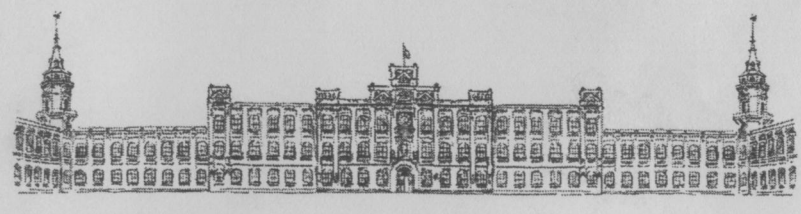
Міністерство освіти і науки, молоді та спорту України
Національний технічний університет України
«Київський політехнічний інститут»

Інженерія програмного забезпечення

Методичні вказівки
до виконання лабораторних робіт

Частина І

Структурні шаблони



Київ-2011

Міністерство освіти і науки, молоді та спорту України
Національний технічний університет України
«Київський політехнічний інститут»

Інженерія програмного забезпечення

Методичні вказівки
до виконання лабораторних робіт
для студентів напряму підготовки
6.050102 «Комп'ютерна інженерія»

Частина І

Структурні шаблони

Рекомендовано Методичною радою НТУУ «КПІ»

Київ
НТУУ «КПІ»
2011

Інженерія програмного забезпечення [Текст] : метод. вказівки до викон. лабораторн. робіт для студ. напряму підготов. 6.050102 «Комп'ютерна інженерія» / Уклад.: А. О. Болдак, О. Н. Абу Усбах. – К.: НТУУ «КПІ», 2011. – Ч. І. Структурні шаблони. – 40 с.

Гриф надано Методичною радою НТУУ «КПІ»
(Протокол № 5 від 03.02.2011 р.)

Навчальне видання

Інженерія програмного забезпечення

Методичні вказівки
до виконання лабораторних робіт
для студентів напряму підготовки
6.050102 «Комп'ютерна інженерія»

Частина І

Структурні шаблони

Укладачі: *Болдак Андрій Олександрович,* канд. техн. наук, доц.
Абу Усбах Олексій Нідалійович, канд. техн. наук, доц.

Відповідальний редактор: *О. В. Бузовський,* д-р техн. наук, проф.

Рецензент: *І. А. Дичка,* д-р техн. наук, проф.

За редакцією укладачів
Надруковано з оригінал-макета замовника

Темплан 2010 р., поз. 2-130

Підп. до друку 16.03.2011. Формат $60\times84^{1}/_{16}$. Папір офс. Гарнітура Times.
Спосіб друку – ризографія. Ум. друк. арк. 2,32. Обл.-вид. арк. 3,86. Наклад 50 пр. Зам. № 11-47.

НТУУ «КПІ» ВПІ ВПК «Політехніка»
Свідоцтво ДК № 1665 від 28.01.2004 р.
03056, Київ, вул. Політехнічна, 14, корп. 15
тел./факс (044) 406-81-78

ЗМІСТ

ВСТУП

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

Практична частина курсу складається з дев'яти лабораторних робіт і призначена для отримання практичних навичок використання шаблонів проектування при розробці програмного забезпечення. Всі лабораторні роботи виконуються в інтегрованому середовищі для розробки програм Eclipse 3.5. Роботи послідовно логічно впорядковані за складністю і охоплюють всі теми, що вивчаються в 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.

Перша частина видання містить методичні вказівки для перших чотирьох лабораторних робіт.

1. Підготовка програмного проекту.
2. Графічна нотація UML. Документування проекту.
3. Структурні шаблони проектування ПЗ. Шаблони Composite. Decorator. Proxy.
4. Структурні шаблони проектування ПЗ - 2. Шаблони Flyweight, Adapter, Bridge, Facade.

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.

Довідка

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 розмічено 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, що являє собою архів формату ZIP з додатковими метаданими в файлі маніфесту (META-INF/MANIFEST.MF). Маніфест може містити інформацію при 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 встановленої робочого середовища Java — JRE. Відмова від використання команд операційної системи і формат XML забезпечують переносимість сценаріїв.

