МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

А.І. Антонюк
А.О. Болдак

ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: ЛАБОРАТОРНИЙ ПРАКТИКУМ

Методичні вказівки
до виконання лабораторних
робіт з дисципліни
«Інженерія програмного забезпечення»

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 123 «Комп'ютерна інженерія»

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензент *Кулаков Ю.О.*, д.т.н., проф., професор кафедри обчислювальної техніки КПІ ім. Ігоря Сікорського

Відповідальний редактор *Писаренко А.В.*, канд. техн. наук, доцент

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 1 від 02.09.2022 р.) за поданням Вченої ради Факультету інформатики та обчислювальної техніки (протокол № 9 від 11.05.2022 р.)

Електронне мережне навчальне видання

Антонюк Андрій Іванович, канд. техн. наук
Болдак Андрій Олександрович, канд. техн. наук

ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: ЛАБОРАТОРНИЙ ПРАКТИКУМ

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ «ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

Методичні вказівки до виконання лабораторних робіт з дисципліни «інженерія програмного забезпечення» [Електронний ресурс] : Методичні вказівки. для студ. Спеціальності 123 «Комп'ютерна інженерія» /А.І. Антонюк, А.О. Болдак; КПІ ім. Ігоря Сікорського. - Електронні текстові дані (1 файл: 512 Кбайт). - Київ : КПІ ім. Ігоря Сікорського, 2022. - 51 с.

ЗМІСТ

ВСТУП

Дисципліна «Інженерія програмного забезпечення» призначена для вивчення методів та засобів програмування, зокрема шаблонів проектування. Студенти, які приступають до вивчення даної дисципліни вже засвоїли курс «Програмування» і в достатній мірі володіють навичками створення простих програм за допомогою мов програмування C++, Python, Object Pascal і Java.

Практична частина курсу складається з дев'яти лабораторних робіт і призначена для отримання навичок використання шаблонів проектування при розробці програмного забезпечення. Всі лабораторні роботи виконуються в інтегрованому середовищі розробки програм Eclipse. Роботи послідовно впорядковані за складністю і охоплюють всі теми, які вивчаються у курсі.

Матеріал до кожної лабораторної роботи містить мету, теоретичні довідки та рекомендації, загальне завдання, варіанти індивідуальних завдань, список питань для самоперевірки, зміст звіту про виконання лабораторних робіт, а також список рекомендованих інформаційних джерел для підготовки і виконання лабораторних робіт.

Послідовність лабораторних робіт така:

1. Підготовка програмного проекту.
2. Графічна нотація UML. Документування проекту.
3. Структурні шаблони проектування програмного забезпечення Composite, Decorator та Proxy.
4. Структурні шаблони проектування програмного забезпечення Flyweight, Adapter, Bridge та Facade.
5. Шаблони поведінки. Шаблони Iterator, Mediator, Observer. Шаблони поведінки.
6. Шаблони Strategy, Chain of Responsibility, Visitor.
7. Шаблони поведінки. Шаблони Memento, State, Command, Interpreter.
8. Породжувальні шаблони. Шаблони Prototype, Singleton, Factory Method.
9. Породжувальні шаблони. Шаблони Abstract Factory, Builder

ЛАБОРАТОРНА РОБОТА № 1.
ПІДГОТОВКА ПРОГРАМНОГО ПРОЕКТУ

Тема

Підготовка програмного проекту.

Мета

Отримання базових навичок з використання мови XML. Вивчення структури типового програмного проекту, форматів стандартних файлів опису проекту. Вивчення формату JAR. Здобуття навичок з використання засобів автоматизації процесу збірки програмних проектів на мові Java - Apache ANT (Another Neat Tool). Розробка програмного проекту на основі типового прикладу.

Короткі теоретичні відомості

Розширювана мова розмітки (англ. *Extensible Markup Language,* скорочено **XML) —** запропонований консорціумом World Wide Web (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для збереження та обміну. Є спрощеною підмножиною мови розмітки SGML. XML документ складається із текстових знаків, і придатний до читання людиною.

Логічна структура. Інструментом розмітки в XML є тег, який використовується для визначення меж елемента. Тег буває: початковий (<Ім'я-елемента>), кінцевий (</Ім'я-елемента>) та порожній або закритий (<Ім'я-елемента/>). XML документ має ієрархічну структуру, яка складається з елементів, асоційованих, з ними атрибутів та інструкцій. Непорожній елемент визначається парою тегів (початковий і кінцевий) з ім'ям цього елементу та тілом, що міститься між цими тегами. Тіло елементу складається з інших елементів та/або тексту. Порожній (без тіла) елемент може визначатися за допомогою одного порожнього тегу з ім'ям цього елементу. Атрибути є послідовністю пар ключ-значення (назва атрибута="значення атрибута") і знаходяться або у початковому, або у порожньому тезі і асоціюються з елементом, ім'я якого зазначено в тезі. Також, за допомогою спеціальних тегів визначаються інструкції обробки документу (<?Обробник параметр ?>) та коментарі (<!-- Текст коментаря -->).

Коректність. Коректний документ (well-formed document) відповідає всім синтаксичним правилам XML. Документ, що не є коректним, не може називатись XML-документом. Коректний XML документ має відповідати:

- Документ має лише один елемент в корені.
- Непорожні елементи позначено початковим та кінцевим тегами. Порожні елементи можуть помічатись «закритим» тегом.
- Один елемент не може мати декілька атрибутів з однаковим іменем. Значення атрибутів знаходяться або в одинарних ('), або у подвійних (") лапках.
- Теги можуть бути вкладені, але, не можуть перекриватись. Кожен некореневий елемент мусить повністю знаходитись в іншому елементі.
- Фактичне та задекларованe кодування документа мають збігатись.

Валідність. Документ називається валідним (англ. *valid),* якщо він є коректним та семантично вірним, тобто відповідає певному словнику XML, що визначається схемою (DTD, XML Schema або іншою).

JAR-файл — Java архів, який являє собою архів формату ZIP з додатковими метаданими в файлі маніфесту (META-INF/MAMFEST.MF). Маніфест може містити інформацію про назву проекту, версію, автора, стартовий клас, підпис файлів, тощо. Для створення JAR може бути використаний будь-який ZIP-сумісний архіватор. Найчастіше застосовують спеціалізовані засоби для роботи з JAR — завдання <jar> для Ant або утиліту jar із JDK.

Apache Ant (англ. *ant* — мураха і водночас акронім — «Another Neat Tool») — java-утиліта для автоматизації процесу збирання програмного продукту. На відміну від make, утиліта Ant повністю незалежна від платформи, потрібна лише наявність на застосовуваній системі встановленого робочого середовища Java — JRE. Відмова від використання команд операційної системи і формат XML забезпечують можливість перенесення сценаріїв.

Управління процесом збирання відбувається за допомогою XML-сценарію, який також називають Build-файлом (build.xml). Цей файл містить кореневий елемент-проект, який складається з

елементів-цілей (<target>). Цілі є еквівалентом процедур в мовах програмування і містять виклики команд-завдань (task). Завдання являє собою XML-елемент, який пов'язаний з java-класом, що виконує певну елементарну дію. Часто вживані завдання: javac, copy, delete, mkdir, jar. Між цілями можуть бути визначені залежності (атрибут depends) — кожна ціль виконується тільки після того, як виконані всі цілі, від яких вона залежить (якщо вони вже були виконані раніше, повторного виконання не здійснюється). Типовими прикладами цілей є clean (видалення проміжних файлів), compile (компіляція всіх класів), deploy (розгортання програми на сервері). Конкретний набір цілей та їхнього взаємозв'язку залежать від специфіки проекту. Ant дозволяє визначати власні типи завдань шляхом створення Java-класів, які реалізують певні інтерфейси, та зв'язування цих класів з елементом сценарію за допомогою елемента <taskdef>.

Завдання

1. Вивчити синтаксис та структуру мови XML. Вільно володіти поняттями тег, елемент, атрибут. Вміти редагувати XML-файли у текстовому редакторі.
2. Ознайомитись з призначенням і структурою архівів JAR. Вміти формувати архіви JAR та запускати на виконання Java-класи з архіву JAR за допомогою засобів командної строки. Знати призначення підпису архіву JAR.
3. Ознайомитись з засобом автоматизації збірки проектів ANT. Вивчити призначення і структуру файлу build.xml, його структурні компоненти - цілі, завдання, залежності, тощо.
4. З сайту лабораторії http://se-111.blogspot.com/ завантажити архів типового проекту для середовища Eclipse (template.zip). Ознайомитися з структурою каталогів типового проекту, XML-файлами опису проекту (.project, .classpath, build.xml).
5. Імпортувати типовий проект до робочого простору (workspace) середовища Eclipse. В пакеті com.lab111 запустити на виконання (run) клас TestMain. Ознайомитися з засобами використання ANT у середовищі Eclipse - редагування файлу build.xml, виконання цілей у відображенні (View) ANT. Виконати цілі ANT з середовища Eclipse та з командного рядка.
6. Модифікувати типовий проект для його використання у наступних лабораторних роботах. Обов’язково замінити назву і автора проекту в файлах опису проекту (.project, build.xml). Модифікувати файл build.xml таким чином, щоб у ньому були всі потрібні цілі, вільно володіти призначенням кожної цілі.
7. Розробити та перевірити в eclipse нову ціль в файлі build.xml згідно варіанту.

Варіанти завдання

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 9.

0. Створити каталог new-out. Скопіювати в цей каталог всі файли проекту з розширенням jar.
1. Видалити з проекту всі файли з розширеннями tmp, jar, class крім тих, що починаються з літери “a”.
2. Створити в каталозі out jar-архів з усіх файлів проекту з розширеннями java, js, html, htm. Скопіювати архів в корінь проекту.
3. Створити в каталозі out jar-архів з усіх файлів проекту з розширенням txt. Видалити такі файли з проекту.
4. Створити каталог build2. Виконати компіляцію сирцевих файлі проекту в створений каталог.
5. Видалити з каталогу out і нижче всі файли, що мають розширення jar. Упакувати в jar всі файли каталогу src, що починаються з літери “z”.
6. Створити **jar-архів** проекту з ім'ям, що містить дату і час створення. При створенні архіву оминати файли з розширеннями **zip** та **jar.**
7. Видалити з кореня проекту всі файли з розширенням jar. Упакувати в jar проект увесь проект окрім того, що міститься в каталозі out.
8. Виконати компіляцію тестової гілки проекту. Скомпільовані класи запакувати в jar.

Питання для самостійної перевірки

1. Призначення мови XML. Поняття словника.
2. Структура XML-документу. Види тегів.
3. Правильність XML-документу. Умови для well-formed XML- документу.
4. Структура JAR-архіву, призначення маніфесту.
5. Як створити JAR-архів, який може бути виконаний. Як запускати на виконання класи в JAR-архіві.
6. Призначення каталогів в типовому проекті.
7. Призначення файлів .project, .classpath, build.xml в типовому проекті.
8. Призначення ANT. Його відмінність від make.
9. Структура файлу build.xml. Словник XML для ANT.
10. Цілі і задачі. Алгоритм створення цілей і задач.
11. Послідовність дій ANT після запуску з командного рядка. Синтаксис запуску ANT з командного рядка.
12. Засоби ANT для роботи з файловою системою (створення/видалення каталогів, копіювання тощо).
13. Засоби ANT для компіляції сирцевих файлів Java.
14. Засоби ANT для створення JAR-архіву.
15. Засоби середовища Eclipse для роботи з ANT.

Протокол

Протокол має містити титульну сторінку, завдання, роздруківку змісту каталогу проекту з відповідними коментарями та роздруківку файлів .project, .classpath, build.xml з відповідними коментарями, результати тестування програми. До протоколу додається файл build.xml з вихідним (сирцевим) кодом.

Список рекомендованих інформаційних джерел

1. XML - Wikipedia. URL: https://en.wikipedia.org/wiki/XML.
2. XML Tutorial. URL: https://www.w3schools.com/xml.
3. Using JAR Files: The Basics. URL: https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html
4. Apache Ant - Wikipedia. URL: https://en.wikipedia.org/wiki/Apache_Ant
5. Apache ANT – A Tool For Automating Software. URL: https://www.softwaretestinghelp.com/apache-ant-selenium-tutorial-23/

ЛАБОРАТОРНА РОБОТА № 2.
ГРАФІЧНА НОТАЦІЯ UML, ДОКУМЕНТУВАННЯ ПРОЕКТУ

Тема

Графічна нотація UML, документування проекту.

Мета

Ознайомлення з видами діаграм UML. Отримання базових навичок з використання діаграми класів мови UML. Здобуття навичок з використання засобів автоматизації UML-моделювання на прикладі ArgoUML/Umbrello. Документування проекту за допомогою JavaDoc.

Короткі теоретичні відомості

UML (англ. Unified Modeling Language) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML призначена для візуалізації, проектування, моделювання та документування при розробці програмного забезпечення. UML не є мовою програмування, але існують засоби кодогенерації на основі UML. UML складається з 13 діаграм, що поділяються на структурні та поведінкові.

Структурні діаграми:

- Класів - Class diagram
- Компонент - Component diagram
- Композитної/складеної структури - Composite structure diagram (Кооперації - Collaboration diagram (UML2.0))
- Розгортування - Deployment diagram
- Об'єктів - Object diagram
- Пакетів - Package diagram

Діаграми поведінки:

- Діяльності - Activity diagram
- Скінчених автоматів (станів) - State Machine diagram
- Прецедентів - Use case diagram

Діаграми взаємодії:

- Кооперації - Collaboration (UML 1.x) / Комунікації - Communication (UML2.0)
- Огляду взаємодії - Interaction overview diagram (UML2.0)
- Послідовності - Sequence diagram
- Синхронізації - UML Timing Diagram (UML2.0)

Діаграма класів — статичне представлення структури моделі. Подає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак, їх динаміку розкрито в діаграмах інших типів.

Клас позначається прямокутником з трьох частин: верхня містить ім’я класу, середня список атрибутів класу, нижня - список операцій класу. Додатково елементи списків можуть позначатись типами та областю видимості. Відношення між класами позначається різними видами ліній та стрілок. Існують такі види відношень:

- Асоціація - показує, що об'єкти однієї сутності (класу) пов'язані з об'єктами іншої сутності. Позначається суцільною лінією. Може бути унарою - односторонньою (позначається відкритою стрілкою, що вказує напрям асоціації) та бінарною. В асоціації можуть брати участь більше двох класів, такі асоціації позначаються лініями, один кінець яких йде до класового блоку, а інший до загального ромбу. Асоціації можуть бути іменованими, тоді на кінцях її лінії позначені ролі, приналежності, індикатори, мультиплікатори, видимості або інші властивості.
- Агрегація - різновид асоціації, при відношенні між цілим і його частинами. Як тип асоціації, агрегація може бути іменованою. Агрегація не може включати відразу декілька класів. Агрегація зустрічається, коли один клас є колекцією або контейнером інших. Час існування

класів-частин не залежить від часу існування класу-цілого. Якщо контейнер буде знищений, то його вміст - ні. Графічно агрегація позначається порожнім ромбом на блоці класу-цілого і лінією, що йде від цього ромбу до класу-частини.

- Композиція - більш суворий варіант агрегації. Під час композиції має місце жорстка залежність часу існування класу-цілого та класів-часток. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно позначається як і агрегація, але з зафарбовані ромбом.
- Узагальнення (генералізація, наслідування) - показує, що один з двох зв'язаних класів (підтип) є більш конкретною формою іншого (супертипу), який називається узагальненням першого. На практиці це означає, що будь-який екземпляр підтипу є також примірником супертипу. Графічно наслідування представляється лінією з закритою стрілкою (порожнім трикутником) у супертипа. Генералізація також відома як наслідування або зв'язок типу "is a".
- Реалізація - відношення між двома елементами моделі, в якому один елемент (клієнт) реалізує поведінку, задану іншим (постачальником). Графічно реалізація представляється також як і генералізація, але з пунктирною лінією.
- Залежність - це відношення використання, при якому зміна в специфікації одного тягне за собою зміну іншого, причому протилежне не обов'язково. Графічно позначається пунктирною стрілкою, що йде від залежного елементу до того, від якого він залежить.

Javadoc - генератор документації в HTML-форматі з коментарів сирцевого коду на Java від Sun Microsystems. Javadoc - стандарт для документування класів Java. Javadoc також надає API для створення доклетів і теглетів, які дозволяють програмісту аналізувати структуру Java-додатку.

Коментарі документації застосовують для документування класів, інтерфейсів, полів (змінних), конструкторів, методів та пакетів. У кожному разі коментар повинен знаходитися перед елементом, що документується. Дескриптори Javadoc починаються з символу "@".

Завдання

1. Ознайомитись з призначенням та видами діаграм мови UML. Вивчити діаграму класів, вільно володіти елементами та відношеннями між ними. Вміти будувати діаграми класів для сирцевого коду Java, а також генерувати програмний код еквівалентний заданій діаграмі класів.
2. Побудувати діаграму класів, яка містить три інтерфейси If1, If2, If3 з методами meth1(), meth2(), meth3() та класи що їх реалізують Cl1, Cl2, Cl3, відповідно.
3. Згідно варіанту (див. нижче) реалізувати на діаграмі класів відношення генералізації та агрегації.
4. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork2. В пакеті розробити інтерфейси і класи згідно діаграми (п.3-4). Реалізація методів має виводити на консоль ім'я класу та назву методу.
5. Ознайомитись із засобами автоматизації UML-моделювання. Вміти використовувати середовища ArgoUML та Umbrello на базовому рівні для розробки діаграми класів та документування програмного забезпечення.
6. За допомогою середовища ArgoUML або Umbrello імпортувати сирцеві коди пакету com.lab111.labwork2 та перевірити відповідність побудованої діаграми класів з розробленою (п.3-4). Зберегти діаграму в каталозі документації проекту.
7. Ознайомитись з синтаксисом коментарів для засобу автоматизації документації JavaDoc. Модифікувати сирцеві коди пакету com.lab111.labwork2 додавши коментарі у форматі JavaDoc.
8. Згенерувати JavaDoc за допомогою Eclipse (меню Project) у каталог документації проекту.
9. Розробити ціль ANT для генерації JavaDoc. Згенерувати JavaDoc за допомогою розробленої цілі ANT.

