МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра інформаційних технологій та комп’ютерної інженерії

В.І. Олевський, Б.В. Молодець

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Методичні рекомендації до виконання лабораторних робіт
для здобувачів ступеня бакалавра
спеціальності F6 Інформаційні технології

Дніпро
НТУ «ДП»
2025

Проектування інформаційних систем [Електронний ресурс]: методичні рекомендації до виконання лабораторних робіт з дисципліни «Проектування інформаційних систем» для здобувачів ступеня бакалавра спеціальності F6 Інформаційні технології / уклад.: В.І. Олевський, Б.В. Молодець ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2025. – 40 с.

Укладачі:
В.І. Олевський, д-р техн. наук, проф.;
Б.В. Молодець, PhD, доц.

Затверджено науково-методичною комісією спеціальності F6 Інформаційні технології (протокол № 6 від 19.05.2025 р.) за поданням кафедри інформаційних технологій та комп'ютерної інженерії (протокол № 15 від 19.05.2025 р.).

Методичні рекомендації містять опис методики виконання лабораторних робіт з дисципліни «Проектування інформаційних систем» студентами спеціальності F6 Інформаційні технології.

Орієнтовано на активізацію навчальної діяльності здобувачів ступеня бакалавра спеціальності F6 Інформаційні технології та закріплення практичних навичок у засвоєнні дисципліни «Проектування інформаційних систем».

Відповідальний за випуск завідувач кафедри інформаційних технологій та комп'ютерної інженерії, д-р техн. наук, проф. В.В. Гнатушенко.

ЗМІСТ

Стор.

ВСТУП

Методичні рекомендації призначені для здобувачів ступеня бакалавра спеціальності F6 Інформаційні технології, які вивчають дисципліну «Проєктування інформаційних систем».

Матеріали містять низку послідовних та взаємопов'язаних лабораторних робіт, спрямованих на формування практичних навичок з аналізу, моделювання та проєктування сучасних інформаційних систем. У процесі виконання завдань студенти ознайомляться з принципами побудови інформаційних систем, основами структурного та об'єктно-орієнтованого підходів до проєктування, методами розробки діаграм UML, побудовою моделей даних, а також основами документації технічних рішень.

Перед початком виконання кожної лабораторної роботи здобувач повинен:

– ознайомитися з відповідними методичними вказівками;

– повторити теоретичний матеріал, що стосується теми лабораторної роботи;

– підготувати відповіді на контрольні запитання, подані наприкінці кожної лабораторної роботи.

Після завершення виконання лабораторної роботи здобувач зобов'язаний:

– продемонструвати результати викладачеві (на комп'ютері або у вигляді письмового звіту);

– оформити звіт за результатами виконаної роботи;

– захистити лабораторну роботу та здати її викладачеві.

У разі своєчасного (протягом двох тижнів від проведення заняття) та коректного виконання завдання студент отримує оцінку 100. У разі прострочення здачі роботи – за кожен день затримки знімається 1 бал.

Критерії оцінювання лабораторних робіт:

Лабораторна робота			
Бали	**Критерій**		
80	Виконання практичної частини	80	Студент правильно, в повному об'ємі виконує лабораторну роботу згідно завданих методичних інструкцій.
		-5	-5 балів за кожну помилку або невиконаний пункт завдання в лабораторній роботі
20	Оформлення звіту	20	Студент правильно оформляє звіт з лабораторної роботи згідно до вимог, та надає відповіді на всі питання
		10	Студент неправильно оформляє звіт з лабораторної роботи або надає відповіді не на всі питання
		0	Студент не надав звіт до лабораторної роботи
-1	Термін виконання	-1	При несвоєчасному виконанні завдання -1 бал за кожен прострочений день
Максимальна кількість балів: 100			

1 ЛАБОРАТОРНА РОБОТА №1
ОФОРМЛЕННЯ ТЕХНІЧНОГО ЗАВДАННЯ ДЛЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1 Мета лабораторної роботи

Набути практичних навичок у формуванні та оформленні технічного завдання (ТЗ) на створення інформаційної системи. В рамках роботи студенти розробляють спрощений, але структурований варіант технічного завдання, що містить основні розділи відповідно до стандартних вимог ISO/IEC/IEEE 29148:2018.

Під час виконання завдання студенти мають:

– ознайомитися зі структурою та змістом типового технічного завдання на створення ІС;

– визначити мету створення інформаційної системи, її основні функції, задачі та очікувані результати впровадження;

– описати вимоги до програмного та технічного забезпечення, функціональні та нефункціональні вимоги;

– навчитися узагальнювати вимоги замовника та формалізувати їх у вигляді проектного документа;

– відпрацювати навички використання стандартів та шаблонів під час написання ТЗ.

У результаті виконання лабораторної роботи студент повинен вміти оформляти технічну документацію, що є першочерговим етапом у процесі розробки будь-якої інформаційної системи

1.2 Організація виконання лабораторної роботи

Для виконання лабораторної роботи необхідно вивчити, використовуючи рекомендовану літературу, конспект лекцій і методичні вказівки, такі питання:

– шо таке технічне завдання;

– які на сьогодні існують стандарти оформлення технічного завдання;

– розглянути структуру існуючих технічних завдань інформаційної системи.

Перед початком оформлення загального опису системі в технічному завданні необхідно буде вказати:

– область застосування системи;

– перелік абревіатур та скорочень, що використовуються в документі;

– перелік документів/ресурсів на які посилаєтесь в ТЗ.

Далі необхідно оформити загальний опис для програми візуалізації географічних та геометричних об'єктів SimpleGisViewer. Наприклад з якими типами файлів працює система, які саме функції роботи з географічними даними підтримує (імпорт/експорт, конвертацію у різні системи координат тощо). Додати інформацію щодо перспектив продукту (конкурентоспроможність системи або ж

взаємодія з іншими системами), характеристики користувачів (опис типових користувачів системи, їх досвід – user experience, навички), обмеження (технічні або регуляторні, що можуть вплинути на розробки інформаційної системи), фактори, що можуть вплинути на вимоги/реалізацію системи.

Після цього детально розписати розділ системних вимог. В цьому розділі необхідно написати про функціональні вимоги системи, викладені як окремі підпункти або ж використовуючи use cases / user stories. Після функціональних вимог системи необхідно вказати її нефункціональні вимоги, такі як:

– продуктивність;
– безпека;
– масштабованість;
– локалізація.

Після нефункціональних вимог додайте інтерфейсні вимоги, які включатимуть:

– користувацькі інтерфейс;
– програмні інтерфейси;
– комунікаційні інтерфейси.

