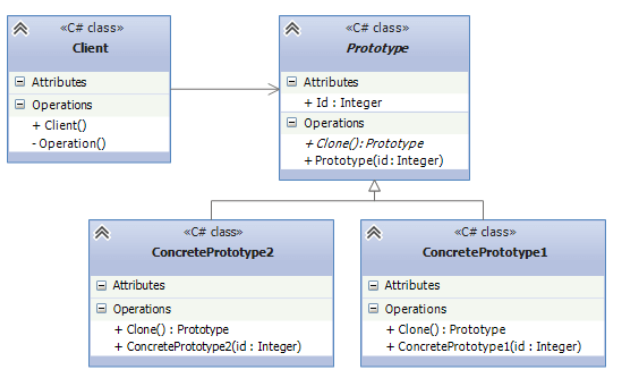
* Полегшує повторне використання коду.

Недоліки:

* Ви жорстко обмежені скелетом існуючого алгоритму.
* Ви можете порушити принцип підстановки Барбари Лісков, змінюючи базову поведінку одного з кроків алгоритму через підклас.
* У міру зростання кількості кроків шаблонний метод стає занадто складно підтримувати.

81. Подати за допомогою діаграми класів структуру патерна Прототип (Prototype), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування





**Прототип** — це породжувальний патерн проектування, що дає змогу копіювати об’єкти, не вдаючись у подробиці їхньої реалізації.

## Застосування

* Коли ваш код не повинен залежати від класів об’єктів, призначених для копіювання.
* Таке часто буває, якщо ваш код працює з об’єктами, поданими ззовні через який-небудь загальний інтерфейс. Ви не зможете прив’язатися до їхніх класів, навіть якби захотіли, тому що конкретні класи об’єктів невідомі.
* Патерн Прототип надає клієнту загальний інтерфейс для роботи з усіма прототипами. Клієнту не потрібно залежати від усіх класів об’єктів, призначених для копіювання, а тільки від інтерфейсу клонування.
* Коли ви маєте безліч підкласів, які відрізняються початковими значеннями полів. Хтось міг створити усі ці класи для того, щоб мати легкий спосіб породжувати об’єкти певної конфігурації.
* Патерн Прототип пропонує використовувати набір прототипів замість створення підкласів для опису популярних конфігурацій об’єктів.
* Таким чином, замість породження об’єктів з підкласів ви копіюватимете існуючі об’єкти-прототипи, внутрішній стан яких вже налаштовано. Це дозволить уникнути вибухоподібного зростання кількості класів програми й зменшити її складність.

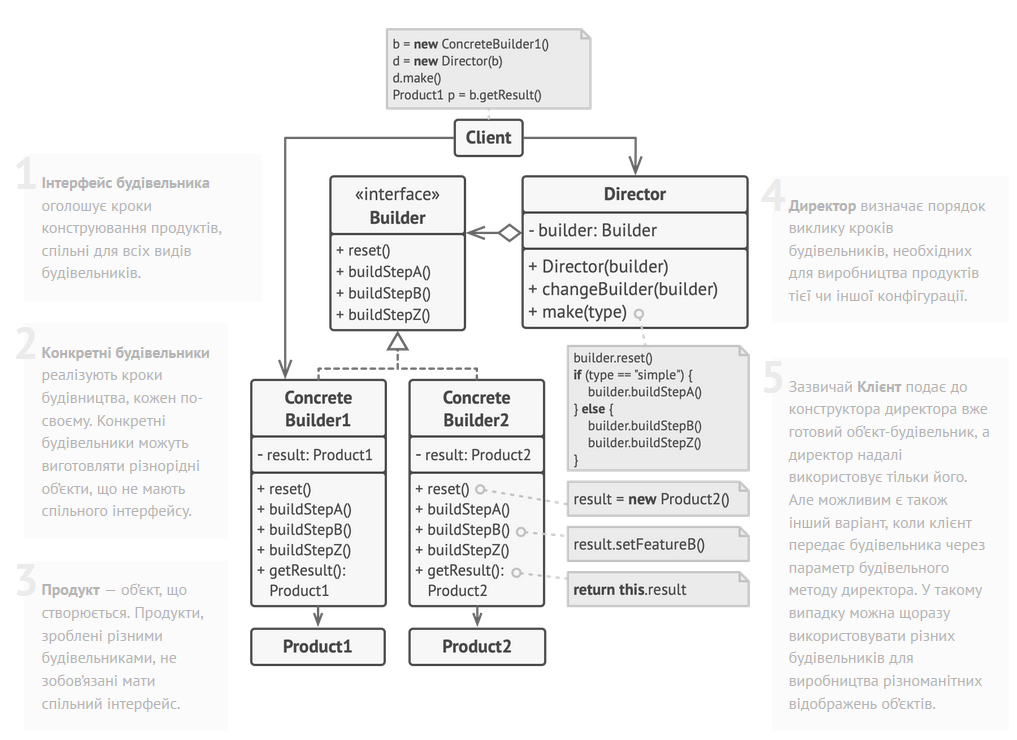
Переваги :