Варіанти завдання

Генералізація (наслідування)

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 9.

0. If1 <-If2; If2 <- If3; Cl1 <- Cl3
1. If1 <- If3; If3 <-If2; Cl1 <- Cl2
2. If1 <- If2; If1 <-If3; Cl2 <- Cl3
3. If2 <- If1; If1 <- If3; Cl1 <- Cl3
4. If2 <- If1; If3 <- If1; Cl2 <- Cl1
5. *If2* <- If1; If2 <- If3; Cl2 <- Cl3
6. If3 <-If2; If2 <- If1; Cl2 <- Cl1
7. If3 <- If1; If1 <-If2; Cl3 <- Cl1
8. If3 <- If2; If3 <- If1; Cl3 <- Cl2

Агрегація

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 5.

0. If1 <- Cl1; Cl3 <- Cl2; Cl3 <- Cl3
1. If1 <- Cl2; Cl1 <- Cl3; Cl1 <- Cl1
2. If1 <- Cl3; Cl1 <- Cl2; Cl1 <- Cl1
3. If1 <- Cl1; If2 <- Cl1; Cl3 <- Cl2
4. If3 <- Cl2; If2 <- Cl3; Cl3 <- Cl1

Питання для самостійної перевірки

1. Призначення мови UML.
2. Коротка характеристика діаграм UML.
3. Елементи діаграми класів та відношення між ними. Унарні та бінарні відношення.
4. Різниця між асоціацією, агрегацією та композицією.
5. Відношення наслідування. Нотація на діаграмі та приклад сирцевого коду Java.
6. Відношення реалізації. Нотація на діаграмі та приклад сирцевого коду Java.
7. Відношення асоціації. Нотація на діаграмі та приклад сирцевого коду Java.
8. Відношення агрегації. Нотація на діаграмі та приклад сирцевого коду Java.
9. Відношення композиції. Нотація на діаграмі та приклад сирцевого коду Java.
10. Відношення залежності. Нотація на діаграмі та приклад сирцевого коду Java.
11. Мультиплікатори і ролі. Призначення і нотація.
12. Стереотипи. Призначення і нотація.
13. Види UML-модельєрів.
14. Призначення JavaDoc. Синтаксис JavaDoc-коментарів.
15. Засоби ANT для роботи з JavaDoc.

Протокол

Протокол має містити титульну сторінку, завдання, роздруківку діаграми класів, розроблений програмний код та генеровану документацію в форматі JavaDoc. Файли вихідного коду (*.java) повинні бути додані до протоколу.

Список рекомендованих інформаційних джерел

1. Unified Modeling Language. URL: https://en.wikipedia.org/wiki/Unified_Modeling_Language
2. Comparison of Unified Modeling Language tools. URL: http://en.wikipedia.org/wiki/List_of_UML_tools
3. Grady Booch, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language Usere Guide. Second edition. - Addison-Wesley, 2005. – 475 p.
4. Class diagram. URL: https://en.wikipedia.org/wiki/Class_diagram
5. Sequence diagram. URL: https://en.wikipedia.org/wiki/Sequence_diagram
6. ArgoUML Documentation Resources. URL: http://www.lysator.liu.se/xenofarm/argouml/work/argouml/www/documentation/index.html
7. Welcome to Umbrello - The UML Modeller. URL: http://uml.sourceforge.net/
8. Umbrello Documentation. URL: https://umbrello.kde.org/documentation.php
9. Javadoc. URL: https://en.wikipedia.org/wiki/Javadoc
10. How to Write Doc Comments for the Javadoc Tool. URL: https://www.oracle.com/cis/technical-resources/articles/java/javadoc-tool.html
11. How to generate a PDF from JavaDoc. URL: https://stackoverflow.com/questions/2322048/how-to-generate-a-pdf-from-javadoc-including-overview-and-package-summaries

ЛАБОРАТОРНА РОБОТА № 3.
СТРУКТУРНІ ШАБЛОНИ ПРОЕКТУВАННЯ.
ШАБЛОНИ COMPOSITE, DECORATOR, PROXY

Тема

Структурні шаблони проектування. Шаблони Composite, Decorator, Proxy.

Мета

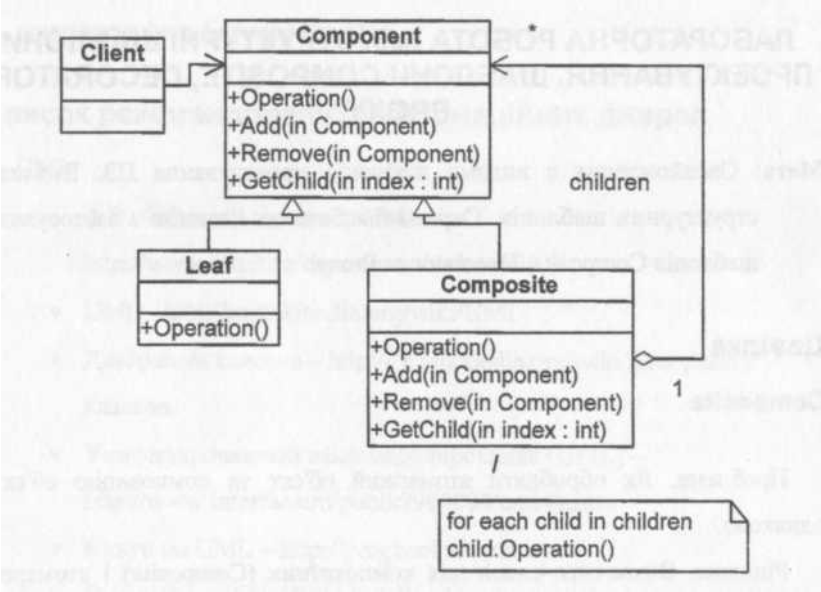
Ознайомлення з видами шаблонів проектування **ПЗ.** Вивчення структурних шаблонів. Отримання базових навичок з застосування шаблонів Composite, Decorator та Proxy.

Короткі теоретичні відомості

Шаблон Composite

Проблема. Як обробляти атомарний об'єкт та композицію об'єктів однаковим чином?

Рішення. Визначити класи для композитних (Composite) і атомарних (Leaf) об'єктів таким чином, щоб вони реалізовували один і той же інтерфейс (Component). Структура шаблону наведена на мал. 1.



Мал. 1. Структура шаблону Composite

Склад шаблону Composite такий:

1. Component – цей компонент призначений для:
   - оголошення інтерфейсу для об'єктів композиції та реалізації відповідної поведінки за замовчуванням для інтерфейсу, загального для всіх класів.
   - оголошення інтерфейсу для доступу до нащадків та управління ними.
   - визначення інтерфейсу для доступу до батьківського компонента в рекурсивній структурі та реалізації його, якщо це доречно (необов'язково).
2. Leaf – листок або пелюсток, елементарний об'єкт призначений для:
   - представлення листяних (тобто елементарних або примітивних) об'єктів в композиції. Листок не має нащадків.
   - визначення поведінки примітивних об'єктів у композиції.

3. Composite – складений об’єкт призначений для:
   - визначення поведінки компонентів, які мають нащадків.
   - зберігання компонентів-нащадків.
   - реалізації операції керування нащадками в інтерфейсі класу Component.
4. Client – клієнт, цей компонент призначений для керування об’єктами у композиції через інтерфейс Component.

Шаблон Decorator

Проблема. Покласти додаткові обов'язки (прозорі для клієнтів) на окремий об'єкт, а не на клас в цілому.

Рішення. В динаміці додати об'єкту нові обов'язки, не вдаючись при цьому до породження підкласів (наслідування). "Component" визначає інтерфейс для об'єктів, на які можуть бути в динаміці покладені додаткові обов'язки, "ConcreteComponent" визначає об'єкт, на який покладаються додаткові обов'язки, "Decorator" - зберігає посилання на об'єкт "Component" і визначає інтерфейс, відповідний інтерфейсу "Component". "ConcreteDecorator" покладає додаткові обов'язки на компонент. "Decorator" переадресує запити об'єкту "Component".

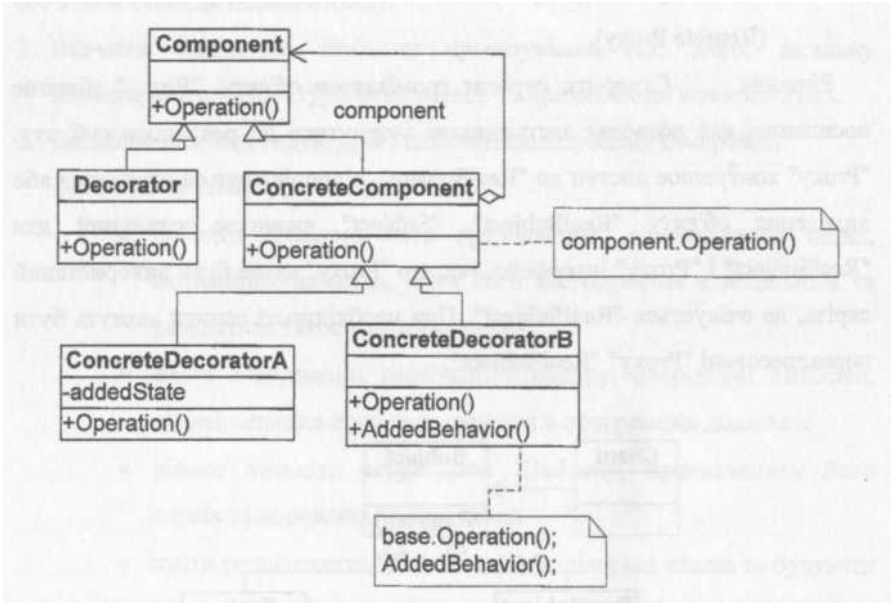
Структура шаблону наведена на мал. 2. Склад шаблону Decorator такий:

1. Component – цей компонент призначений для визначення інтерфейсу об'єктів, до яких можуть динамічним чином покладатися нові обов'язки.
2. ConcreteComponent – конкретний компонент, цей компонент призначений для визначення об’єкту, на який можуть бути покладені додаткові обов’язки.
3. Decorator – декоратор, цей компонент призначений для зберігання посилання на об'єкт Component і визначення інтерфейсу, який відповідає інтерфейсу Component.
4. ConcreteDecorator – конкретний декоратор, цей компонент призначений для додавання обов’язків компоненту.

Застосування декількох "Decorator" до одного "Component" дозволяє довільним чином поєднувати обов'язки, наприклад, одну властивість можна додати двічі.

Більша гнучкість, ніж у статичного наслідування: можна додавати і видаляти обов'язки під час виконання програми в той час як при використанні наслідування треба було б створювати новий клас для кожного додаткового обов'язку. Даний шаблон дозволяє уникнути перевантажених методами класів на верхніх рівнях ієрархії - нові обов'язки можна додавати по мірі необхідності.

"Decorator" і його "Component" не ідентичні, і, крім того, виходить, що система складається з великої кількості дрібних об'єктів, які схожі один на одного і розрізняються тільки способом взаємозв'язку, а не класом і не значеннями своїх внутрішніх змінних - така система складна у вивченні та налагодженні.



Мал. 2. Структура шаблону Decorator

Шаблон Proxy

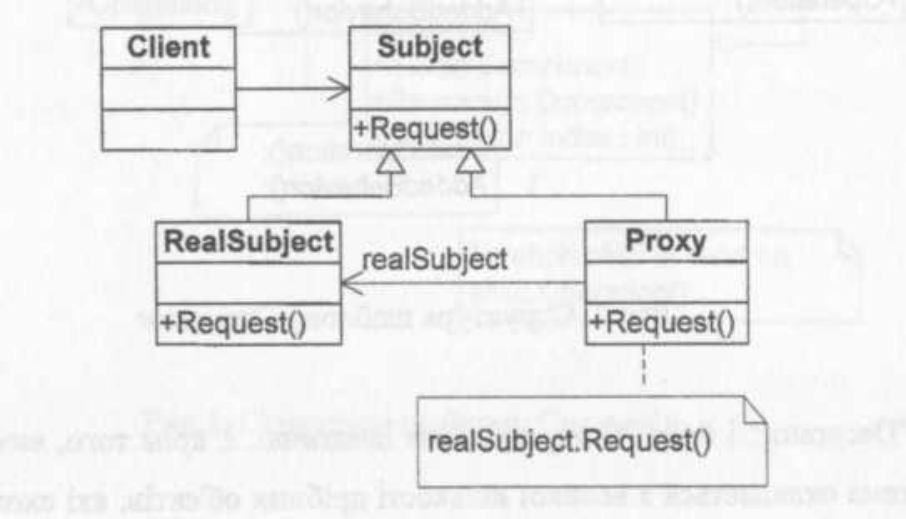
Проблема. Необхідно управляти доступом до об'єкта для таких цілей:

- створювати громіздкі об'єкти "на вимогу" (Virtual Proxy);
- кешувати дані (SmartLink Proxy);
- в динаміці обмежувати доступ (Security Proxy):
- забезпечувати доступ до об'єкта в іншому адресному просторі (Remote Proxy).

Рішення. Створити сурогат громіздкого об'єкту. "Proxy" зберігає посилання, яке дозволяє заступникові звернутися до реального суб'єкту. "Proxy" контролює доступ до "RealSubject", відповідаючи за створення або видалення об'єкту "RealSubject". "Subject" визначає загальний для "RealSubject" і "Proxy" інтерфейс, так, що "Proxy" може бути використаний скрізь, де очікується "RealSubject". При необхідності запити можуть бути переадресовані "Proxy" "RealSubject".

Структура шаблону наведена на мал.3. Склад шаблону Proxy такий:

1. Proxy (ImageProxy) – заступник, цей компонент призначений для:
   - зберігання посилання, яке дозволяє заступнику отримати доступ до реального суб’єкта. Заступник може посилатися на суб’єкт, якщо інтерфейси класів RealSubject та Subject однакові.
   - надання інтерфейсу, ідентичного суб’єкту, так що заступник може замінити справжнього суб'єкта.
   - контролю доступу до реального суб’єкта і несення відповідальності за його створення та видалення.
   - наступні обов'язки залежать від виду довіреного заступника:
     - відповідальність віддалених заступників за кодування запиту та його аргументів, а також за надсилання закодованого запиту реальному суб'єкту в іншому адресному просторі.
     - можливість віртуальних заступників кешувати додаткову інформацію про реальний суб’єкт, з метою відкласти його створення.
     - можливість заступників-захисників перевіряти наявність у абонента дозволів доступу, необхідних для виконання запиту.
2. Subject (Graphic) – суб’єкт, цей компонент призначений для визначення загального інтерфейсу для RealSubject та Proxy, з метою надання заступнику можливості бути використовуваним замість RealSubject в будь-якому місці.
3. RealSubject (Image) – реальний суб’єкт, цей компонент призначений для визначення реального об'єкту, який представляє заступник.



Мал. 3. Структура шаблону Proxy

Завдання

1. Ознайомитись з призначенням та видами шаблонів проектування ПЗ. Вивчити класифікацію шаблонів проектування ПЗ. Знати назви шаблонів, що відносяться до певного класу.
2. Вивчити структурні шаблони проектування ПЗ. Знати загальну характеристику структурних шаблонів та призначення кожного з них.
3. Детально вивчити структурні шаблони проектування Composite, Decorator та Proxy. Для кожного з них:
   - вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки, коли його застосування є доцільним та результати такого застосування;
   - знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
   - вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;
   - вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, що реалізують шаблон.
4. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork3. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.3). В класах, що розробляються, повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку, закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи. Приклад реалізації бізнес-методу:

```
void draw(int x, int y)
        {System.out.println("Метод  draw  з параметрами x="+x+" y="+y);
        }
```

5. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

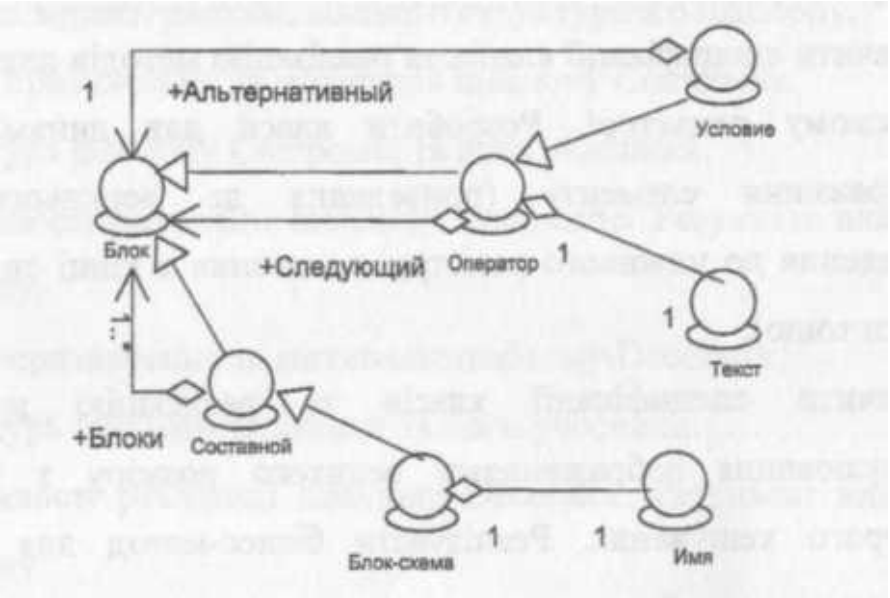
Варіанти завдання

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 12.