Користувацькі вимоги мають містити у собі опис графічного інтерфейсу користувача (GUI) та досвіду користувача (UX).

В програмних інтерфейсах має бути чітко описано які методи та функції повинні бути доступні для взаємодії між користувачем, компонентами програми та зовнішніми системами. Програмний інтерфейс повинен дозволяти іншому програмному забезпеченню або компонентам вашої системи взаємодіяти з нею, керуючи даними, відображенням та іншими функціями. Наприклад, описати програмний інтерфейс для роботи з шарами як забрежено на таблиці 1.1:

Таблиця 1.1 – Приклад опису API метода отримання інформації по об’єкту

Назва поля	Значення
Метод	GET
Опис	Використовується для отримання інформації про географічні об'єкти, що зберігаються у проекті або на картографічному сервері.
Запит	GET /api/objects/{object_id}
Параметр запиту	object_id (string) — унікальний ідентифікатор географічного об'єкта.

Відповідь	{
	"object_id": "12345",
	"type": "line",
	"coordinates": [[50.4501, 30.5031],[50.4512, 30.5040]],
	"layer": "buildings",
	"properties": {
	"name": "Building 1",
	"color": "#FF0000"
	}
	}

Цей опис за необхідності можна доповнити описом можливих помилок програмного інтерфейсу (Наприклад, 400 Bad Request – не вірний форма подання об’єкту, 404 Not Found – не існуючий об’єкт, 500 internal server error – помилка на стороні сервера).

В комунікаційних інтерфейсах необхідно описати типи протоколів та стандарти обміну даних між користувачами та зовнішніми сервісами. Наприклад, в системі SimpleGisViewer для обміну даними через веб буде використано HTTPS протокол – отримуватиме дані з веб-сервісів через GET та POST запити, або передавати дані в хмарне сховище через REST API.

Архітектурні вимоги (обмеження) повинен містити обмеження які не є функціональними вимогами, але вони впливають на реалізації функціоналу систему та обиранні технологій. Важливо розуміти, що архітектурні вимоги на відміну віл нефункціональних вимог обмежують технічні рішення, але не описують поведінку систему. Наприклад:

– вибір технологій і мов програмування – використання Java 17 з обов`язковим використанням Spring Boot;

– операційне середовище – необхідність система бути кросплатформеною, обмеження щодо апаратного забезпечення;

– архітектурний стилі та шаблони – обов’язковість дотримання архітектурних підходів таких як SOLID або Clean Architecture;

– безпекові обмеження – вимоги до автентифікації за допомогою OAuth 2.0 чи JWT, необхідність шифрування даних в базі, захист даних в транзиті;

Після цього в додатках необхідно вказати глосарій з тлумаченням термінів використаних в документі.

За наявності вказати діаграми та моделі що описують систему такі як DFD (Data Flow Diagram), ERD (Entity Relationship Diagram), SD (Sequence Diagram), UD (Usecase Diagram) тощо.

За наявності великого обсягу користувацьких сценаріїв user stories – їх також можна розмістити в додатки.

В результаті опису вимог має бути отримана наступна структура технічного завдання:

1. Вступ
2. Загальний опис
3. Системні вимоги
   3.1Функціональні вимоги
   3.2Нефункціональні вимоги
   3.3Інтерфейсні вимоги
   3.4Архітектурні вимоги
4. Додатки
   4.1Глосарій
   4.2Діаграми та моделі
   4.3Приклади сценаріїв використання

Ці пункти мають стати частиною загального звіту по лабораторній роботі.

1.3 Зміст звіту

Підготувати звіт з виконання лабораторної роботи, який повинен включати:
– номер, тему і мету лабораторної роботи;
– короткий огляд основних понять;
– опис завдання з початковими умовами;
– опис технічного завдання згідно визначеного стандарту;
– висновки

1.4 Питання для підготовки до захисту лабораторної роботи

1. Що таке технічне завдання? Яка його роль у процесі проектування інформаційної системи?
2. Якою має бути структура технічного завдання згідно зі стандартом ISO/IEC/IEEE 29148:2018?
3. Яка різниця між функціональними та нефункціональними вимогами? Навести приклади.
4. Що таке інтерфейсні вимоги? Як вони класифікуються
5. Що таке SMART-критерії вимог та як вони застосовуються у технічному завданні?
6. Що таке сценарій використання (Use Case)? Як оформлюється Use Case в технічному завданні,
7. Яка відмінність між користувацькими та системними вимогами?

2 ЛАБОРАТОРНА РОБОТА №2
РОЗРОБКА UML-ДІАГРАМ КЛАСІВ

2.1 Мета лабораторної роботи

Ознайомитися з основами об’єктно-орієнтованого моделювання та процесом побудови UML-діаграми класів для інформаційної системи з використанням середовища BoUML. Навчитись виявляти класи, атрибути, методи та взаємозв’язки між класами на основі технічного завдання та відображати їх у вигляді структурованої UML-діаграми.

2.2 Організація виконання лабораторної роботи

Для створення діаграм UML в лабораторних роботах використовується програма BoUML версії 7.11 Patch 3 (дата релізу 25 березня 2025). Програма підтримує сучасних дистрибутивів Linux, включаючи Debian Bullseye, Fedora, CentOS, Arch Linux, а також Windows та macOS. Бідьш детально ознайомитись та встановити можна за посиланням https://www.bouml.fr/.

Розглянемо процес створення простої діаграми класів, які представлені наступним кодом на мові програмування C++.

Нехай існує абстрактний базовий клас *BaseData*, який окрім конструктору та деструктору містить чистий віртуальний метод *print()*. Реалізація методу *print()* передбачена в класі *IntData*, нащадку класу *BaseData*. В класі *IntData* передбачена

приватна змінна для зберігання значення цілого типу m_data. Ініціалізація *m_data* відбувається в конструкторі класу *IntData*. Віртуальна функція-член класу *IntData* реалізує виведення на консоль значення *m_data*.

При першому запуску програми *BoUML* потрібно встановити значення змінних оточення програми. Наприклад, можна встановити значення Own identifier в 2, для Character set в MS Windows обрати windows-1251 або UTF-8 для сучасних Unix-сумісних операційних систем (ОС), що продемонстровано на рисунку. 2.1. Якщо в процесі роботи потрібно буде вносити зміни в оточення, то можна скористатися пунктом меню Miscellaneous \ Set environment.