* Дозволяє клонувати об’єкти без прив’язки до їхніх конкретних класів.
* Менша кількість повторювань коду ініціалізації об’єктів.
* Прискорює створення об’єктів.
* Альтернатива створенню підкласів під час конструювання складних об’єктів.

Недоліки:

* Складно клонувати складові об’єкти, що мають посилання на інші об’єкти.

82. Подати за допомогою діаграми класів структуру патерна Будівельник (Builder), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування



**Будівельник** — це породжувальний патерн проектування, що дає змогу створювати складні об’єкти крок за кроком. Будівельник дає можливість використовувати один і той самий код будівництва для отримання різних відображень об’єктів.

**Застосування**

* Коли ви хочете позбутися від «телескопічного конструктора».
* Коли ваш код повинен створювати різні уявлення якогось об’єкта. Наприклад, дерев’яні та залізобетонні будинки.
* Будівельник можна застосувати, якщо створення кількох відображень об’єкта складається з однакових етапів, які відрізняються деталями.
* Інтерфейс будівельників визначить всі можливі етапи конструювання. Кожному відображенню відповідатиме власний клас-будівельник. Порядок етапів будівництва визначатиме клас-директор.
* Коли вам потрібно збирати складні об’єкти, наприклад, дерева [Компонувальника](https://refactoring.guru/uk/design-patterns/composite).
* Будівельник конструює об’єкти покроково, а не за один прохід. Більш того, кроки будівництва можна виконувати рекурсивно. А без цього не побудувати деревоподібну структуру на зразок [Компонувальника](https://refactoring.guru/uk/design-patterns/composite).
* Зауважте, що Будівельник не дозволяє стороннім об’єктам отримувати доступ до об’єкта, що конструюється, доки той не буде повністю готовий. Це захищає клієнтський код від отримання незавершених «битих» об’єктів.