0. Визначити специфікації класів, які подають графічні примітиви та їх композиції у редакторі векторної графіки. Кожний примітив має атрибути розміщення - позиція (координати x та y) і розмір (ширина та висота). Реалізувати бізнес-метод відображення таких атрибутів розміщення для примітивів (задаються в конструкторі) і композицій (обчислюються динамічним чином).
1. Визначити специфікації класів, які подають дерево розбору складного виразу з лапками відповідно до синтаксичних правил:

   *<вираз>::=<простий **вираз**> | **<складний вираз>***
   <простий вираз>::=<константа> | <змінна>
   <константа>::=(<число>)
   <змінна>::=(<ім'я>)
   <складний вираз>::-(<вираз><знак операції><вираз>)
   <знак операції>::=+|-||/*

   Реалізувати бізнес-метод відображення наповнення елемента у вигляді виразу.
2. Визначити специфікації класів для подання ігрового простору з багаторівневою ієрархічною структурою. Реалізувати бізнес-метод обчислення площі, що займає елемент в умовних одиницях.
3. Визначити специфікації класів для подання блок-схем алгоритмів з блоковою організацією відповідно до семантичної діаграми (мал. 4). Реалізувати бізнес-метод ідентифікації елемента та його зв'язки з іншими елементами блок-схеми.



Мал. 4. Семантична діаграма блок-схем алгоритмів

4. Визначити специфікації класів для подання файлової системи у вигляді дерева об'єктів (файл - листовий об'єкт, каталог - вузловий). Кожний об'єкт має атрибут розміру (для файлу задається в конструкторі, для каталогів обчислюється). Реалізувати бізнес-метод отримання розміру для класу каталогу.
5. Визначити специфікації класів та реалізацію методів для подання вибраного графічного елементу у редакторі векторної графіки. Забезпечити можливість динамічної зміни відображення елементу.
6. Визначити специфікації класів додаткових графічних зображень для графічних елементів у редакторі векторної графіки. Навести приклади використання розроблених класів-обгорток.
7. Визначити специфікації класів для подання графічних маніпуляторів геометричних властивостей(положення, розмір) у редакторі векторної графіки.
8. Визначити специфікації класів та реалізацію методів для елементів в текстовому редакторі. Розробити класи для динамічної зміни відображення елементу (приведення до верхнього регістру, приведення до нижнього регістру, додавання в кінці символу нової строки тощо).
9. Визначити специфікації класів та реалізацію методів для маніпулювання зображеннями великого розміру з можливістю прозорого кешування. Реалізувати бізнес-метод для визначення кольору точки за його координатами.
10. Визначити специфікації класів та реалізацію методів для маніпулювання зображеннями з можливістю їх “пізнього завантаження”. Реалізувати бізнес-метод для визначення кольору точки за його координатами.
11. Визначити специфікації класів та реалізацію методів для маніпулювання зображеннями з можливістю контролювання доступу до об'єкта — доступ відкритий лише до точок чиї координати (x, y) лежать в межах x1<x<x2 и y1<y<y2 (значення x1, x2, y1, y2 задаються в конструкторі). Реалізувати бізнес-метод для визначення кольору точки за його координатами.

Питання для самостійної перевірки

1. Шаблони проектування програмного забезпечення. Призначення. Коротка історія створення.
2. Класифікація шаблонів проектування ПЗ.
3. Призначення структурних шаблонів проектування ПЗ.
4. Коротка характеристика кожного структурного шаблону.
5. Назви, призначення та мотивація шаблону Composite.
6. Структура шаблону Composite та його учасники.
7. Особливості реалізації шаблону Composite. Результат використання шаблону.

8. Назви, призначення та мотивація шаблону Decorator.
9. Структура шаблону Decorator та його учасники.
10. Особливості реалізації шаблону Decorator. Результат використання шаблону.
11. Назви, призначення та мотивація шаблону Proxy.
12. Структура шаблону Proxy та його учасники.
13. Особливості реалізації шаблону Proxy. Результат використання шаблону.
14. Види шаблону Proxy.
15. Шаблони, які використовуються сумісно з Composite, Decorator та Proxy.
16. Відмінність Decorator та Proxy в специфікації конструктора.

Протокол

Протокол має містити титульну сторінку (з номером залікової книжки), завдання, роздруківку діаграми класів, розроблений програмний код та генеровану документацію в форматі JavaDoc. Файли вихідного коду (*.java) повинні бути додані до протоколу.

Список рекомендованих інформаційних джерел

1. Design Patterns: elements of reusable object-oriented software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Indianapolis: - Addison-Wesley, 1994. - 417 p. ISBN: 0201633612.
2. Mark Grand. Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Volume 1, 2nd Edition. - Wiley Publishing, 2002. - 480 p. ISBN: 0471258393
3. Design Patterns. URL: https://en.wikipedia.org/wiki/Design_Patterns
4. Design Pattern – Overview. URL: https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm
5. Structural Design Patterns. URL: https://refactoring.guru/design-patterns/structural-patterns
6. Structural Design Pattern. URL: https://www.scaler.com/topics/design-patterns/structural-design-pattern/
7. What is Structural Design Pattern? URL: https://www.scaler.com/topics/design-patterns/structural-design-pattern/
8. Structural design patterns. URL: https://www.javatpoint.com/structural-design-patterns
9. Structural pattern – Wikipedia. URL: https://en.wikipedia.org/wiki/Structural_pattern
10. Structural patterns. URL: https://sourcemaking.com/design_patterns/structural_patterns/

ЛАБОРАТОРНА РОБОТА № 4.
СТРУКТУРНІ ШАБЛОНИ ПРОЕКТУВАННЯ.
ШАБЛОНИ FLYWEIGHT, ADAPTER, BRIDGE, FACADE

Тема

Структурні шаблони проектування. Шаблони Flyweight, Adapter, Bridge, Facade.

Мета

Вивчення структурних шаблонів. Отримання базових навичок з застосування шаблонів Flyweight, Adapter, Bridge, Facade.

Короткі теоретичні відомості

Шаблон Flyweight

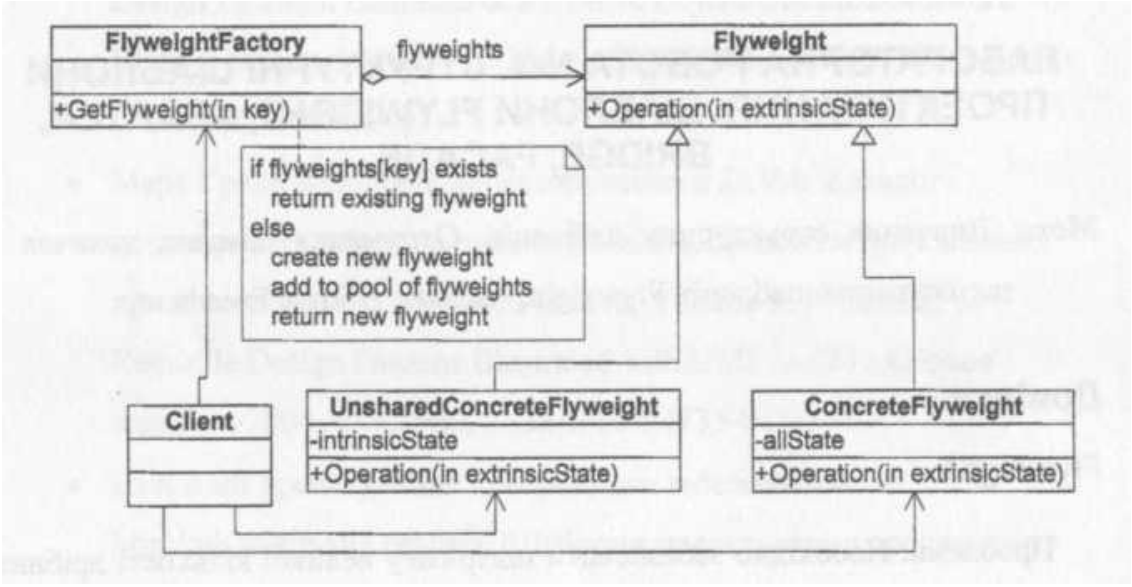
Проблема. Необхідно забезпечити підтримку великої кількості дрібних об'єктів.

Рішення. Створити об'єкт, який можна використовувати одночасно в кількох контекстах, причому, в кожному контексті об’єкт виглядає як незалежний об'єкт (не відрізняється від екземпляра, який не розділяється)."Flyweight" оголошує інтерфейс, за допомогою якого пристосуванці можуть отримати зовнішній стан або якось впливати на нього, "ConcreteFlyweight" реалізує інтерфейс класу "Flyweight" і додає за необхідності внутрішній стан. Внутрішній стан зберігається в об'єкті "ConcreteFlyweight", в той час як зовнішній стан зберігається або обчислюється в "Client" ("Client" передає його "Flyweight" при виклику операцій).

Об'єкт класу "ConcreteFlyweight" розділяється. Будь-який стан у ньому має бути внутрішнім, тобто незалежним від контексту, "FlyweightFactory" - створює об'єкти - "Flyweight" (або надає примірник, що вже існує) та керує ними. "UnsharedConcreteFlyweight" - не всі підкласи "Flyweight" обов'язково розділяються. "Client" - зберігає посилання на одного або кількох "Flyweight", обчислює і зберігає зовнішній стан "Flyweight".

Структура шаблону наведена на мал. 5. Склад шаблону Flyweight такий:

1. Flyweight – легковаговик (пристосуванець), цей компонент призначений для оголошення інтерфейсу, за допомогою якого пристосуванці можуть отримувати зовнішній стан і якось змінювати його.
2. ConcreteFlyweight – конкретний легковаговик, цей компонент призначений для реалізації інтерфейсу класа Flyweight та додавання, в разі потреби, внутрішнього стану. Об’єкт класу ConcreteFlyweight повинен бути розподіленим (тобто спільним). Будь-який стан, який він зберігає, повинен бути внутрішнім, тобто він повинен бути незалежним від контексту.
3. UnsharedConcreteFlyweight – неподільний конкретний легковаговик, цей компонент призначений для закріплення певних підкласів Flyweight, тобто не всі підкласи мають бути поділені. Інтерфейс Flyweight дозволяє поділ, але не нав’язує його. Часто об'єкти UnsharedConcreteFlyweight мають об'єкти ConcreteFlyweight в якості нащадків на якомусь рівні структури пристосуванця.
4. FlyweightFactory – фабрика легковаговиків, цей компонент призначений для створення та управління об’єктами Flyweight, а також для забезпечення належного поділу легковаговиків. Якщо клієнт запитує легковаговика, об'єкт FlyweightFactory надає існуючий екземпляр або створює новий екземпляр, якщо такий не існує.
5. Client – клієнт, цей компонент призначений для зберігання посилання на легковаговика(ів) та для обчислення або зберігання зовнішнього стану легковаговика(ів).



Мал. 5. Структура шаблону Flyweight

Пристосуванці (Flyweights) моделюють сутності, кількість яких занадто велика для подання об'єктами. Має сенс використовувати даний шаблон, якщо одночасно виконуються наступні умови:

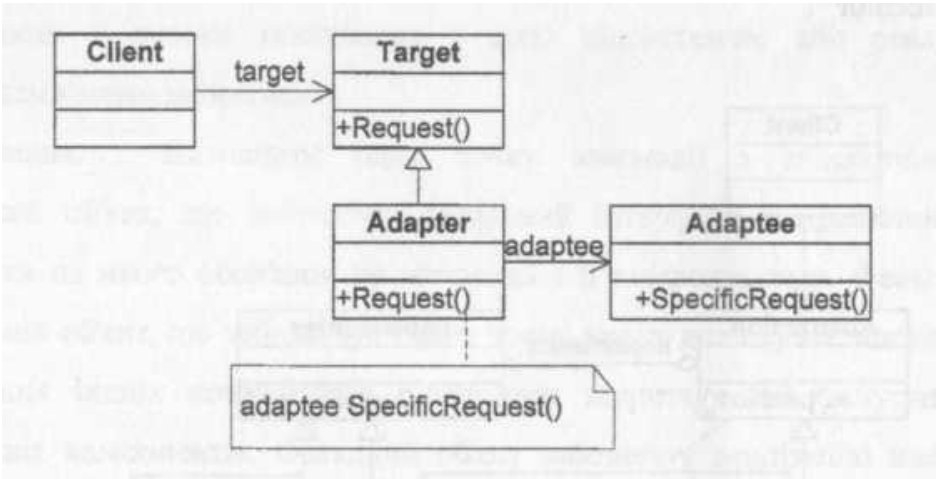
- у додатку використовується велика кількість об'єктів, завдяки чому витрати на зберігання достатньо високі,
- більшу частину стану об'єктів можна винести назовні,
- багато груп об'єктів можна замінити відносно невеликою кількістю об'єктів, оскільки стан об'єктів винесено назовні.

Внаслідок зменшення загального числа екземплярів і винесення стану заощаджується пам'ять.

Шаблон Adapter

Проблема. Необхідно забезпечити взаємодію несумісних інтерфейсів або як створити єдиний стійкий інтерфейс для декількох компонентів з різними інтерфейсами.

Рішення. Конвертувати вихідний інтерфейс компонента до іншого виду за допомогою проміжного об'єкта - адаптера, тобто, додати спеціальний об'єкт із загальним інтерфейсом в рамках даної програми і перенаправити зв'язок від зовнішніх об'єктів до цього об'єкта - адаптера. Структура шаблону наведена на мал. 6.



Мал. 6. Структура шаблону Adapter

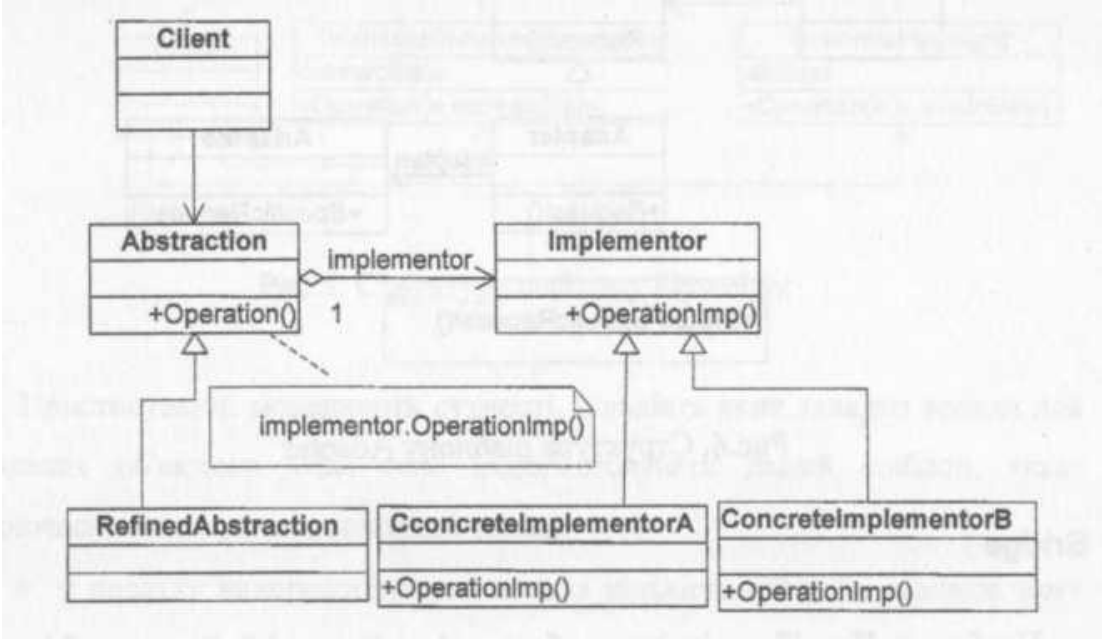
Склад шаблону Adapter такий:

1. Target – мішень, ціль, цільовий інтерфейс, цей компонент призначений для визначення специфічного для домену інтерфейсу, який використовується клієнтом.
2. Client – клієнт, цей компонент призначений для співпраці з об'єктами, що відповідають цільовому інтерфейсу.
3. Adaptee – адаптований, цей компонент призначений для визначення існуючого інтерфейсу, який потребує адаптації.
4. Adapter – адаптер, цей компонент призначений для пристосування інтерфейсу Adaptee (адаптованого) до цільового інтерфейсу.

Шаблон Bridge

Проблема. Потрібно відділити абстракцію від реалізації так, щоб і те і інше можна було змінювати незалежно. При використанні наслідування реалізація жорстко прив'язується до абстракції, що ускладнює незалежну модифікацію.

Рішення. Помістити абстракцію та реалізацію в окремі ієрархії класів. "Abstraction" визначає інтерфейс "Abstraction" і зберігає посилання на об'єкт "Implementor", "RefinedAbstraction" розширює інтерфейс, заданий у "Abstraction". "Implementor" визначає інтерфейс для класів реалізації, він не зобов'язаний точно відповідати інтерфейсу класу "Abstraction" - обидва інтерфейси можуть бути зовсім різні. Зазвичай інтерфейс класу "Implementor" надає тільки примітивні операції, а клас "Abstraction" визначає операції більш високого рівня, що базуються на цих примітивних. "ConcreteImplementor" містить конкретну реалізацію класу "Implementor". Об'єкт "Abstraction" перенаправляє запити "Client" до свого об'єкту "Implementor". Структура шаблону наведена на мал.7.