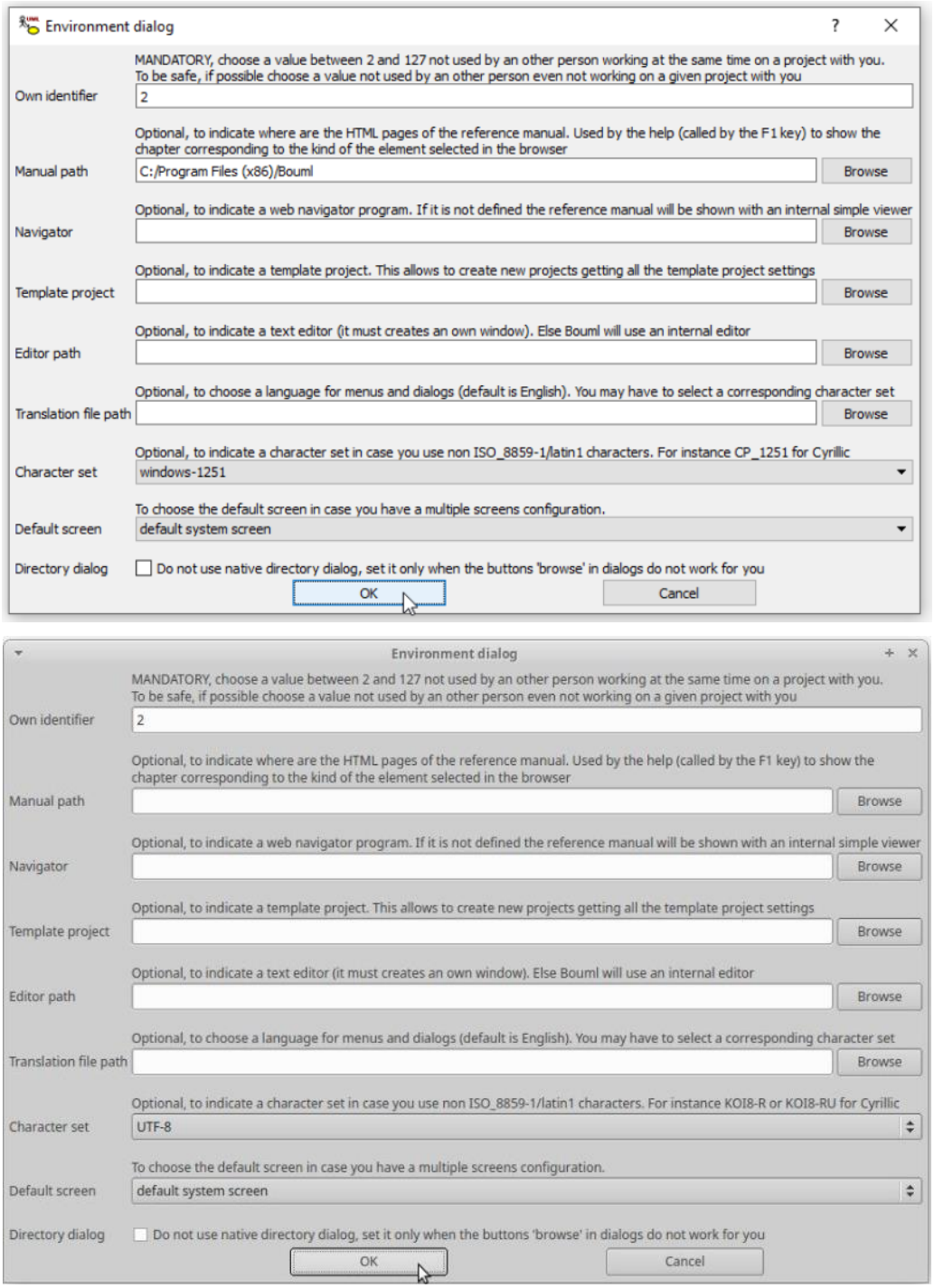


Рисунок 2.1 – Налаштування оточення BoUML при першому запуску у ОС: а) Windows; б) Unix-подібних ОС

Виконайте наступні дії:

8. В ОС на диску C: або Z: , або в іншому доступному підкаталогу, куди є доступ на запис, створіть каталог *uml_projects*.

9. В середовищі *BoUML* оберіть пункт меню Project \ New, оберіть створений каталог *uml_projects* та введіть ім’я проекту uml1. Натиснути кнопку Зберегти (тут та надалі розглядається робота в локалізованій, українській версії ОС). Закрийте інформаційний діалог.

10. Оберіть пункти меню *Languages \ C++ management ...* та *Languages \Verbose code generation.*

11. Натисніть праву кнопку миші на назві проекту та оберіть пункт меню *New class view* (рисунок 2.2). В діалозі введіть назву відображення класів. Наприклад, ClassView1.

12. В контекстному меню ClassView1 оберіть пункт *New class diagram* та в діалозі задайте ім’я нової діаграми класів, наприклад, ClassDiagram1. Після цього зробіть подвійний клік лівою кнопкою миші по назві ClassDiagram1. В результаті відкриється вікно для побудови діаграми класів. Подібним чином починають будувати діаграми й інших типів.