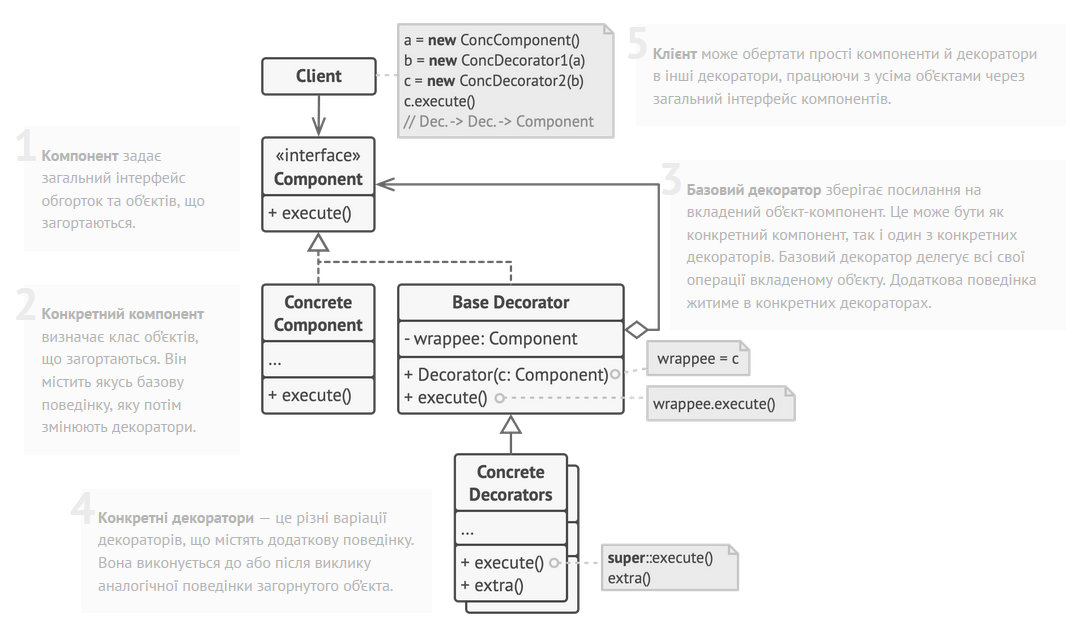
Переваги:

* Дозволяє створювати продукти покроково.
* Дозволяє використовувати один і той самий код для створення різноманітних продуктів.
* Ізолює складний код конструювання продукту від його головної бізнес-логіки.

Недоліки:

* Ускладнює код програми за рахунок додаткових класів.
* Клієнт буде прив’язаний до конкретних класів будівельників, тому що в інтерфейсі будівельника може не бути методу отримання результату.

83. Подати за допомогою діаграми класів структуру патерна Оболонка (Wrapper), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування



Оболонка — це структурний патерн проектування, що дає змогу динамічно додавати об’єктам нову функціональність, загортаючи їх у корисні «обгортки».

## Застосування

* Якщо вам потрібно додавати об’єктам нові обов’язки «на льоту», непомітно для коду, який їх використовує.
* Об’єкти вкладаються в обгортки, які мають додаткові поведінки. Обгортки і самі об’єкти мають однаковий інтерфейс, тому клієнтам не важливо, з чим працювати — зі звичайним об’єктом чи з загорнутим.
* Якщо не можна розширити обов’язки об’єкта за допомогою спадкування.
* У багатьох мовах програмування є ключове слово final, яке може заблокувати спадкування класу. Розширити такі класи можна тільки за допомогою Декоратора.

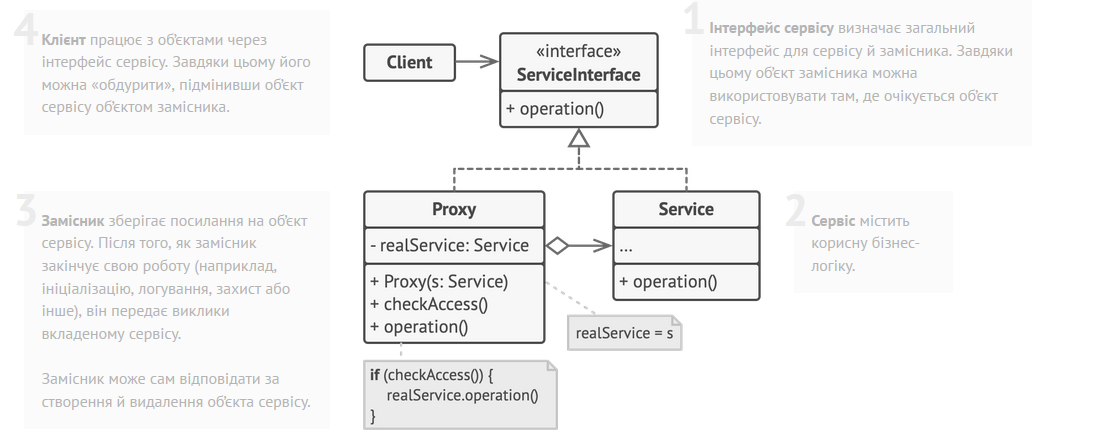
Переваги:

* Більша гнучкість, ніж у спадкування.
* Дозволяє додавати обов’язки «на льоту».
* Можна додавати кілька нових обов’язків одразу.
* Дозволяє мати кілька дрібних об’єктів, замість одного об’єкта «на всі випадки життя».