Мал. 7. Структура шаблону Bridge

Склад шаблону Bridge такий:

1. Abstraction – Абстракція, цей компонент призначений для визначення інтерфейсу абстракції та зберігання посилання на об’єкт типу Implementor.
2. RefinedAbstraction - Уточнена абстракція, цей компонент призначений для розширення інтерфейсу, який був визначений Abstraction.
3. Implementor – Реалізатор, цей компонент призначений для визначення інтерфейсу класів реалізації. Цей інтерфейс не повинен точно відповідати інтерфейсу Abstraction; насправді два інтерфейси можуть бути зовсім різними. Зазвичай інтерфейс Implementor забезпечує лише примітивні операції, а Abstraction визначає операції вищого рівня на основі цих примітивів.

4. ConcreteImplementor – Конкретний Реалізатор, цей компонент призначений для реалізації інтерфейсу Implementor та визначення його конкретної реалізації.

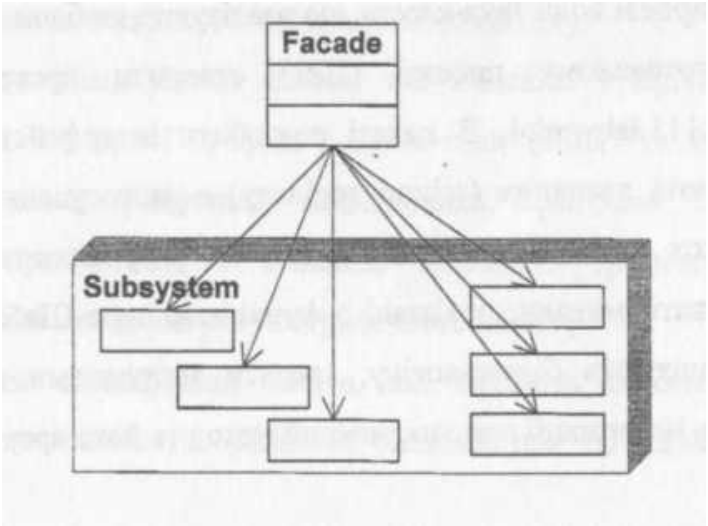
Шаблон може бути застосований якщо, наприклад, реалізацію необхідно змінювати під час роботи програми.

Відокремлення реалізації від інтерфейсу дає можливість конфігурації під час виконання "Implementor" та "Abstraction". Крім того, слід зазначити, що розподіл класів "Abstraction" та "Implementor" усуває залежності від реалізації, що встановлюються на етапі компіляції, щоб змінити клас "Implementor" зовсім не обов'язково перекомпілювати клас "Abstraction".

Шаблон Facade

Проблема. Як забезпечити уніфікований інтерфейс з набором розрізнених реалізацій або інтерфейсів, наприклад, з підсистемою, якщо небажаним є високе зв'язування з цією підсистемою або реалізація підсистеми може змінитися?

Рішення. Визначити одну точку взаємодії з підсистемою - фасадний об'єкт, що забезпечує загальний інтерфейс з підсистемою і покласти на нього обов'язок по взаємодії з її компонентами. Фасад - це зовнішній об'єкт, що забезпечує єдину точку входу для служб підсистеми. Реалізація інших компонентів підсистеми закрита і не доступна для зовнішніх компонентів. Фасадний об'єкт забезпечує реалізацію шаблону "Стійкий до змін" з точки зору захисту від змін у реалізації підсистеми. Структура шаблону наведена на мал. 8.



Мал.8. Структура шаблону Facade

Склад шаблону Facade такий:

1. Facade – фасад, цей компонент призначений для збереження та приховування інформації щодо того, які класи системи відповідають за запит; також цей компонент призначений для делегування запитів клієнтів відповідним об'єктам системи.
2. Subsystem classes – класи підсистем, ці компоненти призначені для:
   - реалізації функціональності підсистем.
   - виконання роботи, яка призначена об'єктом "Фасад".
   - ізолювання від фасаду, тобто вони не зберігають жодних посилань на фасад.

Завдання

1. Закріпити призначення шаблонів проектування ПЗ, їх класифікацію. Знати назву і коротку характеристику кожного з шаблонів, що відносяться до певного класу.
2. Повторити структурні шаблони проектування ПЗ. Знати загальну характеристику структурних шаблонів та призначення кожного з них.

3. Детально вивчити структурні шаблони проектування Flyweight, Adapter, Bridge, Facade. Для кожного з них:
   - вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки, коли його застосування є доцільним, та результати такого застосування;
   - знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
   - вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;
   - вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, які реалізують шаблон.
4. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork4. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.3). У класах, що розробляються, повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку, закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи.
5. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

Варіанти завдання

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 11.

0. Визначити специфікації класів та реалізацію методів для об'єктів-гліфів латинського алфавіту та об'єктів-строк. Об'єкти-гліфи мають атрибути координат та використовуються в якості складових при побудові композитних об'єктів-строк. Забезпечити ефективне використання пам'яті при роботі з великою кількістю об'єктів-гліфів. Реалізувати метод виводу рядка.
1. Визначити специфікації класів, які подають графічні об'єкти у редакторі растрової графіки - примітиви (точка) та їх композиції (прямокутне зображення). Забезпечити ефективне використання пам'яті при роботі з великою кількістю графічних об'єктів. Реалізувати метод рисування графічного об'єкту.
2. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки - примітиви (лінія) та їх композиції (прямокутник, трикутник). Забезпечити ефективне використання пам'яті при роботі з великою кількістю графічних об'єктів. Реалізувати метод рисування графічного об'єкту.
3. Визначити специфікації класів, які подають об'єкти-іконки для зображення елементів файлової системи при побудові графічного інтерфейсу користувача (GUI) - примітиви (файли) та їх композиції (директорії). Забезпечити ефективне використання пам'яті при роботі з великою кількістю графічних об'єктів. Реалізувати метод рисування графічного об'єкту.
4. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки - примітиви (точка) та їх композиції (коло). Примітиви мають атрибути координат в декартовій системі, а об'єкти-композиції — в полярній. Відповідно інтерфейс точки містить методи setX(int) та setY(int), а метод малювання кола може оперувати лише методами setRo(double), setPhi(double) примітива (які відсутні в класі точки). Забезпечити можливість використання функціональності точки при малюванні кола без зміни інтерфейсу точки та методу малювання кола.
5. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки - примітиви (точка) та їх композиції (лінія). Примітиви мають цілочислові атрибути координат (в пікселях), а об'єкти-композиції — раціональні (в сантиметрах). Відповідно інтерфейс точки містить методи setX(int) та setY(int), а метод малювання лінії може оперувати лише методами setX(double), setY(double) примітива (які відсутні в класі точки). Забезпечити можливість використання функціональності точки при малюванні лінії без зміни інтерфейсу точки та методу малювання лінії.

6. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки - примітиви (лінія) та їх композиції (прямокутник). Примітиви мають атрибути координат в системі з центром координат в лівому верхньому куті екрана з інверсною віссю абсцис, а об'єкти-композиції — з центром координат посередині екрана і стандартним напрямком осі абсцис. Забезпечити можливість використання функціональності лінії при малюванні прямокутника без зміни методів точки та методу малювання лінії.
7. Визначити специфікації класів, які подають елементи графічного інтерфейсу користувача (GUI) — кнопка, вікно, тощо. Забезпечити розділення абстракції і реалізації таким чином, щоб елементи інтерфейсу могли мати реалізації для різних бібліотек (наприклад, Qt та GTK) прозорі для користувача. Реалізувати метод малювання елементу.
8. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки (прямокутник) через різні інтерфейси API1 та API2. Забезпечити прозору для користувача можливість заміни реалізації графічних об'єктів. Реалізувати метод малювання елементу.
9. Визначити специфікації класів, які подають об'єкти для маніпулювання елементами файлової системи - файлами та директоріями. Інтерфейс файлу містить методи open(String path, boolean createIfNotExist), close() та delete(String path) для відкриття, закриття та видалення файлу (при createIfNotExist=true файл буде створений, якщо він не існує або обрізаний до нульової довжини, якщо існує). Інтерфейс директорії містить методи create(String path), та rmdir(String path) для створення та видалення директорії. Задати підсистему з 3-ох файлів та 2-х директорій. Забезпечити можливість створення та видалення такої підсистеми через методи create(), destroy() та зміни структури підсистеми без впливу на її користувача.
10. Визначити специфікації класів, які подають графічні об'єкти у редакторі векторної графіки — лінія (Line) та растрове зображення (Image). Інтерфейс лінії містить метод setOpacity(double op), що регулює рівень її непрозорості між повністю непрозорою (op == 1.0) та невидимою (op == 0.0). Інтерфейс растрового зображення містить метод setTransparency (double tr), що регулює рівень її прозорості між повністю прозорою (tr == 1.0) та повністю непрозорою (tr == 0.0). Задати підсистему з зображення та ліній, які його обрамляють. Забезпечити можливість вмикання/вимикання відображення підсистеми через метод show(boolean vis), та зміни типу обрамлення (горизонтальне, вертикальне, повне, тощо) без впливу на користувача такої підсистеми.

Питання для самостійної перевірки

1. Класифікація шаблонів проектування ПЗ.
2. Призначення структурних шаблонів проектування ПЗ.
3. Коротка характеристика кожного структурного шаблону.
4. Назви, призначення та мотивація шаблону Flyweight.
5. Структура шаблону Flyweight та його учасники.
6. Особливості реалізації шаблону Flyweight. Результат використання шаблону.
7. Назви, призначення та мотивація шаблону Adapter.
8. Структура шаблону Adapter та його учасники.
9. Особливості реалізації шаблону Adapter. Результат використання шаблону.
10. Назви, призначення та мотивація шаблону Bridge.
11. Структура шаблону Bridge та його учасники.
12. Особливості реалізації шаблону Bridge. Результат використання шаблону.
13. Назви, призначення та мотивація шаблону Facade.
14. Структура шаблону Facade та його учасники.
15. Особливості реалізації шаблону Facade. Результат використання шаблону.
16. Відмінність Adapter, Decorator та Proxy в специфікації конструктора.
17. Види адаптерів. Двосторонній та динамічний (pluggable) адаптери.
18. Шаблони, які використовуються сумісно з Flyweight, Adapter, Bridge, Facade.

Протокол

Протокол має містити титульну сторінку (з номером залікової книжки), завдання, роздруківку діаграми класів, розроблений програмний код та згенеровану документацію в форматі JavaDoc. Файли вихідного коду (*.java) повинні бути додані до протоколу.

Список рекомендованих інформаційних джерел

1. Design Patterns: elements of reusable object-oriented software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Indianapolis: - Addison-Wesley, 1994. - 417 p. ISBN: 0201633612.
2. Mark Grand. Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Volume 1, 2nd Edition. - Wiley Publishing, 2002. - 480 p. ISBN: 0471258393
3. Design Patterns. URL: https://en.wikipedia.org/wiki/Design_Patterns
4. Design Pattern – Overview. URL: https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm
5. Structural Design Patterns. URL: https://refactoring.guru/design-patterns/structural-patterns
6. Structural Design Pattern. URL: https://www.scaler.com/topics/design-patterns/structural-design-pattern/
7. What is Structural Design Pattern? URL: https://www.scaler.com/topics/design-patterns/structural-design-pattern/
8. Structural design patterns. URL: https://www.javatpoint.com/structural-design-patterns
9. Structural pattern – Wikipedia. URL: https://en.wikipedia.org/wiki/Structural_pattern
10. Structural patterns. URL: https://sourcemaking.com/design_patterns/structural_patterns/

ЛАБОРАТОРНА РОБОТА № 5. ШАБЛОНИ ПОВЕДІНКИ. ШАБЛОНИ ITERATOR, MEDIATOR, OBSERVER

Тема

Шаблони поведінки. Шаблони Iterator, Mediator, Observer.

Мета

Вивчення шаблонів поведінки. Отримання базових навичок з застосування шаблонів Iterator, Mediator та Observer.

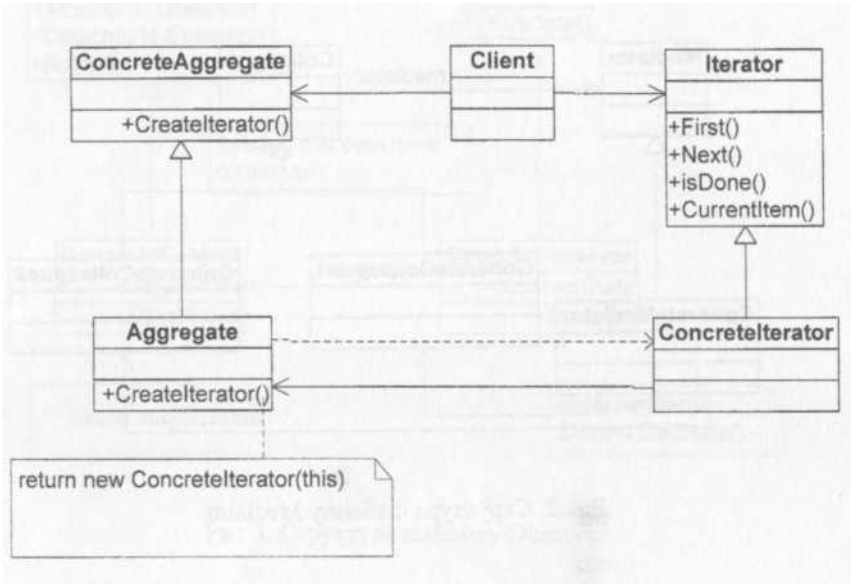
Короткі теоретичні відомості

Шаблон Iterator

Проблема. Композитний об'єкт, наприклад, список, повинен надавати доступ до своїх елементів (об'єктів), не розкриваючи їх внутрішню структуру, причому перебирати список потрібно по-різному в залежності від завдання.

Рішення. Створюється клас "Iterator", який визначає інтерфейс для доступу і перебору елементів, "Concretelterator" реалізує інтерфейс класу "Iterator" і стежить за поточною позицією при обході "Aggregate"."Aggregate" визначає інтерфейс для створення об'єкту - ітератора. "ConcreteAggregate" реалізує інтерфейс створення ітератора і повертає екземпляр класу "Concretelterator", "Concretelterator" відстежує поточний об'єкт в агрегаті і може обчислити наступний об'єкт при переборі.

Шаблон підтримує різні способи перебору агрегату, одночасно можуть бути активні кілька переборів. Структура шаблону наведена на мал. 9.



Мал. 9. Структура шаблону Iterator

Склад шаблону Iterator такий:

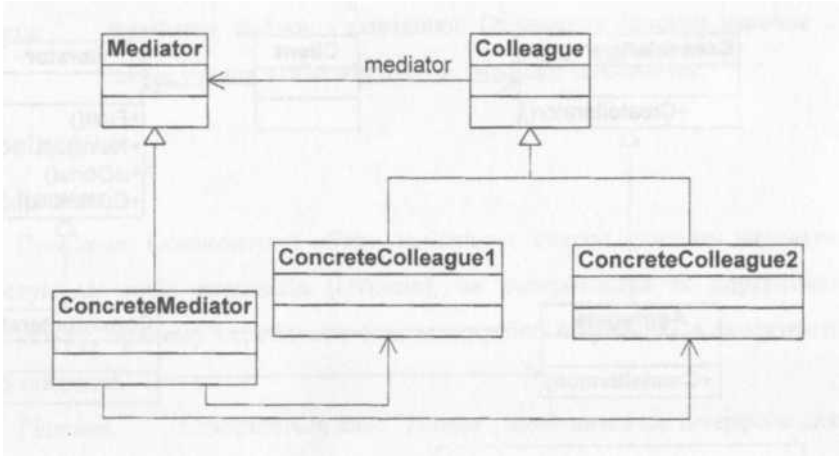
1. Iterator - ітератор, цей компонент призначений для визначення інтерфейсу доступу та обходу елементів.
2. ConcreteIterator – конкретний ітератор, цей компонент призначений для реалізації інтерфейсу Iterator та відстежування поточної позиції при обході агрегату.
3. Aggregate - агрегат (сукупність) , цей компонент призначений для визначення інтерфейсу створення об'єкта Iterator.

4. ConcreteAggregate – конкретний агрегат, цей компонент призначений для реалізації інтерфейсу створення Iterator для повернення екземпляра відповідного ConcreteIterator.

Шаблон Mediator

"Mediator" визначає інтерфейс для обміну інформацією з об'єктами "Colleague", "ConcreteMediator" координує дії об'єктів "Colleague". Кожен клас "Colleague" знає про свій об'єкт "Mediator", всі "Colleague" обмінюються інформацією тільки з посередником, при його відсутності їм довелося б обмінюватися інформацією безпосередньо. "Colleague" посилають запити посередникові та отримують запити від нього. "Mediator" реалізує кооперативну поведінку, пересилаючи кожен запит одному або декільком "Colleague". Шаблон усуває зв'язаність між "Colleague", централізуючи управління.

Структура шаблону наведена на мал. 10.