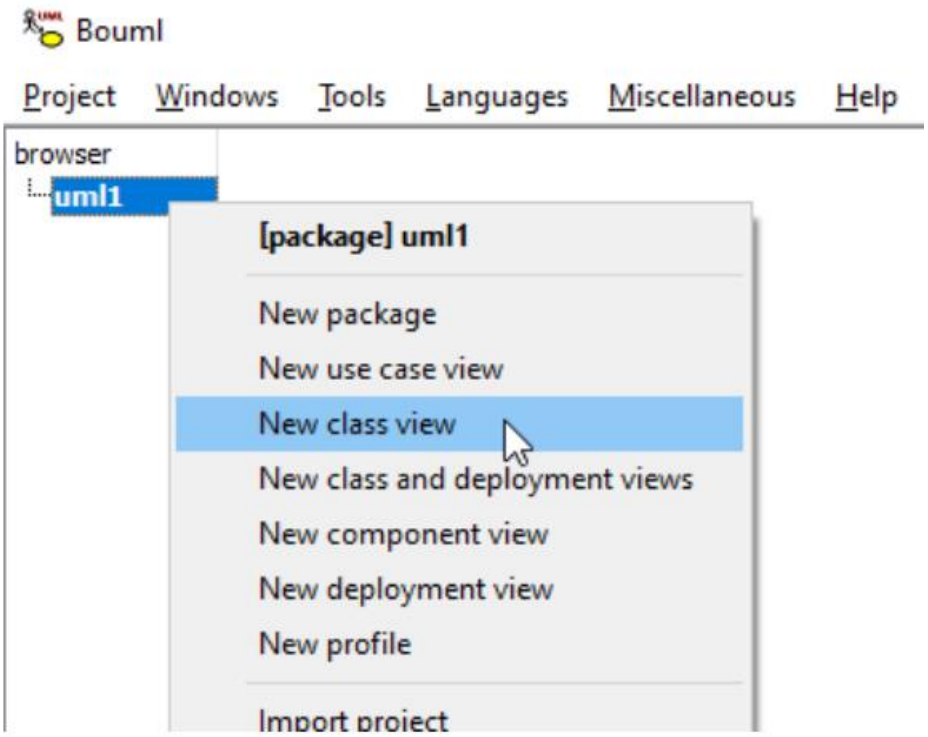


Рисунок 2.2 – Обрання UML-відображення

13. У верхньої панелі інструментів вікна ClassDiagram1, знайдіть кнопку *Add Clas*s, натисніть на неї, а потім перемістіть курсор миші у вільне поле вікна ClassDiagram1 та натисніть ліву кнопку миші. У вікно, що з’явилося на екрані, введіть назву класу – *BaseData*.

14. Знову натисніть на кнопку *Add Class* та розмістить у вікні прямокутник з новим класом *IntData*.

15. Зробіть одиночний клік мишею по прямокутнику класу *BaseData* і оберіть з контекстного меню пункт *Add operation* (рисунок 2.3). Тим самим

починається створення опису нової функції-члена класу (в різних мовах програмування може бути й інше визначення, наприклад, метод класу, а в контексті UML – операція).

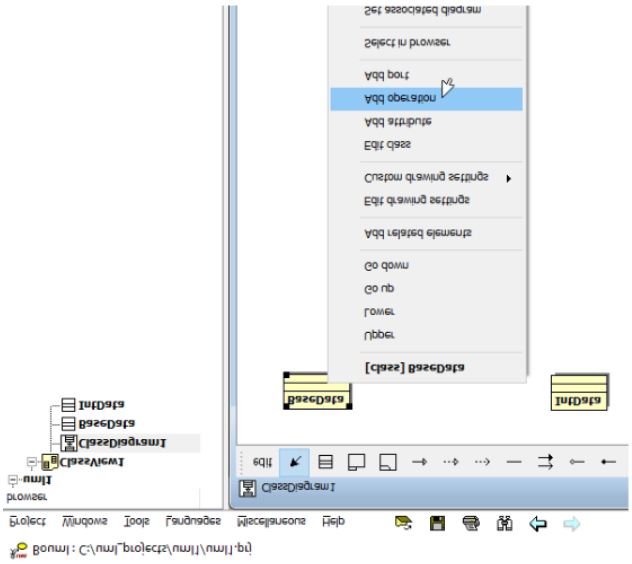


Рисунок 2.3 – Додавання нової операції

16. В діалоговому вікні введіть ім’я операції – *BaseData*, а в наступному діалогу *Operation dialog* в полі введення *stereotype* введіть стереотип *constructor*. Після цього натисніть на OK.

17. Натисніть пункт меню *Project \ Edit \ Edit drawing settings*.

18. В діалоговому вікні налаштувань властивостей відображення, на закладці class [1] встановіть наступні значення:

classes drawing mode: class
show classes members full definition: yes
show classes members visibility: yes
show classes members stereotype: yes
show members multiplicity: yes
show attribute initialisation: yes
show attribute modifiers: yes
Натисніть на кнопку OK.

19. Додайте нову операцію з іменем ~BaseData та у діалозі операції перейдіть на закладку C++ і оберіть опцію *virtual*. Натисніть на кнопку OK.

20. Додайте нову операцію з іменем print та у діалозі операції в полі *value type* вкажіть *void*, оберіть опцію *abstract*, перейдіть на закладку C++ і оберітьопцію virtual. Натисніть на кнопку OK. Таким чином створений абстрактний базовий клас.

21. Виконайте клік на прямокутнику класу *IntData* і у контекстному вікні оберіть пункт меню *Add attribute*. Задайте ім’я змінній-члену класу (атрибуту класу в контексті UML) – *m_data*. В діалогу налаштувань атрибуту класу в полі *type* введіть тип *int*. Зверніть увагу, що за замовчанням змінна розташована в *private*-частині класу. Натисніть на кнопку OK.

22. В клас *IntData* додайте операцію *IntData*. В діалогу налаштувань властивостей операції в полі *stereotype* введіть стереотип *constructor*. В частині *parameters*, в поле *Name* введіть назву першого параметру конструктора: *data*; в

поле *Type* вкажіть *int*, в полі *Default value* вкажіть значення 0. Натисніть на кнопку OK.

23. Додайте в клас *IntData* віртуальний деструктор з іменем *~IntData* за аналогією деструктора класу *BaseData*.

24. Натисніть на кнопку *Generalisation*, розташовану серед інструментів вікна ClassDiagram1. Переведіть курсор на клас IntData, натисніть ліву кнопку миші та не відпускаючи її, переведіть курсор на прямокутник класу BaseData і відпустить кнопку миші. Тепер на діаграмі класів позначено, що клас IntData є нащадком класу BaseData.

25. Відкрийте контекстне меню класу IntData та оберіть пункт Add inherited operation \ BaseData::print() : void.

26. Збережіть проект, обравши пункт меню Project \ Save або натиснувши на відповідну кнопку.Розблену діаграму можна зберегти відразу, наприклад, в графічному растровому PNG-форматі. Для цього натисніть праву кнопку миші на вільному полі у вікні ClassDiagram1 та оберіть пункт меню Save optimal picture part (png). Приклад розробленої діаграми представлений на рисунку. 2.4.

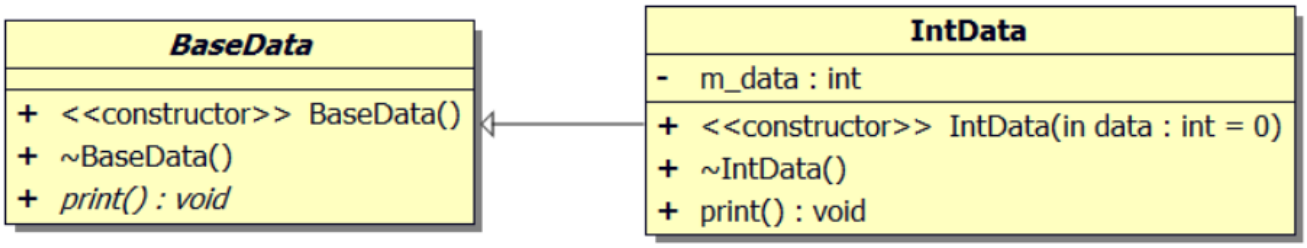


Рисунок 2.4 – Приклад діаграми класів

Наступним кроком може бути розробка коду, що реалізує цю діаграму класів. Середовище BoUML також дозволяє виконувати генерування коду обраної мови за розробленою діаграмою класів. Це процес буде розглянутий в наступній лабораторній роботі.