Управління процесом збирання відбувається за допомогою XML-сценарію, який також називають Build-файлом (build.xml). Цей файл містить кореневий елемент- проект, що складається з елементів-цілей (<target>). Цілі є еквівалентом процедур в мовах програмування і містять виклики команд-завдань (task). Завдання являє собою XML-елемент, що пов'язаний з java-класом, який 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, що реалізують певні інтерфейси, та зв'язування цього класу 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 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

Список рекомендованих інформаційних джерел

XML

- XML. Материал из Википедии – свободной энциклопедии. – http://ru.wikipedia.org/wiki/XML.
- Язык XML– практическое введение. – http://www.citforum.ru/internet/xml/index.shtml.

JAR

- Trail: JAR Files. – http://khpi–iip.mipk.kharkiv.edu/library/extent/prog/jar/index.html.

ANT

- Apache Ant. –http://ru.wikipedia.org/wiki/Apache_Ant.
- Ant за 10 шагов. –http://www.opennet.ru/base/dev/ant_10.txt.html.

ЛАБОРАТОРНА РОБОТА №2. 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.

Довідка

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 (UML1.x) / Комунікації - Communication (UML2.0)
  - Огляду взаємодії - Interaction overview diagram (UML2.0)
  - Послідовності - Sequence diagram
  - Синхронізації - UML Timing Diagram (UML2.0)

Діаграма кла́сів — 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 і 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 на 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 ромба 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 (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 генералізація 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, що містить три інтерфейси If1, If2, If3 з методами meth1(), meth2(), meth3 та класи що їх реалізують Cl1, Cl2, Cl3 відповідно.
3. Згідно варіанту (нижче) реалізувати на діаграмі класів відношення генералізації та агрегації.
4. В підготованому проекті (JP1) створити програмний пакет com.lab111.labwork2. В пакеті розробити інтерфейси і класи згідно діаграмі (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 <- If3; 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. I1 <- Cl1; Cl3 <- Cl2; Cl3 <- Cl3
1. I1 <- Cl2; Cl1 <- Cl3; Cl1 <- Cl1
2. I1 <- Cl3; Cl1 <- Cl2; Cl1 <- Cl1
3. I1 <- Cl1; I2 <- Cl1; Cl3 <- Cl2
4. I3 <- Cl2; I2 <- 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-моделерів.

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 згенеровану документацію в форматі JavaDoc.

Список рекомендованих інформаційних джерел

UML

- А.В. Бабич - Введение в UML – http://www.intuit.ru/department/se/intuml/
- UML - http://ru.wikipedia.org/wiki/UML
- Диаграмма классов – http://ru.wikipedia.org/wiki/Диаграмма классов
- Унифицированный язык моделирования (UML) – http://www.interface.ru/public/990804/uml4b.htm
- Книги по UML – http://progbook.net/uml/
- Comparison of Unified Modeling Language tools – http://en.wikipedia.org/wiki/List_of_UML_tools

AgroUML

- ArgoUML Documentation Resources – http://argouml.tigris.org/documentation/

Umbrello

- Руководство Umbrello UML Modeller – http://www.nundesign.com/st/uml_doc/
- Umbrello UML Modeller - http://uml.sourceforge.net/

JavaDoc

- Теория и практика Java: Мне нужно задокументировать ЭТО? – http://www.ibm.com/developerworks/ru/library/j-jtp0821/index.html
- Javadoc - Создание Javadoc документации – http://www.cs.vsu.ru/%7Esvv/progis/Javadoc.pdf

ЛАБОРАТОРНА РОБОТА №3. 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.

Довідка

Composite

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

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

Decorator

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

Рішення. Динамічно додати об'єкту нові обов'язки не вдаючись при цьому до породження підкласів (наслідування). "Component" визначає інтерфейс для об'єктів, на які можуть бути динамічно 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".