Мал. 10. Структура шаблону Mediator

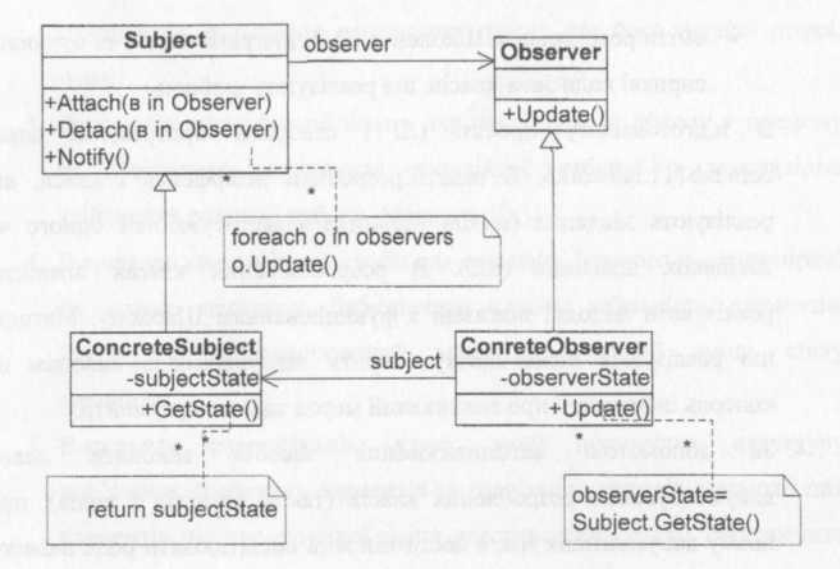
Склад шаблону Mediator такий:

1. Mediator – посередник, цей компонент призначений для визначення інтерфейсу спілкування з об'єктами Colleague.
2. ConcreteMediator – конкретний посередник, цей компонент призначений для реалізації поведінки спілкування за рахунок координації об'єктів Colleague та зберігання інформації про своїх Colleague.
3. Colleague - клас колег, цей компонент призначений для визначення класа Colleague, який відповідає об’єкту посередник, та забезпечення спілкування кожного Colleague зі своїм посередником, забороняючи спілкування з іншими об’єктами.

Шаблон Observer

Проблема. Один об'єкт ("Observer", з англ. спостерігач) повинен знати про зміну станів або деякі події іншого об'єкта. При цьому необхідно підтримувати низький рівень зв'язування з об'єктом - "Observer".

Рішення. Визначити інтерфейс "Observer". Об'єкти "ConcreteObserver" - передплатники реалізують цей інтерфейс та динамічним чином реєструються для отримання інформації про деяку подію в "Subject". Потім при реалізації обумовленої події в "ConcreteSubject" сповіщаються всі об'єкти передплатники. Структура шаблону наведена на мал. 11.



Мал. 11. Структура шаблону Observer

Шаблон Observer має такий склад:

1. Subject - суб'єкт, цей компонент призначений для збереження інформації про своїх спостерігачів (кількість спостерігачів за суб'єктом не обмежена) та забезпечення інтерфейсу приєднання та від'єднання об'єктів Observer.
2. Observer – спостерігач, цей компонент призначений для визначення інтерфейсу оновлення для об’єктів, які слід повідомляти про зміни в суб'єкті.
3. ConcreteSubject – конкретний суб’єкт, цей компонент призначений для зберігання стану, який є цікавим для об’єктів ConcreteObserver, та надсилання повідомлення своїм спостерігачам про зміну стану.
4. ConcreteObserver – конкретний спостерігач, цей компонент призначений для:
   - підтримки посилання на об’єкт ConcreteSubject;
   - зберігання даних, які відповідають даним суб’єкта;
   - реалізації інтерфейсу оновлення, який визначено у класі Observer, з метою підтримки його стану у відповідності з суб’єктом.

Завдання

1. Вивчити шаблони поведінки для проектування ПЗ. Знати загальну характеристику шаблонів поведінки та призначення кожного з них.
2. Детально вивчити шаблони поведінки для проектування ПЗ - Iterator, Mediator та Observer. Для кожного з них:
   - вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки коли його застосування є доцільним та результати такого застосування;
   - знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
   - вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;
   - вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, що реалізують шаблон.
3. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork5. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.2). В розроблюваних класах повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи.
4. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

Варіанти завдання

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 9.

0. Визначити специфікації класів, які інкапсулюють лінійний список об’єктів та реалізують можливість послідовного обходу у прямому та зворотному напрямках оминаючи порожні елементи цієї структури та не розкриваючи її сутності перед користувачем.
1. Визначити специфікації класів, які інкапсулюють лінійний список цілих чисел та реалізують можливість звичайного послідовного обходу та послідовного обходу в упорядкованій структурі.
2. Визначити специфікації класів які інкапсулюють лінійний список символьних рядків та реалізують можливість звичайного послідовного обходу та обходу з додатковою фільтрацію агрегату (Наприклад фільтрація по довжині рядка, по його першій літері, тощо).
3. Визначити специфікації класів для послідовного обходу у прямому та зворотному напрямках реляційної таблиці з можливістю здійснення операції вибору (фільтрації).
4. Визначити специфікації класів для елементу ігрового поля (комірки) та самого простору. Забезпечити слабку зв'язаність елементів. Реалізувати централізований механізм сумісної зміни стану елементів.
5. Визначити специфікацію класу, який інкапсулює структуру пов’язаних графічних елементів та реалізацію методів взаємодії цих елементів під час сумісної зміни властивостей (колір). Забезпечити слабку зв'язаність елементів.
6. Визначити специфікації класів для подання реляційної таблиці та обмеження зовнішнього ключа з можливістю його перевірки під час зміни значень полів. Забезпечити слабку зв'язаність елементів.
7. Визначити специфікації класів для подання елементів графічного інтерфейсу користувача — GUI (вікна, кнопки, текстові області). Реалізувати механізм реакції на події в будь-якому з елементів.
8. Визначити специфікації класів для подання реляційної таблиці. Реалізувати механізм тригерів — виконання додаткових дій при зміні елемента.

Питання для самостійної перевірки

1. Класифікація шаблонів проектування ПЗ.
2. Призначення шаблонів поведінки для проектування ПЗ.
3. Коротка характеристика кожного шаблону поведінки.
4. Назви, призначення та мотивація шаблону Iterator.
5. Структура шаблону Iterator та його учасники.
6. Особливості реалізації шаблону Iterator. Результат використання шаблону.
7. Назви, призначення та мотивація шаблону Mediator.
8. Структура шаблону Mediator та його учасники.
9. Особливості реалізації шаблону Mediator. Результат використання шаблону.
10. Назви, призначення та мотивація шаблону Observer.
11. Структура шаблону Observer та його учасники.
12. Особливості реалізації шаблону Observer. Результат використання шаблону.
13. Шаблони, які використовуються сумісно з Iterator, Mediator та Observer.

Протокол

Протокол має містити титульну сторінку (з номером залікової книжки), завдання, роздруківку діаграми класів, розроблений програмний код та згенеровану документацію в форматі JavaDoc. Файли вихідного коду (*.java) повинні бути додані до протоколу.

Список рекомендованих інформаційних джерел

1. Design Patterns: elements of reusable object-oriented software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Indianapolis: - Addison-Wesley, 1994. - 417 p. ISBN: 0201633612.
2. Mark Grand. Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Volume 1, 2nd Edition. - Wiley Publishing, 2002. - 480 p. ISBN: 0471258393
3. Design Patterns. URL: https://en.wikipedia.org/wiki/Design_Patterns
4. Design Pattern – Overview. URL: https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm
5. Behavioral pattern - Wikipedia. URL: https://en.wikipedia.org/wiki/Behavioral_pattern#:~:text=In%20software%20engineering%2C%20behavioral%20design,flexibility%20in%20carrying%20out%20communication
6. Behavioral Design Patterns. URL: https://refactoring.guru/design-patterns/behavioral-patterns
7. Behavioral patterns. URL: https://sourcemaking.com/design_patterns/behavioral_patterns
8. Behavioral Design Patterns. URL: https://www.javatpoint.com/behavioral-design-patterns
9. Behavioral Design Patterns. URL: https://programmingline.com/software-design-patterns/behavioral-design-patterns
10. Behavioral Patterns. URL: https://howtodoinjava.com/design-patterns/behavioral/

ЛАБОРАТОРНА РОБОТА № 6. ШАБЛОНИ ПОВЕДІНКИ. ШАБЛОНИ STRATEGY, CHAIN OF RESPONSIBILITY, VISITOR

Тема

Шаблони поведінки. Шаблони Strategy, Chain of Responsibility, Visitor.

Мета

Вивчення шаблонів поведінки. Отримання базових навичок з застосування шаблонів Strategy, Chain of Responsibility, Visitor.

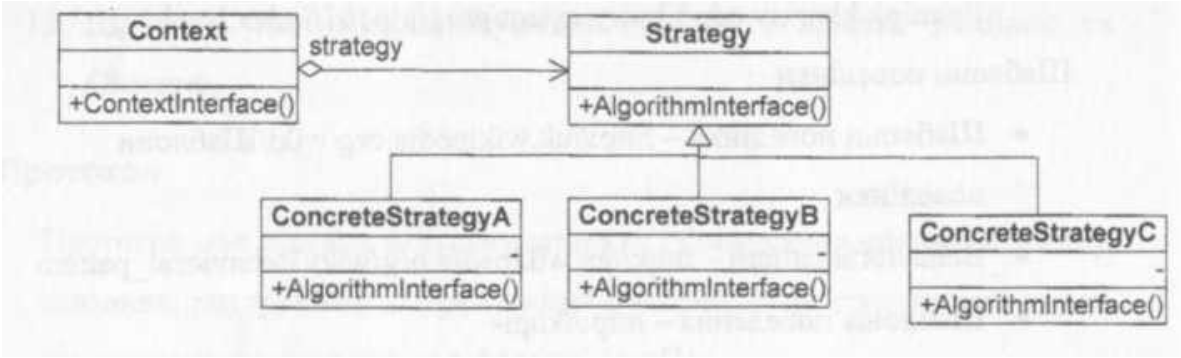
Короткі теоретичні відомості

Шаблон Strategy

Проблема. Як спроектувати змінювані, але надійні алгоритми або стратегії.

Рішення. Визначити для кожного алгоритму або стратегії окремий клас зі стандартним інтерфейсом “Strategy”.

Створюється декілька класів “ConcreteStrategy”, кожен з яких містить один і той же поліморфний метод "AlgorithmInterface". Об'єкт стратегії зв'язується з контекстним об'єктом (тим об'єктом, до якого застосовується алгоритм). Структура шаблону наведена на мал. 12.



Мал. 12. Структура шаблону Strategy

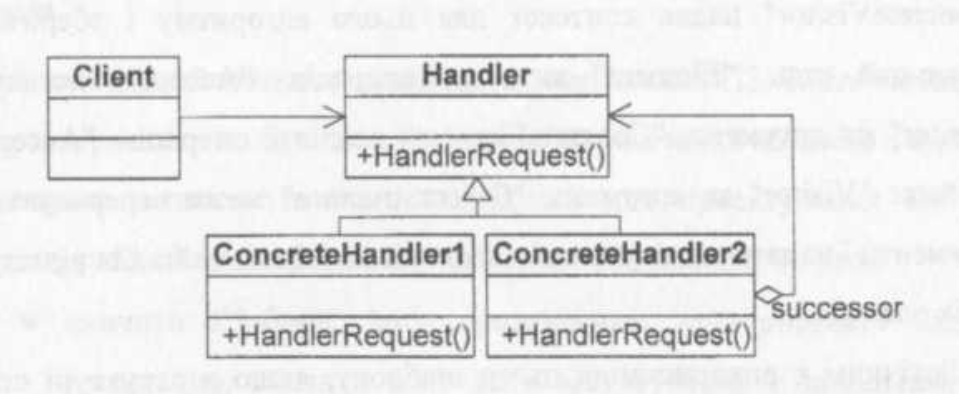
Шаблон Strategy має такий склад:

1. Strategy - стратегія, цей компонент призначений для оголошення інтерфейсу, загального для всіх підтримуваних алгоритмів. Клас Контекст використовує цей інтерфейс для виклику алгоритму, визначеного у ConcreteStrategy.
2. ConcreteStrategy - конкретна стратегія, цей компонент призначений для реалізації алгоритму, який був декларований у інтерфейсі Strategy.
3. Context - контекст, цей компонент призначений для:
   - налаштування на об'єкт ConcreteStrategy;
   - зберігання посилання на об’єкт Стратегії;
   - визначення інтерфейсу, який дозволяє стратегії отримувати доступ до своїх даних.

Шаблон Chain of Responsability

Проблема. Запит повинен бути оброблений декількома об'єктами.

Рішення. Зв'язати об'єкти - одержувачі запиту в ланцюжок і передавати запит вздовж цього ланцюжка, поки він не буде оброблений. "Handler" визначає інтерфейс для обробки запитів, і, можливо, реалізує зв'язок з наступником. "ConcreteHandler" обробляє запит, за який відповідає. Маючи доступ до свого наступника "ConcreteHandler" направляє запит до нього, якщо не може обробити запит сам. Структура шаблону наведена на мал. 13.



Мал.13. Структура шаблону Chain of Responsability

Склад шаблона Chain of responsibility (з англ. ланцюжок відповідальності) такий:

1. Handler - Обробник, цей компонент призначений для визначення інтерфейсу обробки запитів та реалізації посилання наступника (додатково).
2. ConcreteHandler – конкретний обробник, цей компонент призначений для:
   - обробки запитів, за які відповідає;
   - отримання доступу до свого наступника;
   - пересилання запиту своєму наступнику в разі, якщо ConcreteHandler не може обробляти цей запит.
3. Client - клієнт, цей компонент призначений для ініціації запиту до об'єкта ConcreteHandler на ланцюжку.

Логічним є застосування цього шаблону, якщо є більше одного об'єкта, здатного обробити запит і обробник заздалегідь невідомий (і повинен бути знайдений автоматично) або якщо весь набір об'єктів, які здатні обробити запит, повинен задаватися автоматично.

Шаблон послаблює зв'язаність (об'єкт не зобов'язаний "знати", хто саме опрацює його запит). Але немає гарантій, що запит буде оброблений, оскільки він не має явного одержувача.

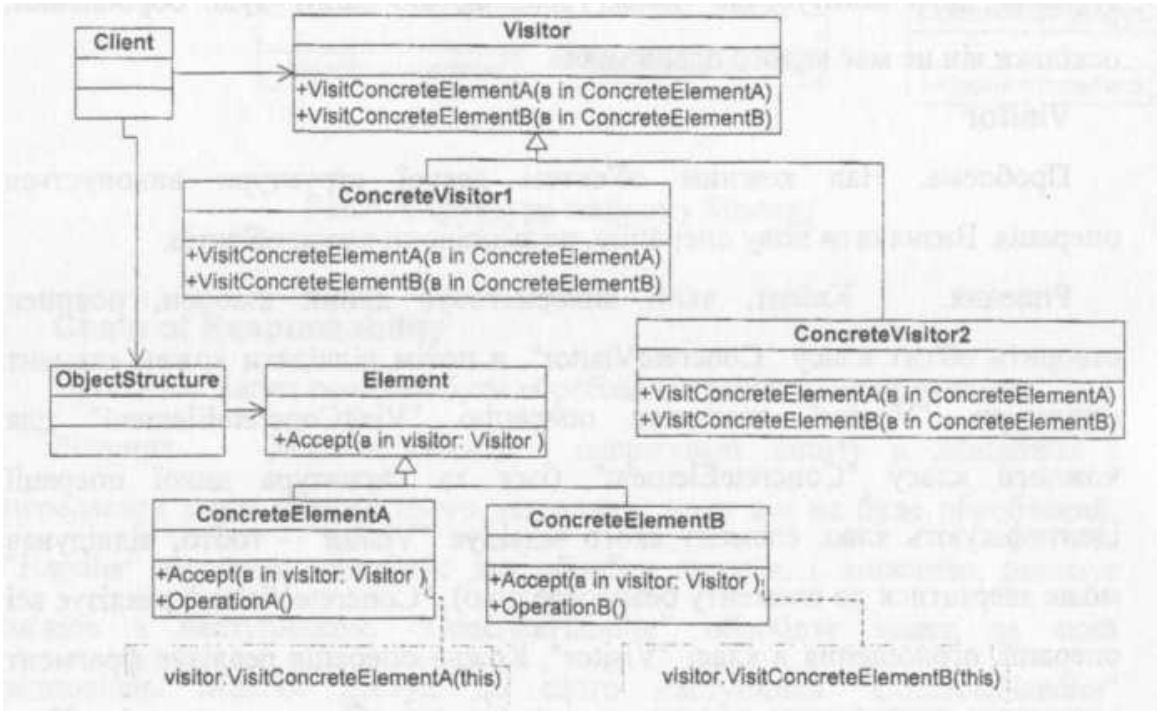
Шаблон Visitor

Проблема. Над кожним об'єктом деякої структури виконується операція. Визначити нову операцію, не змінюючи класи об'єктів.

Рішення. Клієнт, який використовує даний шаблон, повинен створити об'єкт класу "ConcreteVisitor", а потім відвідати кожен елемент структури. "Visitor" оголошує операцію "VisitConcreteElement" для кожного класу "ConcreteElement" (ім'я та сигнатура даної операції ідентифікують клас, елемент якого відвідує "Visitor" - тобто, відвідувач може звертатися до елементу безпосередньо). "ConcreteVisitor" реалізує всі операції, оголошення в класі "Visitor". Кожна операція реалізує фрагмент алгоритму, визначеного для класу відповідного об'єкта в структурі. Клас "ConcreteVisitor" надає контекст для цього алгоритму і зберігає його локальний стан. "Element" визначає операцію "Accept", яка приймає "Visitor" як аргумент, "ConcreteElement" реалізує операцію "Accept", яка приймає "Visitor" як аргумент. "ObjectStructure" може перерахувати свої аргументи і надати відвідувачеві високорівнева інтерфейс для відвідування своїх елементів.

Логічним є використання цього шаблону, якщо в структурі присутні об'єкти багатьох класів з різними інтерфейсами, і необхідно виконати над ними операції, що залежать від конкретних класів, або якщо класи, що визначають структуру об'єктів змінюються рідко, але нові операції над цією структурою додаються часто.