Перейдемо до розробки коду за вказаною діаграмою. Розмістимо все, що відноситься до класу *BaseData* в файли BaseData.h та BaseData.cpp; все, що відноситься до класу *IntData* в файли IntData.h та IntData.cpp, а також зробимо тестовий головний модуль test_data.cpp з головною функцією *main()*.Файли розмістимо у підкаталогу *uml1\src*. Вміст файлів представлений в додатку Б.

Команда збірки в ОС MS Windows передбачає, що на комп’ютері вже був встановлений IDE Code::Blocks з компілятором проекту MinGW-W64 та відбулися налаштування змінній оточення PATH, в якій зазначений шлях до розташування компілятору.

На основі попередньо створеної UML-діаграми класів та розглянутих концепцій об’єктно-орієнтованого програмування, наступним кроком є практична реалізація передбачених структур у вигляді Java-класів. Це дозволить перевірити коректність моделі, уточнити зв’язки між класами та забезпечити початкову функціональність системи. Необхідно зробити наступне:

1. Представити реалізації класів з теоретико-практичної частини на мові програмування Java.

2. Розширити представлену в теоретико-практичній частині діаграму класів класом DoubleData з реалізацією.

3. Додати до існуючих класів можливість введення даних з консолі, операції призначення даних та їх повернення (set/get-методи).

4. Розширити створену діаграму ієрархією класів для обробки динамічних масивів даних-об’єктів різних типів (IntData та DoubleData). При розробці цієї частини діаграми, врахувати певні відносини, між новими класами динамічних масивів та вже існуючими класами IntData та DoubleData.

2.3 Зміст звіту

Підготувати звіт з виконання лабораторної роботи, який повинен включати:

– номер, тему і мету лабораторної роботи;
– короткий огляд основних понять;
– опис завдання з початковими умовами;
– опис розроблених діаграм класів (особливо цікавлять *IntData* та *DoubleData*);
– лістинг початкових реалізацій класів (вказати роль класів, методів, змінних);
– тестування (Короткий опис прикладів використання з прикріпленими скріншотами);
– висновки.

2.4 Питання для підготовки до захисту лабораторної роботи

1. Що таке діаграма класів UML і для чого вона використовується?
2. Які основні елементи діаграми класів?
3. Яка різниця між асоціацією, агрегацією та композицією між класами?
4. Як позначаються атрибути та методи класу на діаграмі класів?
5. Що означають модифікатори доступу +, -, # в UML класах?
6. Як відображається спадковість між класами?
7. Що таке інтерфейс і як він позначається на діаграмі класів?
8. У чому полягає відмінність між статичним та динамічним зв’язком класів?

3 ЛАБОРАТОРНА РОБОТА №3
РОЗРОБКА UML-ДІАГРАМ КОМПОНЕНТІВ ТА РОЗГОРТАННЯ

3.1 Мета лабораторної роботи

Ознайомитися з процесом побудови UML-діаграм компонентів та діаграм розгортання за допомогою програмного середовища BoUML. Набути практичних навичок моделювання архітектури програмного забезпечення на компонентному рівні та апаратного середовища його функціонування. Ознайомитись з призначенням та структурою кожного типу діаграм, навчитись правильно відображати взаємозв'язки між компонентами системи та елементами інфраструктури.

3.2 Організація виконання лабораторної роботи

Розглянемо процес побудови в BoUML діаграми розгортання. Цей тип діаграм допомагає BoUML генерувати файли source-коду.

Початкова вимога – існують класи: абстрактний базовий *BaseData* та його нащадок *IntData*. Пройдемо на прикладі процес генерування каркасу коду, створивши елементи діаграми розгортання. Для генерування непотрібне повне відображення діаграми. Її можна зробити згодом.

1. Перед генеруванням коду потрібно встановити відповідні налаштування шляхів. Виконайте наступне:

– відкрийте файловий провідник MS Windows та створіть підкаталог *uml_projects \ uml1 \ code;*

– відкрийте діалог налаштувань опцій генерування коду по пункту меню *Project \ Edit \ Edit generation settings*;

– перейдіть на закладку Directory та натисніть напроти пункту *C++ root dir* кнопку *Set it absolute*;

– в поле введення сформуйте рядок (як приклад): *C:/uml_projects/uml1/code/* та натисніть на кнопку OK.

2. Відкрийте проект uml1 та в контекстному меню проекту оберіть пункт меню *New deployment view* (рисунок 3.1). В діалоговому вікні введіть ім’я діаграми. Наприклад, *DeploymentView1*.

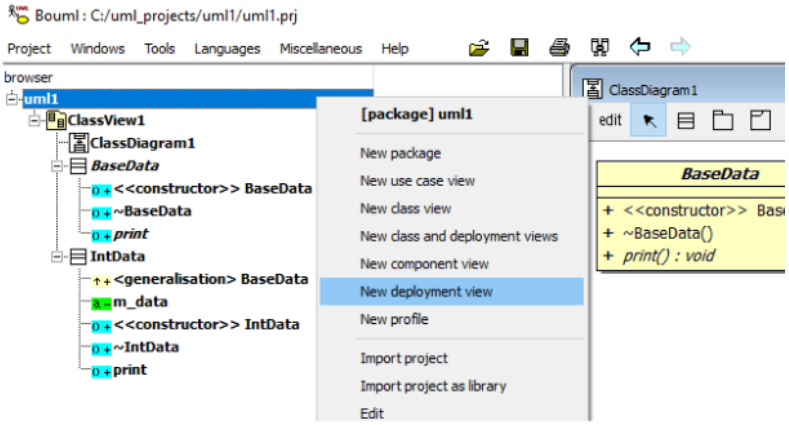


Рисунок 3.1 – Обрання побудови нової діаграми розгортання

3. В контекстному меню DeploymentView1 оберіть пункт *New artifact*. В діалоговому вікні введіть ім’я нового артефакту – *BaseData*. Таким же чином створіть артефакт *IntData*.

4. Зробіть подвійний клік лівою кнопкою миші на артефакті *BaseData* і в списку *stereotype* оберіть *source*. Перейдіть на закладку *Associated classes* та перенесіть зі списку *BaseData*-клас з лівої частини в праву. Натисніть кнопку OK.

5. Повторіть пункт 4 для артефакту *IntData*, зв’язавши його з класом *IntData*.