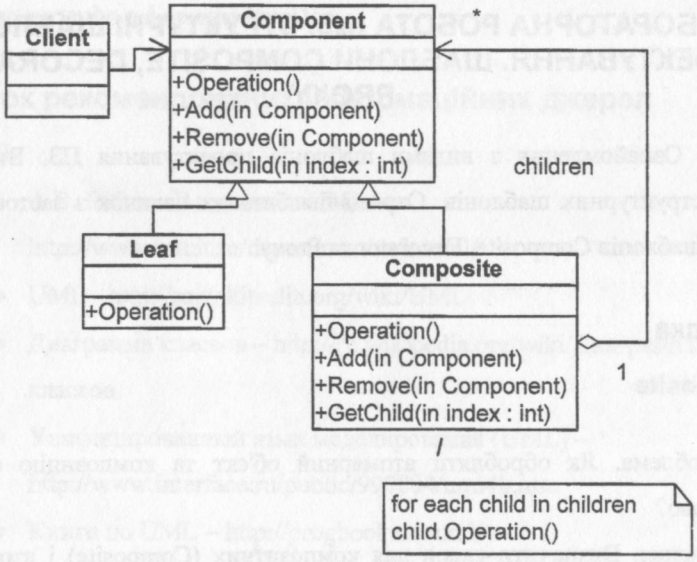


Рис.1. Структура шаблону Composite

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.

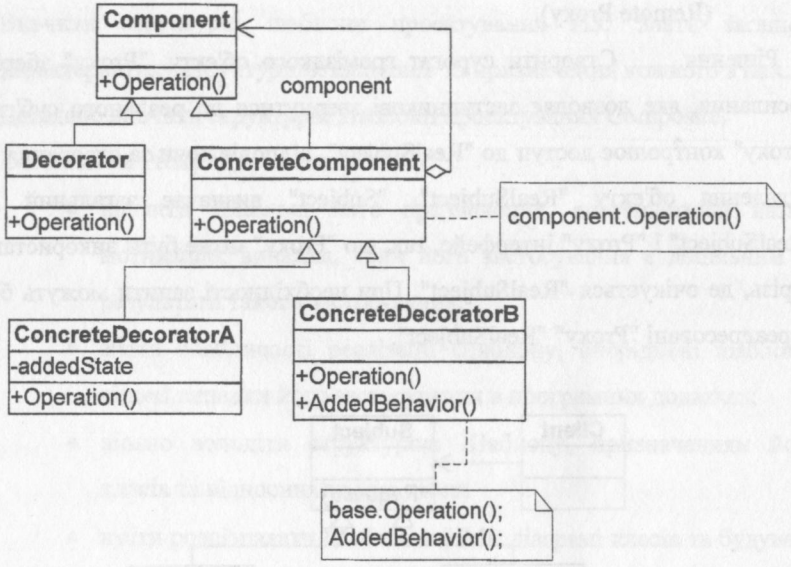


Рис.2. Структура шаблону Decorator

"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 в вивченні та налагодженні.

Proxy

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

- створювати громіздкі об'єкти "на вимогу" (Virtual Proxy);
- кешувати дані (SmartLink Proxy);
- динамічно 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".

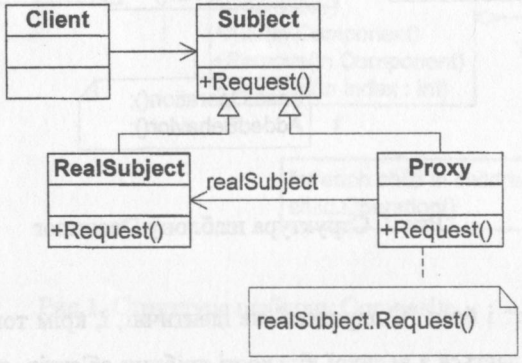


Рис.3. Структура шаблону Proxy

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 (динамічно обчислюються).
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.