Шаблон спрощує додавання нових операцій, об'єднує споріднені операції в класі "Visitor". При цьому ускладняється додавання нових класів "ConcreteElement", оскільки потрібно оголошення нової абстрактної операції в класі "Visitor". Структура шаблону наведена на мал. 14.



Мал. 14. Структура шаблону Visitor

Шаблон Visitor має такий склад:

1. Visitor – відвідувач, цей компонент призначений для оголошення операції Visit для кожного класу ConcreteElement в структурі об'єкта. Ім'я та сигнатура операції визначають клас, який надсилає відвідувачу запит на відвідування. Це дозволяє відвідувачеві отримати доступ до елемента безпосередньо через його конкретний інтерфейс.
2. ConcreteVisitor - конкретний відвідувач, цей компонент призначений для реалізації всіх операцій, які були заявлені відвідувачем (Visitor). Кожна операція реалізує фрагмент алгоритму, визначений для відповідного класу об'єкта в структурі. ConcreteVisitor надає контекст алгоритму та зберігає його локальний стан. Цей стан часто накопичує результати під час обходу структури.
3. Element – елемент, цей компонент призначений для визначення операції Accept, яка приймає відвідувача як аргумент.
4. ConcreteElement - конкретний елемент, цей компонент призначений для реалізації операції Accept, яка приймає відвідувача як аргумент.
5. ObjectStructure - структура об’єктів, цей компонент призначений для:
   - перерахування своїх елементів.
   - забезпечення інтерфейсу високого рівня, що дозволяє відвідувачеві відвідувати свої елементи.
   - можливості створювати складовий об'єкт (композит або колекцію, такі як список чи множина).

Завдання

1. Повторити шаблони поведінки для проектування ПЗ. Знати загальну характеристику шаблонів поведінки та призначення кожного з них.
2. Детально вивчити шаблони поведінки для проектування ПЗ — Strategy, Chain of Responsibility та Visitor. Для кожного з них:
   - вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки коли його застосування є доцільним та результати такого застосування;
   - знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
   - вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;

- вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, що реалізують шаблон.

3. В підготованому проекті (ЛР1) створити програмний пакет com.lablll.labwork6. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.2). В розроблюваних класах повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи.
4. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

Варіанти завдання

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 10.

0. Визначити специфікації класу, який містить масив цілих чисел та метод його сортування. Забезпечити можливість динамічної зміни алгоритму та напрямку сортування шляхом зовнішньої параметризації.
1. Визначити специфікації класу, який містить таблицю та метод її відображення у вигляді діаграми. Забезпечити можливість динамічної зміни типу діаграми шляхом зовнішньої параметризації.
2. Визначити специфікації класу, який містить математичну функцію та метод її відображення у вигляді графіка. Забезпечити можливість динамічної зміни системи координат графіка (декартова, полярна тощо) шляхом зовнішньої параметризації.
3. Визначити специфікації класів, що реалізують контейнери для цілих чисел та текстових строк з можливістю їх сортування. Забезпечити можливість динамічної зміни алгоритму сортування шляхом зовнішньої параметризації. Реалізація алгоритму сортування має бути незалежною від типу даних, що сортуються.
4. Визначити специфікації класів, що реалізують елементи графічного інтерфейсу користувача — панелі (композит) та кнопки (компонент). Реалізувати децентралізований механізм обробки події переміщення курсору миші. Кількість компонентів інтерфейсу, які реагують на цю подію, може змінюватись динамічним чином.
5. Визначити специфікації класів, що реалізують обробку HTTP-запитів різних типів (наприклад GET та POST). Реалізувати можливість динамічної зміни кількості обробників. Забезпечити децентралізацію та слабку зв'язаність обробників.
6. Визначити специфікації класів для елементу ігрового поля (комірки) та самого простору. Забезпечити слабку зв'язаність елементів. Реалізувати децентралізований механізм сумісної зміни стану елементів.
7. Визначити специфікації класів, що реалізують елементи графічного інтерфейсу користувача — панелі (композит) та кнопки (компонент). Реалізувати механізм додаткових операцій над структурою графічного інтерфейсу без зміни її елементів. В якості ілюстрації такого механізму розробити операцію підрахунку кількості елементів одного типу.
8. Визначити специфікації класів, що реалізують елементи структури комп'ютера (процесор, пам'ять, відеокарта тощо). Реалізувати механізм додаткових операцій над структурою комп'ютера без зміни її елементів. В якості ілюстрації такого механізму розробити операцію визначення потужності, що потребляє комп'ютер.
9. Визначити специфікації класів, що реалізують елементи мережевої структури (кабель, сервер, робоча станція). Реалізувати механізм додаткових операцій над мережевою структурою без зміни її елементів. В якості ілюстрації такого механізму розробити операцію визначення кошторису такої структури.

Питання для самостійної перевірки

1. Класифікація шаблонів проектування ПЗ.
2. Призначення шаблонів поведінки для проектування ПЗ.
3. Коротка характеристика кожного шаблону поведінки.
4. Назви, призначення та мотивація шаблону Strategy.
5. Структура шаблону Strategy та його учасники.
6. Особливості реалізації шаблону Strategy. Результат використання шаблону.
7. Назви, призначення та мотивація шаблону Chain of Responsibility.
8. Структура шаблону Chain of Responsibility та його учасники.
9. Особливості реалізації шаблону Chain of Responsibility. Результат використання шаблону.
10. Назви, призначення та мотивація шаблону Visitor.
11. Структура шаблону Visitor та його учасники.
12. Особливості реалізації шаблону Visitor. Результат використання шаблону.
13. Шаблони, які використовуються сумісно з Strategy, Chain of Responsibility та Visitor.

Протокол

Протокол має містити титульну сторінку (з номером залікової книжки), завдання, роздруківку діаграми класів, розроблений програмний код та згенеровану документацію в форматі JavaDoc. Файли вихідного коду (*.java) повинні бути додані до протоколу.

Список рекомендованих інформаційних джерел

1. Design Patterns: elements of reusable object-oriented software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Indianapolis: - Addison-Wesley, 1994. - 417 p. ISBN: 0201633612.
2. Mark Grand. Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Volume 1, 2nd Edition. - Wiley Publishing, 2002. - 480 p. ISBN: 0471258393
3. Design Patterns. URL: https://en.wikipedia.org/wiki/Design_Patterns
4. Design Pattern – Overview. URL: https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm
5. Behavioral pattern - Wikipedia. URL: https://en.wikipedia.org/wiki/Behavioral_pattern#:~:text=In%20software%20engineering%2C%20behavioral%20design,flexibility%20in%20carrying%20out%20communication
6. Behavioral Design Patterns. URL: https://refactoring.guru/design-patterns/behavioral-patterns
7. Behavioral patterns. URL: https://sourcemaking.com/design_patterns/behavioral_patterns
8. Behavioral Design Patterns. URL: https://www.javatpoint.com/behavioral-design-patterns
9. Behavioral Design Patterns. URL: https://programmingline.com/software-design-patterns/behavioral-design-patterns
10. Behavioral Patterns. URL: https://howtodoinjava.com/design-patterns/behavioral/

ЛАБОРАТОРНА РОБОТА № 7. ШАБЛОНИ ПОВЕДІНКИ. ШАБЛОНИ MEMENTO, STATE, COMMAND, INTERPRETER

Тема

Шаблони поведінки. Шаблони Memento, State, Command, Interpreter.

Мета

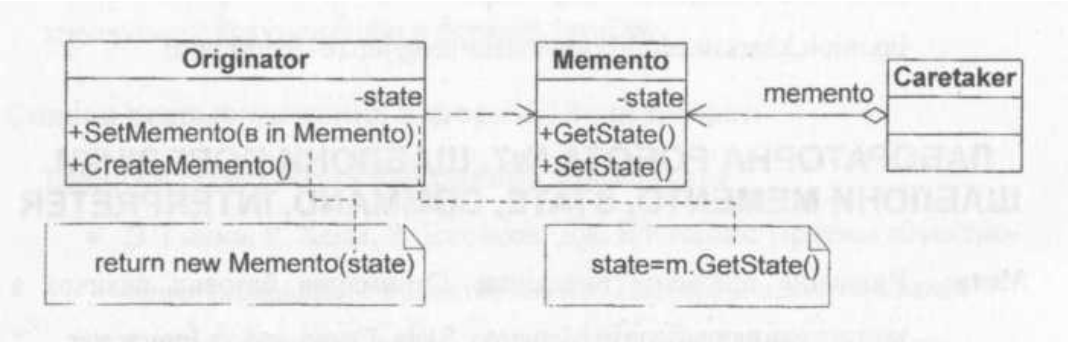
Вивчення шаблонів поведінки. Отримання базових навичок з застосування шаблонів Memento, State, Command, Interpreter.

Короткі теоретичні відомості

Шаблон Memento

Проблема. Необхідно зафіксувати стан об'єкту для реалізації, наприклад, механізму відкату.

Рішення. Зафіксувати і винести (не порушуючи інкапсуляції) за межі об'єкта його внутрішній стан так, щоб згодом можна було його відновити в об'єкті. "Memento" зберігає внутрішній стан об'єкта "Originator" і забороняє доступ до себе всім іншим об'єктам окрім "Originator", який має доступ до всіх даних для відновлення в колишньому стані. "Caretaker" може лише передавати "Memento" іншим об'єктам. "Originator" створює "Memento", що містить знімок поточного внутрішнього стану і використовує "Memento" для відновлення внутрішнього стану. "Caretaker" відповідає за збереження "Memento", при цьому не робить ніяких операцій над "Memento" і не досліджує його внутрішній вміст. "Caretaker" запитує "Memento" у "Originator", деякий час тримає його у себе, а потім повертає "Originator".Структура шаблону наведена на мал. 15.



Мал. 15. Структура шаблону Memento

Шаблон Memento має такий склад:

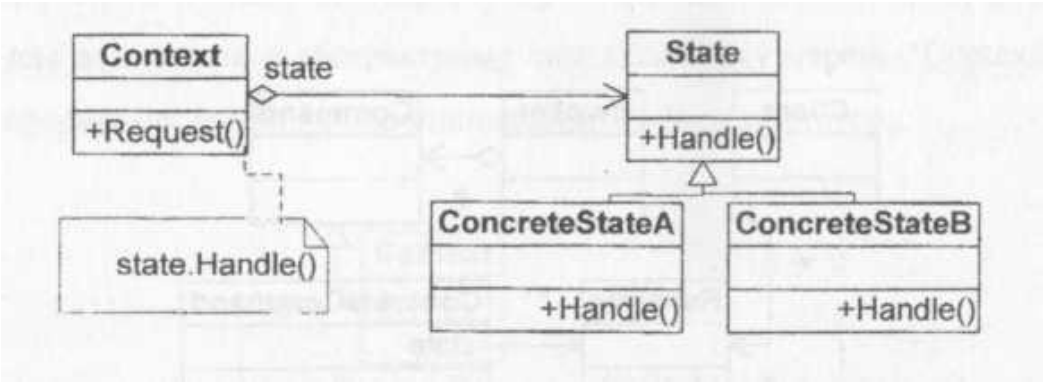
1. Memento – пам’ятка, цей компонент призначений для:
   - зберігання внутрішнього стану об’єкта Originator (творець). Обсяг збереженої інформації може бути різним і визначається потребами його творця.
   - захисту від доступу всіх інших об'єктів, окрім творця. Memento мають фактично два інтерфейси: вузький та широкий. Доглядач (caretaker) має вузький інтерфейс до Memento, для надання можливості передачі пам’ятки іншим об’єктам. В той же час Originator (творець) має широкий інтерфейс, для надання можливості доступу до всіх даних, необхідних для відновлення свого попереднього стану. Лише творцю, який виготовив пам’ятку, дозволяється доступ до внутрішнього стану пам’ятки.
2. Originator – автор, творець, хазяїн, цей компонент призначений для:
   - створення пам’ятки зі знімком поточного внутрішнього стану;
   - використання пам’ятки при відновленні внутрішнього стану.
3. Caretaker – доглядач, посильний, цей компонент призначений для реалізації механізму скасування та збереження пам’ятки; ніколи не оперує з вмістом пам’ятки і не вивчає її вміст.

При використанні шаблону не розкривається інформація, яка доступна тільки "Originator", спрощується структура "Originator". Але з використанням "Memento" можуть бути пов'язані значні витрати, якщо "Originator" повинен копіювати великий обсяг інформації, або якщо копіювання повинно проводитися часто.

Шаблон State

Проблема. Варіювати поведінку об'єкта в залежності від його внутрішнього стану.

Рішення. Клас "Context" делегує запити, які залежать від стану, поточному об'єкту "ConcreteState" (зберігає екземпляр підкласу "ConcreteState", яким визначається поточний стан), і визначає інтерфейс, що представляє інтерес для клієнтів. "ConcreteState" реалізує поведінку, асоційовану з якимось станом об'єкта "Context". "State" визначає інтерфейс для інкапсуляції поведінки, асоційованої з конкретним екземпляром "Context". Шаблон локалізує залежну від стану поведінку і ділить її на частини, що відповідають станам, переходи між станами стають явними. Структура шаблону наведена на мал. 16.



Мал. 16. Структура шаблону State

Шаблон State має такий склад:

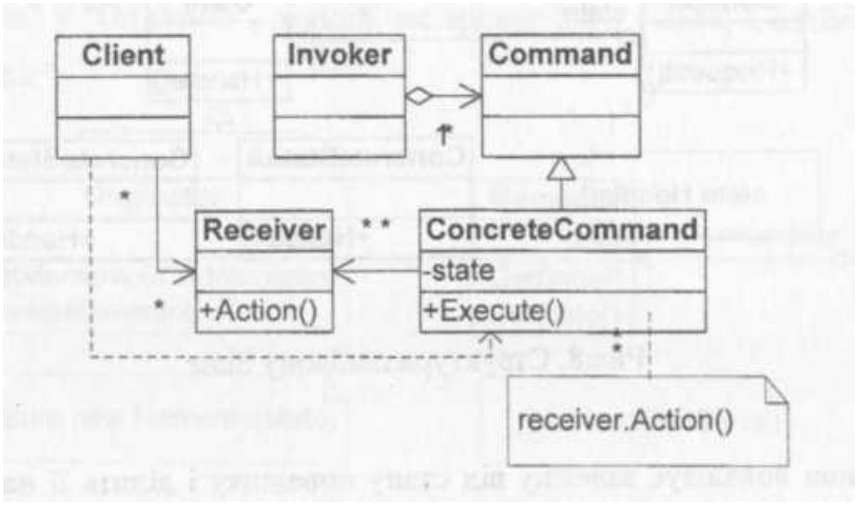
1. Context - контекст, цей компонент призначений для визначення інтерфейсу, який цікавить клієнтів, та зберігання екземпляра підкласу ConcreteState, який визначає поточний стан.
2. State - стан, цей компонент призначений для визначення інтерфейсу інкапсуляції поведінки, пов'язаної з певним станом контексту.
3. Підкласи ConcreteState - конкретний стан, цей компонент призначений для реалізації кожним підкласом поведінки, яка відповідає певному стану Контексту.

Шаблон Command

Проблема. Необхідно надіслати об'єкту запит, не знаючи про те, виконання якої операції запитане і хто буде одержувачем.

Рішення. Інкапсулювати запит як об'єкт. "Client" створює об'єкт "ConcreteCommand", який викликає операції одержувача для виконання запиту, "Invoker" відправляє запит, виконуючи операцію "Command" Execute(). "Command" оголошує інтерфейс для виконання операції, "ConcreteCommand" визначає зв'язок між об'єктом "Receiver" і операцією Action(), і, крім того, реалізує операцію Execute() шляхом виклику відповідних операцій об'єкта "Receiver". "Client" створює екземпляр класу "ConcreteCommand" і встановлює його одержувача, "Invoker" звертається до команди для виконання запиту, "Receiver" (будь-який клас) має інформацію про способи виконання операцій, необхідних для виконання запиту. Структура шаблону наведена на мал. 17.

Шаблон Command розриває зв'язок між об'єктом, який ініціює операції, і об'єктом, що має інформацію про те, як її виконати, крім того створюється об'єкт "Command", який можна розширювати і маніпулювати ним як об'єктом.



Мал. 17. Структура шаблону Command

Склад шаблону Command такий:

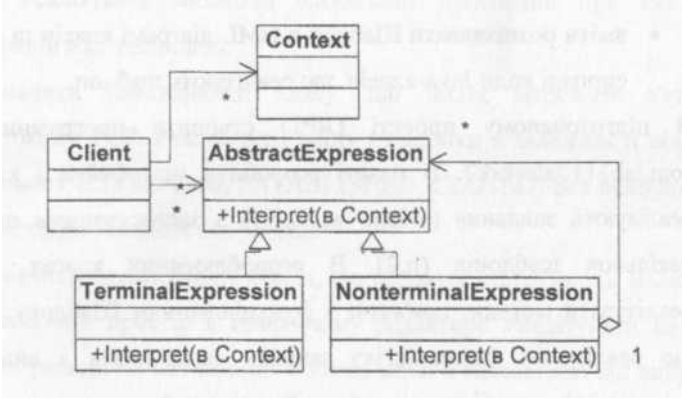
1. Command - команда, цей компонент призначений для оголошення інтерфейсу виконання операції.
2. ConcreteCommand – конкретна команда, цей компонент призначений для визначення зв'язку між об'єктом Receiver (одержувач) та дією, а ще реалізації виконання, викликаючи відповідні операції у одержувача.
3. Client - клієнт, цей компонент призначений для створення об'єкта ConcreteCommand та встановлення його одержувача.
4. Invoker - виклик (ініціатор) , цей компонент призначений для звертання до команди з метою виконання запиту.
5. Receiver – одержувач, цей компонент призначений для виконання операції, пов'язаної з виконанням запиту. Одержувачем може служити будь-який клас.