6. Відкриваючи властивості класів *BaseData* та *IntData*, переконайтеся, що на закладці *Uml* в поле *artifact* буде визначений потрібний *source*-модуль (артефакт).

7. Відкрийте діалог властивостей артефакту *BaseData*, перейдіть на закладку *C++ header* та натисніть кнопку *Default definition*. Перейдіть на закладку *C++ source* та натисніть кнопку *Default definition*. Натисніть кнопку OK.

8. Повторить пункт 7 для артефакту *IntData*.

9. Створіть новий артефакт main, надайте йому стереотип source, перейдіть на закладку *C++ source* та в полі *Source file definition* введіть фрагмент коду:

```
#include "IntData.h"
int main()
{
        IntData data(10);
        data.print();
        return 0;
}
```

10. Виконайте пункт *Tools \ Generate C++*.

Якщо все до цього було зроблено вірно, то в підкаталозі uml1 \ code будуть згенеровані п’ять файлів (cpp- та h-файли) з каркасом коду.

Розмістимо створені артефакти на діаграмі розгортання.

1. В контекстному меню DeploymentView1 оберіть пункт New deployment diagram. Дайте назву для нової діаграми, наприклад, DeploymentDiagram1.

2. Курсором миші перетягніть з дерева елементів проекту артефакти BaseData, IntData та main у вільне поле вікна діаграми розгортання.

3. Створіть новий артефакт test_data (в роботі №2 відбувалось тестування об’єктів класу IntData), відкрийте вікно його властивостей та у списку stereotype оберіть executable. Натисніть на кнопку OK. Перетягніть main у вікно DeploymentDiagram1.

4. В панелі вікна DeploymentDiagram1 натисніть на інструмент малювання стрілки Inheritance та намалюйте стрілку курсором миші від IntData до BaseData. Натисніть на інструмент малювання стрілки Dependency та намалюйте стрілку курсором миші від main до IntData. Натисніть на інструмент малювання стрілки Association та намалюйте стрілку курсором миші від test_data до main.

5. Викличте Generalisation dialog подвійним кліком миші на стрілці успадкування та у списку stereotype оберіть стереотип from. Натисніть на кнопку OK.

6. В панелі вікна DeploymentDiagram1 натисніть на Add Deployment Node, переведіть курсор на вільне поле діаграми та клікніть по полю вікна. Дайте назву обчислювальному вузлу, наприклад, Server. Натисніть на кнопку OK.

7. В дереві елементів проекту виконайте подвійний клік лівою кнопкою миші по створеному обчислювальному вузлу розгортання та в діалозі в списку *stereotype* оберіть *executionEnvironment*. Натисніть на кнопку OK.

8. На зображені обчислювального вузла натисніть праву кнопку та оберіть пункт меню *Go down* змінивши порядок відображення елементів діаграми.

9. Перемістить кнопкою миші зроблені артефакти в середину обчислювального вузла.

10. Створіть ще один обчислювальний вузол з іменем *Remote Workstation*.

11. З’єднайте обидва вузла за допомогою інструменту *Network* з панелі інструментів вікна *DeploymentDiagram1*.

12. Розмістить коментар за допомогою інструменту *Note* з панелі інструментів вікна *DeploymentDiagram1*. Текст коментаря: Gigabit Ethernet.

13. За допомогою інструменту *Anchor* з панелі інструментів вікна DeploymentDiagram1, з'єднайте прямокутник коментаря з лінією Network.

Результат діаграми розгортання представлений на рисунку 3.2.

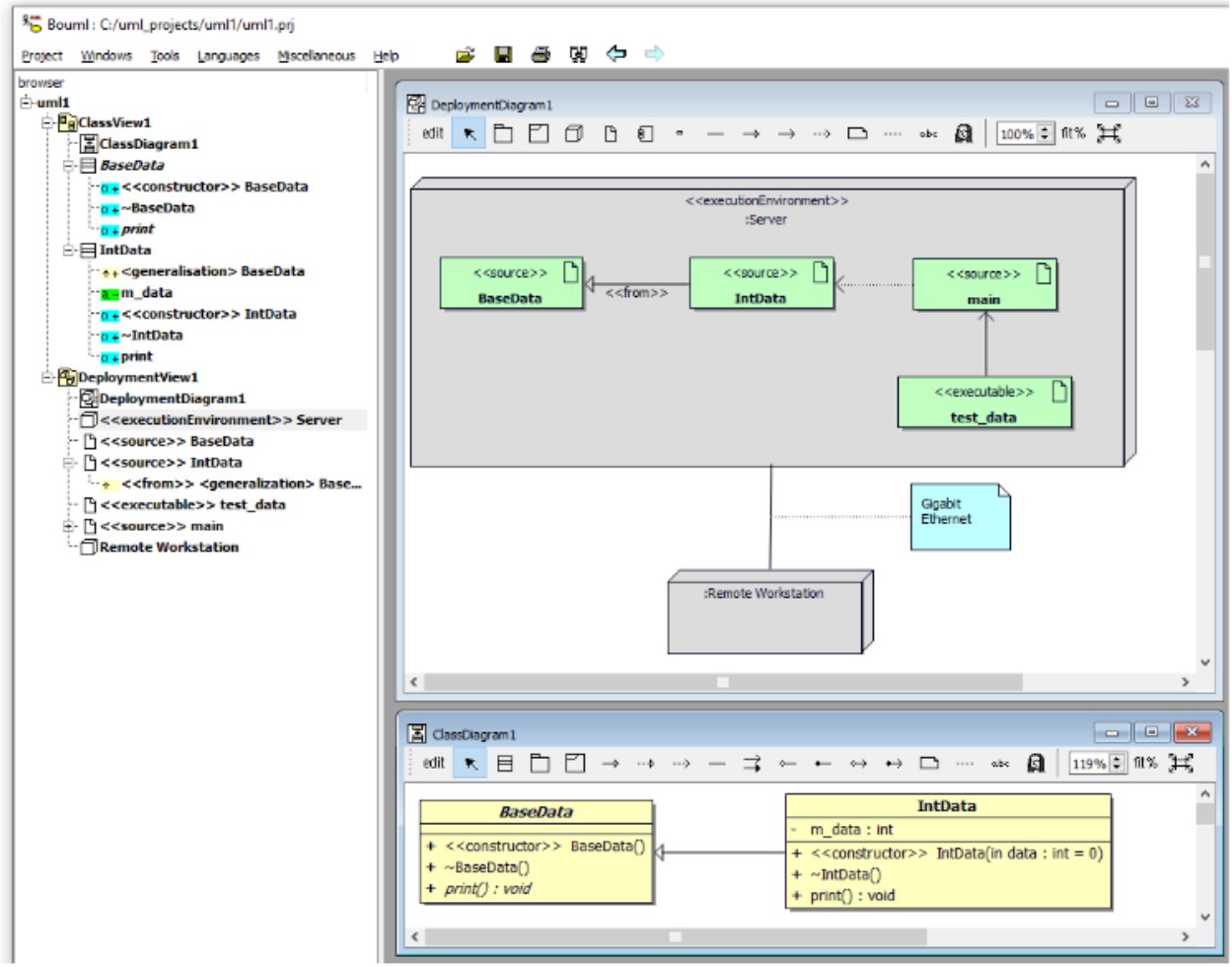


Рисунок 3.2 – Приклади розроблених UML-діаграм класів та розгортання в середовищі BoUML