Недоліки:

* Важко конфігурувати об’єкти, які загорнуто в декілька обгорток одночасно.
* Велика кількість крихітних класів.

84. Подати за допомогою діаграми класів структуру патерна Сурогат (Surrogate), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування



**Cурогат** — це структурний патерн проектування, що дає змогу підставляти замість реальних об’єктів спеціальні об’єкти-замінники. Ці об’єкти перехоплюють виклики до оригінального об’єкта, дозволяючи зробити щось до чи після передачі виклику оригіналові.

Застосування

* Лінива ініціалізація (віртуальний проксі). Коли у вас є важкий об’єкт, який завантажує дані з файлової системи або бази даних.
* Замість того, щоб завантажувати дані відразу після старту програми, можна заощадити ресурси й створити об’єкт тоді, коли він дійсно знадобиться.
* Захист доступу (захищаючий проксі). Коли в програмі є різні типи користувачів, і вам хочеться захистити об’єкт від неавторизованого доступу. Наприклад, якщо ваші об’єкти — це важлива частина операційної системи, а користувачі — сторонні програми (корисні чи шкідливі).
* Проксі може перевіряти доступ під час кожного виклику та передавати виконання службовому об’єкту, якщо доступ дозволено.
* Локальний запуск сервісу (віддалений проксі). Коли справжній сервісний об’єкт знаходиться на віддаленому сервері.

Переваги:

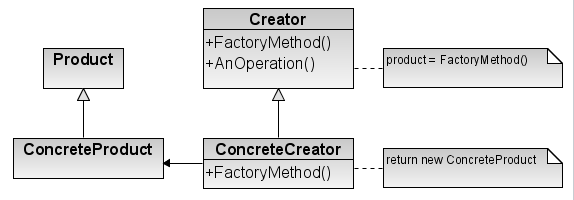
* Дозволяє контролювати сервісний об’єкт непомітно для клієнта.
* Може працювати, навіть якщо сервісний об’єкт ще не створено.
* Може контролювати життєвий цикл службового об’єкта.

Недоліки:

* Ускладнює код програми внаслідок введення додаткових класів.
* Збільшує час отримання відклику від сервісу.

85. Подати за допомогою діаграми класів структуру патерна Фабрика (Factory), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

Фабричний метод — це породжувальний патерн проектування, який визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об’єктів.



* **Product** — продукт:
  + визначає інтерфейс об'єктів, що створюються фабричним методом;
* **ConcreteProduct** — конкретний продукт:
  + реалізує інтерфейс *Product*;
* **Creator** — творець:
  + оголошує фабричний метод, що повертає об'єкт класу *Product*. *Creator* може також визначати реалізацію за умовчанням фабричного методу, що повертає об'єкт *ConcreteProduct*;
  + може викликати фабричний метод для створення об'єкта *Product*;
* **ConcreteCreator** — конкретний творець:
  + заміщує фабричний метод, що повертає об'єкт *ConcreteProduct*.

Застосування

* Коли типи і залежності об’єктів, з якими повинен працювати ваш код, невідомі заздалегідь.
* Коли ви хочете надати користувачам можливість розширювати частини вашого фреймворку чи бібліотеки.
* Коли ви хочете зекономити системні ресурси, повторно використовуючи вже створені об’єкти, замість породження нових.