Шаблон Interpreter

Проблема. Існує підвладна змінам задача, що зустрічається часто.

Рішення. Створити інтерпретатор, який вирішує цю задачу.

Завдання пошуку рядків за зразком може бути вирішено за допомогою створення інтерпретатора, що визначає граматику мови. "Client" будує речення у вигляді абстрактного синтаксичного дерева, у вузлах якої знаходяться об'єкти класів "TerminalExpression" і "NonterminalExpression" (рекурсивне), потім "Client" ініціалізує контекст і викликає операцію Interpret(Context). На кожному вузлі типу "TerminalExpression" реалізується операція Interpret(Context) для кожного підвиразу. Для класу "NonterminalExpression" операція Interpret(Context) визначає базу рекурсії. "AbstractExpression" визначає абстрактну операцію Interpret(Context), загальну для всіх вузлів у абстрактному синтаксичному дереві. "Context" містить інформацію, глобальну по відношенню до інтерпретатору. Структура шаблону наведена на мал. 18.



Мал. 18. Структура шаблону Interpreter

Склад шаблону Interpreter такий:

1. AbstractExpression - абстрактний вираз, цей компонент призначений для оголошення абстрактної операції інтерпретації, загальної для всіх вузлів у дереві абстрактного синтаксису.
2. TerminalExpression - кінцевий вираз, цей компонент призначений для реалізації операції Interpret, пов'язаної з символами кінцевого виразу в граматиці, та для створення екземпляра, необхідного кожному символу кінцевого виразу в реченні.
3. NonterminalExpression - некінцевий вираз, цей компонент призначений для:
   - відображення у вигляді відповідного класу кожного правила R :: = R1 R2 ... Rn в граматиці;
   - зберігання змінних екземплярів типу AbstractExpression для кожного із виразів R1, … , Rn;
   - реалізації операції Interpret для некінцевих виразів у граматиці; інтерпретація може бути рекурсивною для змінних, які представляють вирази R1, … , Rn.
4. Context – контекст, цей компонент призначений для збереження інформацію, яка є загальною для Interpreter.
5. Client – клієнт цей компонент призначений для побудови (або отримання) абстрактного дерева синтаксису, що представляє певне речення в мові, яку визначає граматика. Дерево абстрактного синтаксису збирається з примірників класів NonterminalExpression та TerminalExpression; також цей компонент призначений для виклику операцію Interpret.

Завдання

1. Повторити шаблони поведінки для проектування ПЗ. Знати загальну характеристику шаблонів поведінки та призначення кожного з них.
2. Детально вивчити шаблони поведінки для проектування ПЗ - Memento, State, Command та Interpreter. Для кожного з них:
   - вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки коли його застосування є доцільним та результати такого застосування;
   - знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
   - вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;
   - вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, що реалізують шаблон.
3. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork7. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.2). В розроблюваних класах повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи.
4. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

Варіанти завдання

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 8.

0. Визначити специфікації класів, які подають у векторному редакторі графічні елементи (коло, трикутник тощо) з різними атрибутами (колір, позиція, розмір тощо). Реалізувати механізм збереження/встановлення стану елемента.
1. Визначити специфікації класу, що подає персонажа в ігровому просторі з необхідними атрибутами (позиція персонажу, склад артефактів, рівень “здоров'я” тощо). Реалізувати механізм збереження/встановлення стану персонажа.
2. Визначити специфікації класів, які подають операції над таблицю в БД. Реалізувати механізм організації транзакцій при виконанні операцій над таблицею.

3. Визначити специфікації класу, що подає мережеве з'єднання протоколу TCP. Реалізувати зміну поведінки в залежності від стану з'єднання (LISTENING, ESTABLISHED, CLOSED) без використання громіздких умовних операторів.
4. Визначити специфікації класів, що подають інструменти малювання та робочий простір в графічному редакторі. Реалізувати механізм зміни реакції на натискання кнопки миші в залежності від вибраного інструменту. Уникати використання громіздких умовних конструкцій.
5. Визначити специфікації класів, що подають чергу HTTP-запитів на обробку. Реалізувати можливість виключення запитів з черги без обробки, та зміни позиції запиту через зміну значення пріоритету.
6. Визначити специфікації класів, що подають реакції на натискання пунктів меню та кнопок інструментальної панелі. Забезпечити можливість динамічної зміни реакції, а також формування макро-реакцій (послідовність з наперед заданих реакцій).
7. Визначити специфікації класів для розбору алгебраїчних виразів з операціями +, -, *, /.

Питання для самостійної перевірки

1. Класифікація шаблонів проектування ПЗ.
2. Призначення шаблонів поведінки для проектування ПЗ.
3. Коротка характеристика кожного шаблону поведінки.
4. Назви, призначення та мотивація шаблону Memento.
5. Структура шаблону Memento та його учасники.
6. Особливості реалізації шаблону Memento. Результат використання шаблону.
7. Назви, призначення та мотивація шаблону State.
8. Структура шаблону State та його учасники.
9. Особливості реалізації шаблону State. Результат використання шаблону.
10. Назви, призначення та мотивація шаблону Command.
11. Структура шаблону Command та його учасники.
12. Особливості реалізації шаблону Command. Результат використання шаблону.
13. Назви, призначення та мотивація шаблону Interpreter.
14. Структура шаблону Interpreter та його учасники.
15. Особливості реалізації шаблону Interpreter. Результат використання шаблону.
16. Шаблони, які використовуються сумісно з Memento, State, Command та Interpreter.

Протокол

Протокол має містити титульну сторінку (з номером залікової книжки), завдання, роздруківку діаграми класів, розроблений програмний код та згенеровану документацію в форматі JavaDoc. Файли вихідного коду (*.java) повинні бути додані до протоколу.

Список рекомендованих інформаційних джерел

1. Design Patterns: elements of reusable object-oriented software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Indianapolis: - Addison-Wesley, 1994. - 417 p. ISBN: 0201633612.
2. Mark Grand. Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Volume 1, 2nd Edition. - Wiley Publishing, 2002. - 480 p. ISBN: 0471258393
3. Design Patterns. URL: https://en.wikipedia.org/wiki/Design_Patterns
4. Design Pattern – Overview. URL: https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm
5. Behavioral pattern - Wikipedia. URL: https://en.wikipedia.org/wiki/Behavioral_pattern#:~:text=In%20software%20engineering%2C%20behavioral%20design,flexibility%20in%20carrying%20out%20communication
6. Behavioral Design Patterns. URL: https://refactoring.guru/design-patterns/behavioral-patterns
7. Behavioral patterns. URL: https://sourcemaking.com/design_patterns/behavioral_patterns
8. Behavioral Design Patterns. URL: https://www.javatpoint.com/behavioral-design-patterns
9. Behavioral Design Patterns. URL: https://programmingline.com/software-design-patterns/behavioral-design-patterns
10. Behavioral Patterns. URL: https://howtodoinjava.com/design-patterns/behavioral/

ЛАБОРАТОРНА РОБОТА № 8. ПОРОДЖУВАЛЬНІ ШАБЛОНИ. ШАБЛОНИ PROTOTYPE, SINGLETON, FACTORY METHOD

Тема

Породжувальні шаблони. Шаблони Prototype, Singleton, Factory Method.

Мета

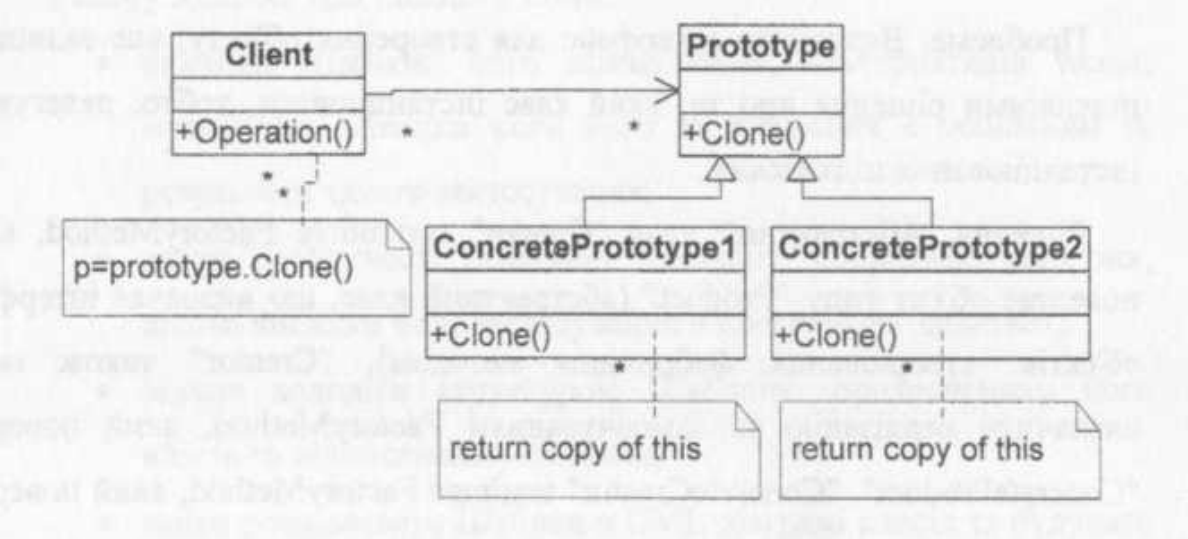
Вивчення породжувальних шаблонів. Отримання базових навичок з застосування шаблонів Prototype, Singleton та Factory Method.

Короткі теоретичні відомості

Шаблон Prototype (прототип, шаблон, який породжує об'єкти)

Проблема. Система не повинна залежати від того, як в ній створюються, компонуються і представляються об'єкти.

Рішення. Створювати нові об'єкти за допомогою клонування. "Prototype" оголошує інтерфейс для клонування самого себе. "Client" створює новий об'єкт, звертаючись до "Prototype" з запитом клонувати "Prototype". Структура шаблону наведена на мал. 19.



Мал.19. Структура шаблону Prototype

Шаблон Prototype має такий склад:

1. Prototype – прототип, цей компонент призначений для оголошення інтерфейсу для клонування.
2. ConcretePrototype - конкретний прототип, цей компонент призначений для реалізації операції з клонування себе самого.
3. Client – клієнт, цей компонент призначений для створення нового об'єкту, шляхом звернення до прототипу з запитом клонувати себе.

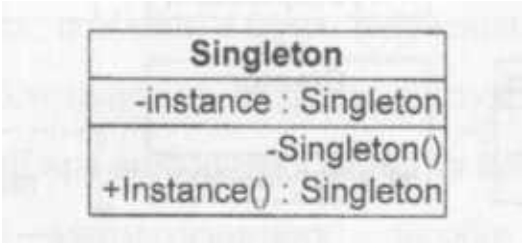
Шаблон Singleton

Проблема. Необхідний лише один примірник спеціального класу, різні об'єкти повинні звертатися до цього примірника через єдину точку доступу.

Рішення. Створити клас і визначити статичний метод класу, який повертає цей єдиний об'єкт.

Раціональніше створювати саме статичний примірник спеціального класу, а не оголосити необхідні методи статичними, оскільки при використанні методів примірника можна застосувати механізм наслідування й створювати підкласи. Статичні методи в мовах програмування не поліморфні

і не допускають перекриття в похідних класах. Рішення на основі створення екземпляра є більш гнучким, оскільки згодом може знадобитися вже не єдиний екземпляр об'єкта, а декілька. Раціональніше створювати саме статичний примірник спеціального класу, а не оголосити необхідні методи статичними, оскільки при використанні методів примірника можна застосувати механізм наслідування й створювати підкласи. Статичні методи в мовах програмування не поліморфні і не допускають перекриття в похідних класах. Рішення на основі створення екземпляра є більш гнучким, оскільки згодом може знадобитися вже не єдиний екземпляр об'єкта, а декілька. Структура шаблону наведена на мал. 20.



Мал. 20. Структура шаблону Singleton

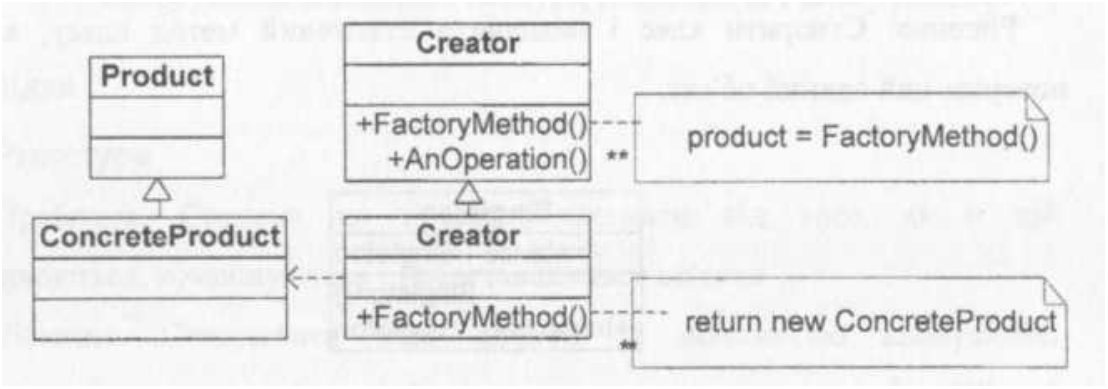
Шаблон Singleton має такий склад:

1. Singleton – одинак, цей компонент призначений для визначення операції Instance, яка дозволяє клієнтам отримати доступ до єдиного унікального екземпляра (Instance - це є операція класу), та підтримки відповідальності за створення власного унікального екземпляра.

Шаблон Factory Method

Проблема. Визначити інтерфейс для створення об'єкту, але залишити підкласам рішення про те, який клас інстанціювати, тобто, делегувати інстанціювання підкласам.

Рішення. Абстрактний клас "Creator" оголошує FactoryMethod, який повертає об'єкт типу "Product" (абстрактний клас, що визначає інтерфейс об'єктів, створюваних фабричним методом). "Creator" також може визначити реалізацію за замовчуванням FactoryMethod, який повертає "ConcreteProduct". "ConcreteCreator" заміщає FactoryMethod, який повертає об'єкт "ConcreteProduct". "Creator" "покладається" на свої підкласи в реалізації FactoryMethod, що повертає об'єкт "ConcreteProduct". Шаблон позбавляє проектувальника від необхідності вбудовувати в код залежні від програми класи. При цьому виникає додатковий рівень підкласів. Структура шаблону наведена на мал. 21.



Мал. 21. Структура шаблону Factory Method

Шаблон Factory Method має такий склад:

1. Product – продукт, цей компонент призначений для визначення інтерфейсу об'єктів, які створюються фабричним методом.
2. ConcreteProduct - конкретний продукт, цей компонент призначений для реалізації інтерфейсу Product.

3. Creator – творець, цей компонент призначений для:
   - оголошення фабричного методу, який повертає об'єкт типу Product; при цьому Творцю може бути надана можливість визначення реалізації за замовчуванням для фабричного методу, який повертає об’єкт ConcreteProduct;
   - виклику фабричного методу для створення об’єкта Product.
4. ConcreteCreator - конкретний творець, цей компонент призначений для заміщення фабричного методу, який повертає екземпляр ConcreteProduct.

Завдання

1. Вивчити породжувальні шаблони. Знати загальну характеристику породжувальних шаблонів та призначення кожного з них.
2. Детально вивчити породжувальні шаблони - Prototype, Singleton та Factory Method. Для кожного з них:
   - вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки коли його застосування є доцільним та результати такого застосування;
   - знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
   - вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;
   - вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, які реалізують шаблон.
3. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork8. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.2). В розроблюваних класах повністю реалізувати методи, пов'язані з функціонуванням Шаблону. Методи, що реалізують бізнес-логіку закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи.
4. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

Варіанти завдання

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 11.

0. Визначити специфікації класів для подання композитної структури ігрового простору. Реалізувати глибоке та поверхневе клонування такої структури.
1. Визначити специфікації класів та реалізацію методів для механізму клонування графічних елементів у редакторі векторної графіки. Забезпечити можливість як глибокого так і поверхневого клонування.
2. Визначити специфікації класів, які подають графічні елементи (примітиви та їх композиції) у редакторі векторної графіки. Реалізувати механізм клонування елементів з параметром глибини.
3. Визначити специфікації класів для подання файлової системи у вигляді дерева об'єктів (файл - листовий об'єкт, каталог - вузловий). Реалізувати механізм клонування таких об'єктів з параметром глибини.
4. Визначити специфікації класів для подання розпорядника гри, який складається з ігрового простору та списку ігрових фішок. Забезпечити можливість створення тільки одного примірника розпорядника.
5. Визначити специфікації класів для подання реляційної таблиці, схеми бази даних та валідатора запитів до таблиці. Забезпечити можливість створення тільки одного примірника схеми бази даних.
6. Визначити специфікації класів для подання композитної структури алгебраїчного виразу, карти змінних та обчислювача виразу. Забезпечити існування тільки одної карти змінних.
7. Визначити специфікації класів, які інкапсулюють лінійний список об’єктів та ітератор послідовного обходу у прямому та зворотньому напрямках для цієї структури.
8. Визначити специфікації класів, які інкапсулюють лінійний список цілих чисел та ітератор послідовного обходу у прямому напрямку в упорядкованій структурі.

9. Визначити специфікації класів для реалізації агрегату та його ітератору, який реалізує можливість зміни алгоритму пошуку наступного елементу під час виконання програми.
10. Визначити специфікації класів для реалізації композиту та його ітераторів — для обходу структури методами пошуку в глибину (DFS) та ширину (BFS).