Створимо невеличкий приклад діаграми компонентів. Нехай потрібно на мові C++ розробити компонент, який оперує множиною реалізацій інтерфейсу запису в потік. Відомо, що на мові C++ не є потребою створювати інтерфейси, але в певних випадках це також можна зробити. Інтерфейси на мові програмування C++ реалізують через абстрактні класи, в яких присутні тільки чисто віртуальні методи. Приклад реалізації:

14. В контекстному меню *ClassView1* (з проекту uml1) оберіть пункт *New class diagram* та надайте ім'я новій діаграмі класів, наприклад, ClassDiagram2.

15. Розмістіть в *ClassDiagram2* новий клас, який далі буде представлений як інтерфейс. Задайте йому ім'я *IStreamWriter*. Часто в якості префіксу, який показує, що елемент є інтерфейсом, використовують символ 'I'. У властивостях нового елементу в списку *stereotype* оберіть *interface*.

16. Розмістіть в *ClassDiagram2* нові класи *StreamConsoleWriter* та *StreamBinaryWriter*.

17. В панелі інструментів *ClassDiagram2*, натисніть мишею на інструмент *Realize* та з'єднайте клас *StreamConsoleWriter* та інтерфейс *IStreamWriter* (в напрямку зображення інтерфейсу). Виконайте також дії з класом *StreamBinaryFileWriter*.

18. Розмістіть в *ClassDiagram2* нові класи *ComponentWriter* та *ClientWriter*.

19. В панелі інструментів *ClassDiagram2*, натисніть мишею на інструмент *Aggregation* та з'єднайте клас *ComponentWriter* та інтерфейс *IStreamWriter*. (у напрямку *IStreamWriter*). Натисніть мишею на інструмент *Directional Association* та з'єднайте клас *ClientWriter* та інтерфейс *IStreamWriter*. (у напрямку *IStreamWriter*).

20. В контекстному меню проекту *uml1* оберіть пункт *New component view*. Задайте ім'я нового виду компонентів, наприклад, *ComponentView1*.

21. В контекстному меню *ComponentView1* оберіть пункт *New component diagram* та задайте ім'я, наприклад, *ComponentDiagram1*.

22. Оберіть пункт меню *Project \ Edit \ Edit drawing settings*. В діалогу перейдіть на закладку component та виставте опціям такі значення:

show component's required and provided interfaces: yes

show component's realizations: yes

show stereotype properties: yes

Натисніть на кнопку OK.

23. У вікні діаграми компонентів натисніть на інструмент *Add Component* та потім зробіть клік на вільному полі вікна *ComponentDiagram1*. Задайте ім'я компоненту – *ComponentWriter*.

24. Відкрийте діалог властивостей *ComponentWriter* та зі списку *stereotype* оберіть implement. На закладці *Realizing classes* (реалізація класів) оберіть клас *ComponentWriter* на перемістіть його вправу частину. На закладці *Provided classes* (класи, що надаються) оберіть класи *StreamConsoleWriter* та *StreamBinaryFileWriter* та перемістіть їх в праву частину. Натисніть на кнопку OK.

25. У вікні діаграми компонентів натисніть на інструмент *Add Component* та потім зробіть клік на вільному полі вікна *ComponentDiagram1*. Задайте ім'я компоненту – *ClientWriter*.

26. Відкрийте діалог властивостей *ClientWriter*. На закладці *Realizing classes* (реалізація класів) оберіть клас *ClientWriter* на перемістіть його вправу частину. На закладці *Required classes* (необхідні класи) оберіть класи *StreamConsoleWriter* та *StreamBinaryFileWriter* та перемістіть їх в праву частину. Натисніть на кнопку OK.

27. У вікні діаграми *ComponentDiagram1* натисніть на інструмент *Required interface* та проведіть курсором миші від компоненту *ClientWriter* у напрямку *ComponentWriter*, але зупиніться на вільному полі вікна діаграми. З діалогу оберіть *StreamBinaryFileWriter*. Повторіть дії, обравши інтерфейс *StreamConsoleWriter*.

28. У вікні діаграми *ComponentDiagram1* натисніть на інструмент *Provided interface* та проведіть курсором миші від компоненту *ComponentWriter* у напрямку *ClientWriter*, але зупиніться на вільному полі вікна діаграми. З діалогу оберіть *StreamBinaryFileWriter*. Повторіть дії, обравши інтерфейс *StreamConsoleWriter*.

Вирівняйте з'єднання інтерфейсів, як показано на рисунку 3.3.