Переваги:

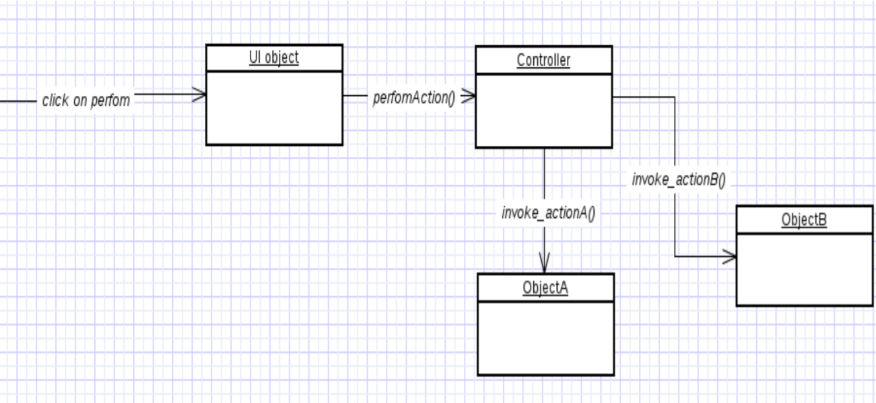
* Позбавляє клас від прив’язки до конкретних класів продуктів.
* Виділяє код виробництва продуктів в одне місце, спрощуючи підтримку коду.
* Спрощує додавання нових продуктів до програми.
* Реалізує принцип відкритості/закритості.

Недостатки:

* Може призвести до створення великих паралельних ієрархій класів, адже для кожного класу продукту потрібно створити власний підклас творця.

86. Подати за допомогою діаграми класів структуру патерна Контроллер (Controller), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

Контролер отримує введення користувача, обробляє його і посилає назад результат обробки, наприклад, у вигляді подання.



Переваги:

Зручно накопичувати інформацію про системні події (в разі, якщо системні операції виконуються в деякій певній послідовності).

Поліпшуються умови для повторного використання компонентів (системні події обробляються Контроллером а не елементами інтерфейсу користувача).

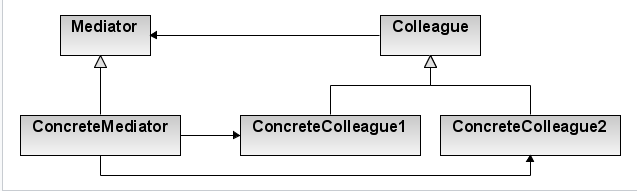
Недоліки:

Контролер може бути перевантажений.

87. Подати за допомогою діаграми класів структуру патерна Перенаправлення (Indirection), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

Перенаправлення – те саме, що й Посередник.

**Посередник** — це поведінковий патерн проектування, що дає змогу зменшити зв’язаність великої кількості класів між собою, завдяки переміщенню цих зв’язків до одного класу-посередника.



* **Mediator** – посередник:
  + визначає інтерфейс для обміну інформацією з об'єктами *Colleague*;
* **ConcreteMediator** – конкретний посередник:
  + реалізує кооперативну поведінку, координуючи дії об'єктів *Colleague*;
  + володіє інформацією про колег, та підраховує їх;
* **Класи Colleague** – колеги:
  + кожному класу *Colleague* відомо про свій об'єкт *Mediator*;
  + усі колеги обмінюються інформацією виключно через посередника, інакше за його відсутності їм довелося б спілкуватися між собою напряму

Слід використовувати шаблон *Посередник* у випадках, коли:

* існують об'єкти, зв'язки між котрими досить складні та чітко задані. Отримані при цьому залежності не структуровані та важкі для розуміння;
* не можна повторно використовувати об'єкт, оскільки він обмінюється інформацією з багатьма іншими об'єктами;
* поведінка, розподілена між кількома класами, повинна піддаватися налагодженню без створювання множини підкласів.

Переваги

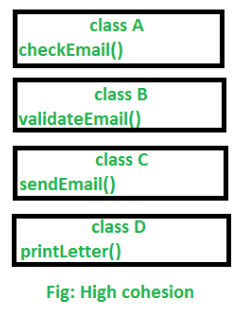
* Усуває залежності між компонентами, дозволяючи використовувати їх повторно.
* Спрощує взаємодію між компонентами.
* Централізує керування в одному місці.