Питання для самостійної перевірки

1. Класифікація шаблонів проектування ПЗ.
2. Призначення породжувальних шаблонів.
3. Коротка характеристика кожного породжувального шаблону.
4. Назви, призначення та мотивація шаблону Prototype.
5. Структура шаблону Prototype та його учасники.
6. Особливості реалізації шаблону Prototype. Результат використання шаблону.
7. Назви, призначення та мотивація шаблону Singleton.
8. Структура шаблону Singleton та його учасники.
9. Особливості реалізації шаблону Singleton. Результат використання шаблону.
10. Назви, призначення та мотивація шаблону Factory Method.
11. Структура шаблону Factory Method та його учасники.
12. Особливості реалізації шаблону Factory Method. Результат використання шаблону.
13. Шаблони, які використовуються сумісно з Prototype, Singleton та Factory Method.
14. Глибоке та поверхневе клонування.

Протокол

Протокол має містити титульну сторінку (з номером залікової книжки), завдання, роздруківку діаграми класів, розроблений програмний код та згенеровану документацію в форматі JavaDoc. Файли вихідного коду (*.java) повинні бути додані до протоколу.

Список рекомендованих інформаційних джерел

1. Design Patterns: elements of reusable object-oriented software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Indianapolis: - Addison-Wesley, 1994. - 417 p. ISBN: 0201633612.
2. Mark Grand. Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Volume 1, 2nd Edition. - Wiley Publishing, 2002. - 480 p. ISBN: 0471258393
3. Design Patterns. URL: https://en.wikipedia.org/wiki/Design_Patterns
4. Design Pattern – Overview. URL: https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm
5. Creational pattern - Wikipedia. URL: https://en.wikipedia.org/wiki/Creational_pattern
6. Creational Design Pattern. URL: https://social.technet.microsoft.com/wiki/contents/articles/13211.creational-design-pattern.aspx
7. Creational patterns. URL: https://sourcemaking.com/design_patterns/creational_patterns
8. Introduction to Creational Design Patterns. URL: https://www.baeldung.com/creational-design-patterns
9. Creational Design Patterns. URL: https://www.gofpatterns.com/creational/index.php
10. Creational Patterns. URL: https://howtodoinjava.com/design-patterns/creational/
/

ЛАБОРАТОРНА РОБОТА № 9.
ПОРОДЖУВАЛЬНІ ШАБЛОНИ.
ШАБЛОНИ ABSTRACT FACTORY, BUILDER

Тема

Породжувальні шаблони. Шаблони Abstract Factory, Builder.

Мета

Вивчення породжувальних шаблонів. Отримання базових навичок з застосування шаблонів Abstract Factory та Builder.

Короткі теоретичні відомості

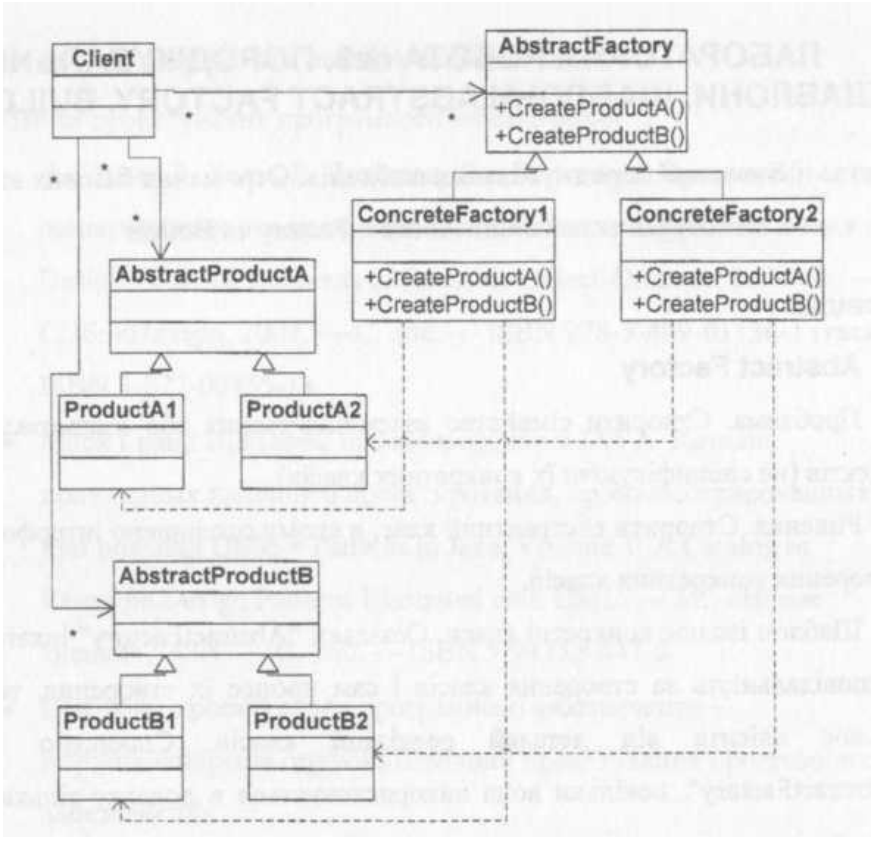
Шаблон Abstract Factory

Проблема. Створити сімейство взаємопов'язаних або взаємозалежних об'єктів (не специфікуючи їх конкретних класів).

Рішення. Створити абстрактний клас, в якому оголошено інтерфейс для створення конкретних класів.

Шаблон ізолює конкретні класи. Оскільки "AbstractFactory" інкапсулює відповідальність за створення класів і сам процес їх створення, то вона ізолює клієнта від деталей реалізації класів. Спрощено заміну' "AbstractFactory", оскільки вона використовується в додатку тільки один раз при інстанціюванні.

Слід зазначити, що інтерфейс "AbstractFactory" фіксує набір об'єктів, які можна створити. Це в певній мірі ускладнює розширення "AbstractFactory" для виготовлення нових об'єктів. Структура шаблону наведена на мал. 22.



Мал. 22. Структура шаблону Abstract Factory

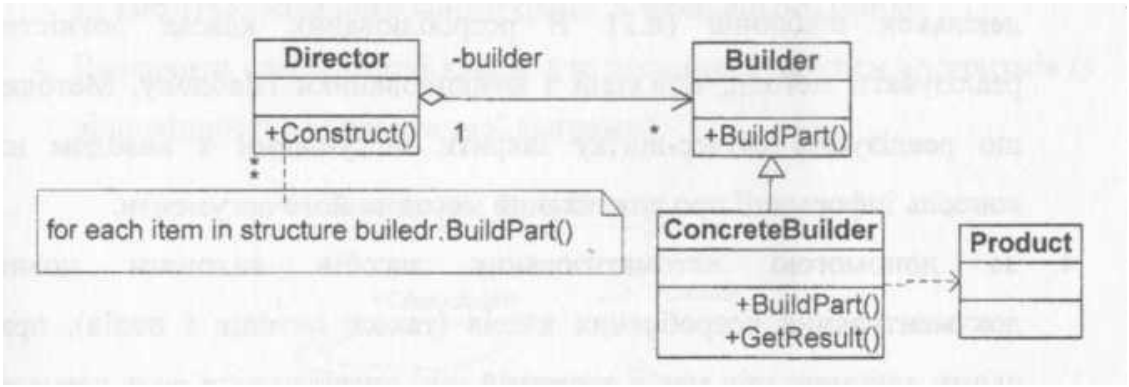
Шаблон Abstract Factory має такий склад:

1. AbstractFactory - абстрактна фабрика, цей компонент призначений для оголошення інтерфейсу для операцій, які створюють абстрактні об'єкти-продукти.
2. ConcreteFactory - конкретна фабрика, цей компонент призначений для реалізації операції по створенню конкретних об'єктів-продуктів.
3. AbstractProduct - абстрактний продукт, цей компонент призначений для оголошення інтерфейсу для типу об'єкта-продукта.
4. ConcreteProduct - конкретний продукт, цей компонент призначений для визначення об'єкта-продукта, який повинен бути створений відповідною конкретною фабрикою; також призначений для реалізації інтерфейсу AbstractProduct.
5. Client - клієнт, цей компонент призначений для використання лише інтерфейсів, які були оголошені класами AbstractFactory і AbstractProduct.

Шаблон Builder

Проблема. Відокремити конструювання складного об'єкту від його подання, так щоб в результаті одного і того ж конструювання могли виходити різні подання. Алгоритм створення складного об'єкта не повинен залежати від того, з яких частин складається об'єкт і як вони стикуються між собою.Рішення. "Client" створює об'єкт - розпорядник "Director" і конфігурує його об'єктом - "Builder". "Director" повідомляє "Builder" про те, що потрібно побудувати чергову частину "Product"."Builder" обробляє запити "Director" і додає нові частини до "Product", потім "Client" забирає "Product" у "Builder".

Об'єкт "Builder" надає об'єкту "Director" абстрактний інтерфейс для конструювання "Product", за яким може приховати подання та внутрішню структуру продукту, та, крім того, процес складання "Product". Для зміни внутрішнього представлення "Product" досить визначити новий вид "Builder". Даний шаблон ізолює код, що реалізує створення об'єкта та його подання. Структура шаблону наведена на мал. 23.



Мал.23. Структура шаблону Builder

Шаблон Builder має такий склад:

1. Builder - будівельник, цей компонент призначений для визначення абстрактного інтерфейсу для створення частин об'єкта Product.
2. ConcreteBuilder - конкретний будівельник, цей компонент призначений для:
   - конструювання та збирання до купи частин продукту, шляхом реалізації інтерфейсу Builder.
   - визначення та відстеження подання, яке він створює.
   - надання інтерфейсу для отримання доступу до продукту.
3. Director - директор, цей компонент призначений для побудови об'єкта за допомогою інтерфейсу Builder.
4. Product - продукт, цей компонент призначений для:
   - представлення складного об'єкта, який будується; при цьому ConcreteBuilder створює внутрішнє подання продукту та визначає процес його збирання.
   - підключення класів, які визначають складові частини, включаючи інтерфейси для складання частин у кінцевий результат.

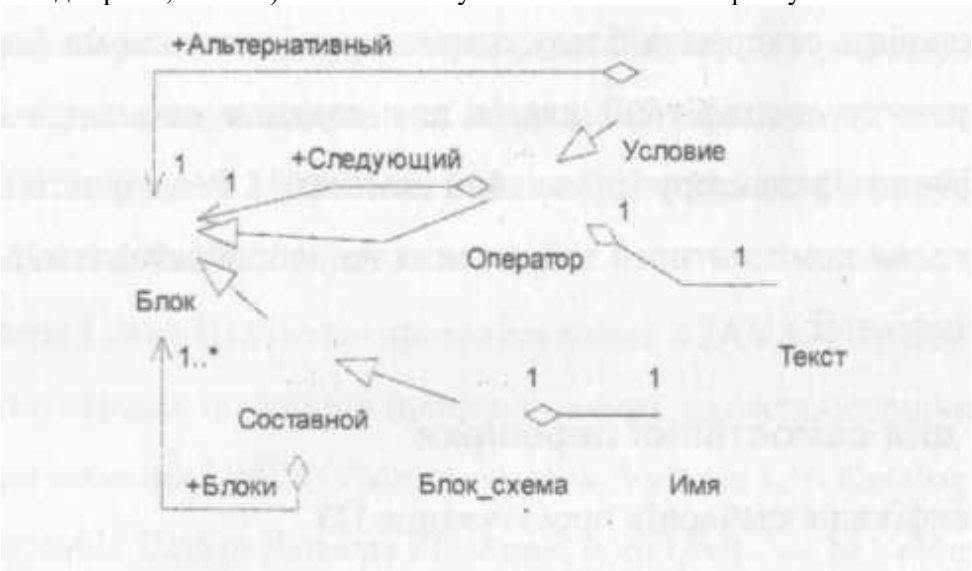
Завдання

1. Повторити породжувальні шаблони. Знати загальну характеристику породжувальних шаблонів та призначення кожного з них.
2. Детально вивчити породжувальні шаблони - Abstract Factory та Builder. Для кожного з них:
   - вивчити Шаблон, його призначення, альтернативні назви, мотивацію, випадки коли його застосування є доцільним та результати такого застосування;
   - знати особливості реалізації Шаблону, споріднені шаблони, відомі випадки його застосування в програмних додатках;
   - вільно володіти структурою Шаблону, призначенням його класів та відносинами між ними;
   - вміти розпізнавати Шаблон в UML діаграмі класів та будувати сирцеві коди Java-класів, що реалізують шаблон.
3. В підготованому проекті (ЛР1) створити програмний пакет com.lab111.labwork9. В пакеті розробити інтерфейси і класи, що реалізують завдання (згідно варіанту) з застосуванням одного чи декількох шаблонів (п.2). В розроблюваних класах повністю реалізувати методи, пов'язані з функціюванням Шаблону. Методи, що реалізують бізнес-логіку закрити заглушками з виводом на консоль інформації про викликаний метод та його аргументи.
4. За допомогою автоматизованих засобів виконати повне документування розроблених класів (також методів і полів), при цьому документація має в достатній мірі висвітлювати роль певного класу в загальній структурі Шаблону та особливості конкретної реалізації.

Варіанти завдання

Номер варіанту завдання обчислюється як залишок від ділення номеру залікової книжки на 11.

0. Визначити специфікації класів для подання сімейства віджетів графічного інтерфейсу користувача з реалізацією на різних API (WinAPI, GTK). Забезпечити можливість прозорого для клієнта розширення реалізацією для інших API (Qt, OSX).
1. Визначити специфікації класів для подання сімейства інструментів роботи з об'єктними даними через різні API (DB, File). Забезпечити можливість прозорого для клієнта розширення реалізацією для інших API (WebService).
2. Визначити специфікації класів для подання сімейства інструментів універсального інтерактивного середовища розробки (Validator, Compiler, Debugger) з їх реалізацією для різних мов (Java, C++). Забезпечити можливість прозорого для клієнта розширення реалізацією для мов (ObjectPascal).
3. Визначити специфікації класів для прямокутного ігрового простору та завантажувача його конфігурації із зовнішнього файлу.
4. Визначити специфікації класів для подання блок-схем алгоритмів (у відповідності до семантичної діаграми, мал. 24) та її завантажувача із зовнішнього файлу.



Мал. 24. Семантична діаграма блок-схем алгоритмів

5. Визначити специфікації класів для будівника дерева розбору складного виразу (у відповідності до Форми Бекуса-Наура) на основі його символьного подання.

<вираз>::=<простий вираз> | <складний вираз>
<простий вираз>::=<константа> | <змінна>
<константа>::=(<число>)
<змінна>::=(<ім'я>)
<складний вираз>::=(<вираз><знак операції><вираз>)
<знак операції>::=+|-|*|/

6. Визначити специфікації класів для подання запису, реляційної таблиці та її завантажувача із зовнішнього файлу.
7. Визначити специфікації класів для подання реляційної таблиці та будівника прямого добутку таблиць.
8. Визначити специфікації класів для подання реляційної таблиці та будівника проекції таблиць.
9. Визначити специфікації класів для подання реляційної таблиці, схеми бази даних та відповідного завантажувача. Забезпечити можливість створення тільки одного примірника схеми бази даних.
10. Визначити специфікації класів для подання елементів векторного графічного редактору (примітив і композит). Реалізувати можливість побудови композитного зображення на основі завантаженого файлу- специфікації.

Питання для самостійної перевірки

1. Класифікація шаблонів проектування ПЗ.
2. Призначення породжувальних шаблонів.
3. Коротка характеристика кожного породжувального шаблону.
4. Назви, призначення та мотивація шаблону Prototype.
5. Структура шаблону Prototype та його учасники.
6. Особливості реалізації шаблону Prototype. Результат використання шаблону.
7. Назви, призначення та мотивація шаблону Singleton.
8. Структура шаблону Singleton та його учасники.
9. Особливості реалізації шаблону Singleton. Результат використання шаблону.
10. Назви, призначення та мотивація шаблону Factory Method.
11. Структура шаблону Factory Method та його учасники.
12. Особливості реалізації шаблону Factory Method. Результат використання шаблону.
13. Шаблони, які використовуються сумісно з Prototype, Singleton та Factory Method.
14. Глибоке та поверхневе клонування.

Протокол

Протокол має містити титульну сторінку (з номером залікової книжки), завдання, роздруківку діаграми класів, розроблений програмний код та згенеровану документацію в форматі JavaDoc. Файли вихідного коду (*.java) повинні бути додані до протоколу.

Список рекомендованих інформаційних джерел

1. Design Patterns: elements of reusable object-oriented software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Indianapolis: - Addison-Wesley, 1994. - 417 p. ISBN: 0201633612.
2. Mark Grand. Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Volume 1, 2nd Edition. - Wiley Publishing, 2002. - 480 p. ISBN: 0471258393
3. Design Patterns. URL: https://en.wikipedia.org/wiki/Design_Patterns
4. Design Pattern – Overview. URL: https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm
5. Creational pattern - Wikipedia. URL: https://en.wikipedia.org/wiki/Creational_pattern
6. Creational Design Pattern. URL: https://social.technet.microsoft.com/wiki/contents/articles/13211.creational-design-pattern.aspx
7. Creational patterns. URL: https://sourcemaking.com/design_patterns/creational_patterns
8. Introduction to Creational Design Patterns. URL: https://www.baeldung.com/creational-design-patterns
9. Creational Design Patterns. URL: https://www.gofpatterns.com/creational/index.php
10. Creational Patterns. URL: https://howtodoinjava.com/design-patterns/creational/