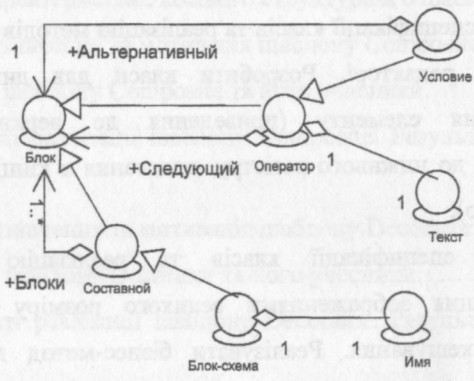


Рис.4. Семантична діаграма блок-схем алгоритмів

Реалізувати бізнес-метод ідентифікації елемента та його зв'язків з іншими елементами блок-схеми.

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 згенеровану документацію в форматі JavaDoc.

Список рекомендованих інформаційних джерел

Шаблони проектування програмного забезпечення

- Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software. — СПб: «Питер», 2007. — С. 366. — ISBN 978-5-469-01136-1 (также ISBN 5-272-00355-1)
- Марк Гранд Шаблоны проектирования в JAVA. Каталог популярных шаблонов проектирования, проиллюстрированных при помощи UML = Patterns in Java, Volume 1. A Catalog of Reusable Design Patterns Illustrated with UML. — М.: «Новое знание», 2004. — С. 560. — ISBN 5-94735-047-5
- Шаблони проектування програмного забезпечення – http://uk.wikipedia.org/wiki/Шаблони проектування програмного забезпечення
- Обзор паттернов проектирования – http://citforum.ru/SE/project/pattern/
- Объектно-ориентированное проектирование, паттерны проектирования (Шаблоны) – http://www.javenue.info/themes/ood/
- David Gallardo. Шаблоны проектирования Java - http://khpi-iip.mipk.kharkiv.edu/library/extent/prog/jdp101/index.html

Структурні шаблони

- Структурні шаблони – http://uk.wikipedia.org/wiki/Структурні шаблони
- Структурные шаблоны – http://khpi-iip.mipk.kharkiv.edu/library/extent/prog/jdp101/part5.html

- Шаблоны проектирования: структурные паттерны – http://www.pcmag.ru/solutions/detail.php?ID=34464
- Структурные шаблоны проектирования - http://piarmedia.ru/?page_id=17

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

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

Довідка

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 він 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".

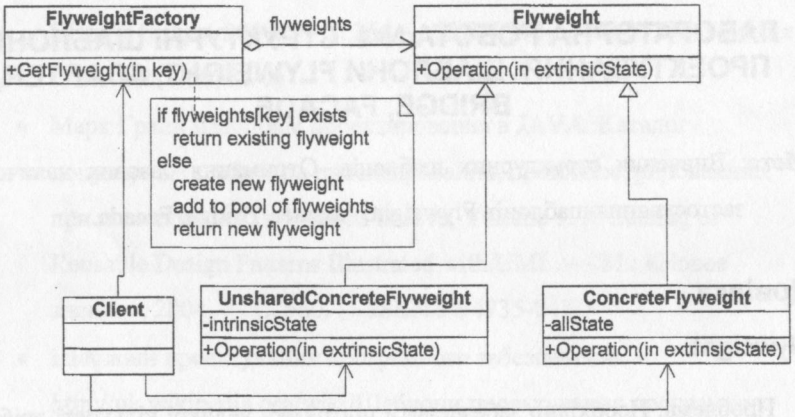


Рис.5. Структура шаблону Flyweight

Пристосуванці 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 економиться пам'ять.

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.