Недоліки

* Посередник може **сильно «роздутися»**

88. Подати за допомогою діаграми класів структуру патерна Високе зчеплення (HighCohesion), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

Шаблон вирішує загальну проблему управління складністю за рахунок регулювання ступеня зачеплення класів. Зачеплення (cohesion) (або більш точно, функціональне зачеплення) - це міра пов'язаності і сфокусованість обов'язків класу. Вважається, що елемент має високий ступінь зачеплення, якщо його обов'язки тісно пов'язані між собою і він не виконує величезних обсягів роботи. Клас з низьким ступенем зачеплення виконує багато різнорідних функцій або незв'язаних між собою обов'язків.



Використання:

Необхідно забезпечити виконання об'єктами різнорідних функцій.

Переваги:

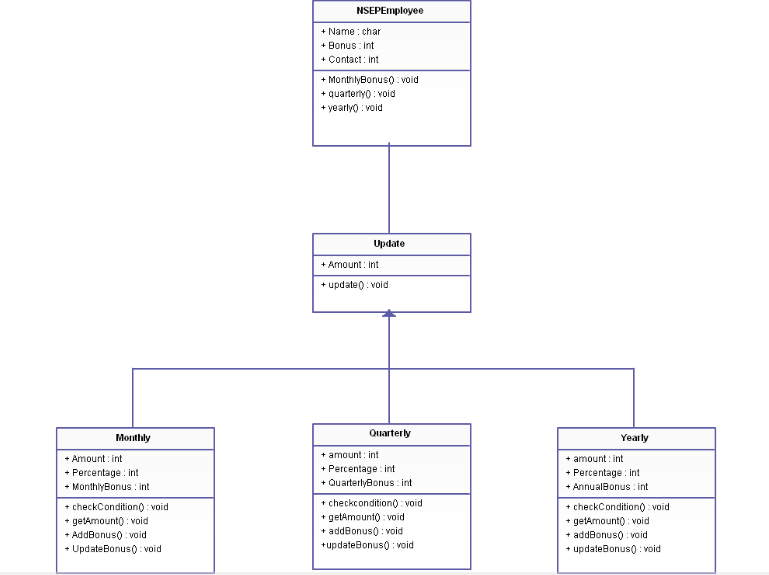
Класи з високим ступенем зачеплення прості в підтримці і повторному використанні.

Недоліки:  
Іноді буває невиправдано використовувати високу зачеплення для розподілених серверних об'єктів. В цьому випадку для забезпечення швидкодії необхідно створити кілька більших серверних об'єктів зі слабким зачепленням.

89. Подати за допомогою діаграми класів структуру патерна Транзакція (Transaction), коротко описати випадки його застосування, переваги та недоліки використання даного зразка проектування

Організовує бізнес-логіку в процедури, які керують кожна своїм запитом. Більшість бізнес-додатків можна представити у вигляді набору транзакцій. Якісь із них вибирають дані, якісь - змінюють. Кожна взаємодія користувача і системи містить певний набір дій. У деяких випадках це може бути просто висновок даних з БД. В інших випадках ці дії який можуть містити в собі безліч обчислень і перевірок.

Патерн Transaction організовує всю цю логіку в одну процедуру, працюючи в БД безпосередньо або через тонку обгортку. Кожна транзакція має свій Transaction, хоча загальні підзадачі можуть бути розбиті на процедури.



Переваги:

Головним достоїнством типового рішення транзакції є простота. Саме такий вид організації логіки, ефективний з точки зору сприйняття і продуктивності, дуже характерний і природний для невеликих додатків.

Недоліки:

Повторення фрагментів коду. Оскільки кожен сценарій транзакції покликаний обслуговувати одну транзакцію, всі загальні порції коду доводиться відтворювати знову і знову.