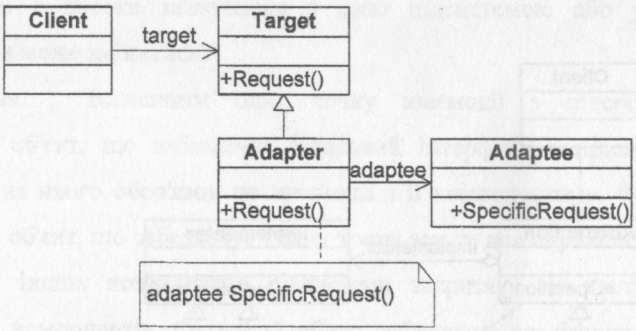


Рис.6. Структура шаблону Adapter

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" .

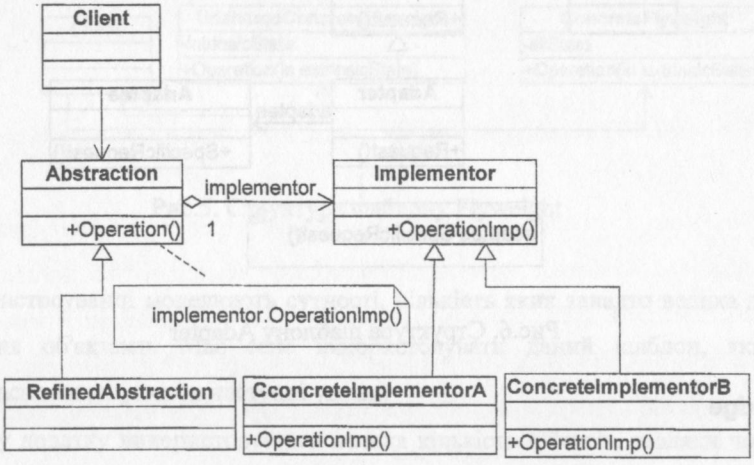


Рис.7. Структура шаблону Bridge

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" і "Implementor" усуває залежності від реалізації, що встановлюються на етапі компіляції: щоб змінити клас "Implementor" зовсім не обов'язково перекомпілювати клас "Abstraction".

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.

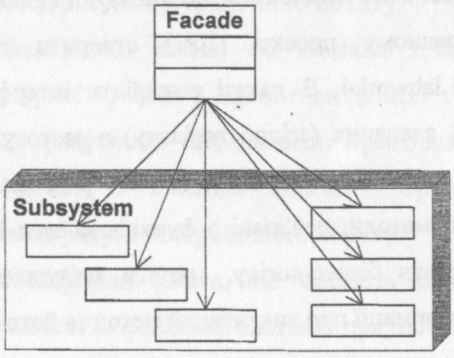


Рис.8. Структура шаблону Facade

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, що 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 вісі 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.

Список рекомендованих інформаційних джерел

Шаблони проектування програмного забезпечення

- Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software. — СПб: «Питер», 2007. — С. 366. — ISBN 978-5-469-01136-1 (также ISBN 5-272-00355-1)
- Марк Гранд Шаблоны проектирования в JAVA. Каталог популярных шаблонов проектирования, проиллюстрированных при помощи UML = Patterns in Java, Volume 1. A Catalog of Reusable Design Patterns Illustrated with UML. — М.: «Новое знание», 2004. — С. 560. — ISBN 5-94735-047-5

- Шаблони проектування програмного забезпечення – http://uk.wikipedia.org/wiki/Шаблони проектування програмного забезпечення
- Обзор паттернов проектирования – http://citforum.ru/SE/project/pattern/
- Объектно-ориентированное проектирование, паттерны проектирования (Шаблоны) – http://www.javenue.info/themes/ood/
- David Gallardo. Шаблоны проектирования Java - http://khpi-iip.mipk.kharkiv.edu/library/extent/prog/jdp101/index.html

Структурні шаблони

- Структурні шаблони – http://uk.wikipedia.org/wiki/Структурні шаблони
- Структурные шаблоны – http://khpi-iip.mipk.kharkiv.edu/library/extent/prog/jdp101/part5.html
- Шаблоны проектирования: структурные паттерны – http://www.pcmag.ru/solutions/detail.php?ID=34464
- Структурные шаблоны проектирования – http://piarmedia.ru/?page_id=17