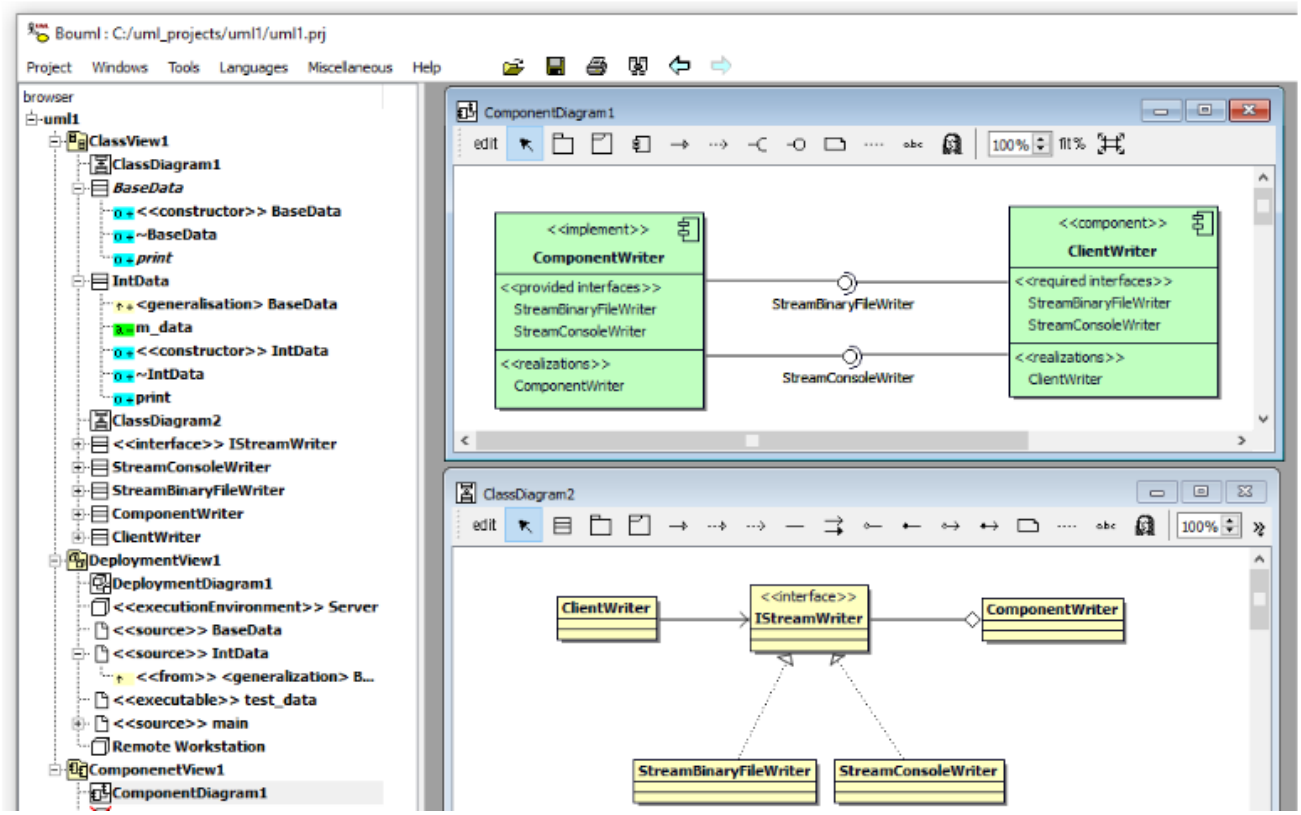


Рисунок 3.3 - Приклад реалізації діаграми компонентів

Самостійно в лабораторній роботі необхідно буде зробити:

1. Побудувати діаграму розгортання для генерації шаблону коду на Java для створеної в лабораторній роботі 2 діаграми класів, з використанням динамічних масивів даних.

2. Розширити представлену діаграму класів, яка використовується для побудови діаграми компонентів, методами (як приклад):

bool Open(String fileName, int mode);

void Close();

bool Write(Object Buffer);

Перелічені функції повинні бути в інтерфейсі IStreamWriter. Можливий опис та використання додаткових методів на власний розсуд, які не заперечать лозіці.

В звіті надати модифіковану діаграму класів, діаграми компонентів та розгортання, а також код програмних модулів з результатами тестування програм. Продемонструвати розроблені діаграми та програми викладачу.

3.3 Зміст звіту

Підготувати звіт з виконання лабораторної роботи, який повинен включати:

– номер, тему і мету лабораторної роботи;

– теоретичні відомості (короткий опис діаграми класів, діаграми компонентів та діаграми розгортання)

– практична частина, яка містиме:

a. Діаграма класів з урахуванням інтерфейсу IStreamWriter;
b. Діаграма компонентів, побудована на основі модифікованої діаграми класів;
c. Діаграма розгортання для генерації шаблону коду на Java;

d. Код програмних модулів (реалізація інтерфейсу, приклад використання);
e. Результати тестування (вивід у консолі);

– висновки.

3.4 Питання для підготовки до захисту лабораторної роботи

1. Що таке діаграма компонентів в UML і яку роль вона відіграє в розробці програмного забезпечення?
2. Які основні елементи містить діаграма компонентів?
3. Що таке інтерфейс у контексті UML-діаграм компонентів? Як його позначають?
4. Як позначається залежність одного компонента від іншого?
5. Які основні елементи включає діаграма розгортання?
6. У чому полягає відмінність між вузлом (node) і артефактом (artifact)?
7. Як діаграма розгортання відповідає реальному середовищу виконання програми?

4 ЛАБОРАТОРНА РОБОТА №4
РОЗРОБКА UML-ДІАГРАМ КОМПОНЕНТІВ ТА РОЗГОРТАННЯ

4.1 Мета лабораторної роботи

Розробити в середовищі BoUML діаграму класів, що моделює ядро графічної підсистеми програми-візуалізатора просторових векторних даних. Сформувати об'єктно-орієнтовану ієрархію класів, яка відображає ключові типи просторових об'єктів (точки, лінії, полігони тощо) та їхні властивості й взаємозв’язки. Реалізувати основні методи для опрацювання та візуалізації цих об’єктів мовою Java. Набути навичок аналізу та проєктування програмної архітектури з використанням UML, а також навичок програмної реалізації класів відповідно до побудованої моделі.

4.2 Організація виконання лабораторної роботи

Класи-сутності (entity class) або модельні класи визначають суть будь-якої інформаційної системи. Аналіз вимог спрямований переважно на виявлення класів-сутностей. Однак, для функціонування системи потрібні також класи іншого типу.

Користувачам системи необхідні класи, які визначають GUI-об'єкти, так звані прикордонні класи (boundary classes, класи уявлення (view classes)). Щоб функціонувати належним чином, системі також необхідні класи, які керують програмної логікою – керуючі класи (control classes).

Залежно від конкретного підходу до моделювання прикордонні та керуючі класи можуть розглядатися або не розглядатися на деякому рівні уявлення в ході аналізу вимог.

Нехай необхідно спроектувати просту програму-візуалізатор просторових даних, представлених в графічному, векторному вигляді точковими, лінійними та полігональними об'єктами. Розглянемо аналіз та виділення класів-сутностей і

керуючих класів програми.

Основними вимогами до зберігання та представлення даних в програмі є такі.

1. Точкові, лінійні та полігональні об'єкти представлені для відображення в деякій локальній системі координат (використовуються координати x і y).

2. Для відображення множини векторних однотипових даних вони групуються в окремий шар.

3. Інформація про властивості відображення векторних даних конкретного шару, повинна бути представлена в окремому файлі опису.

4. Для спрощення завдання не розглядається атрибутивна інформація, яка пов'язана з векторними об'єктами.

5. Для відображення множини векторних шарів вони повинні бути згруповані в проекті відображення та описані в окремому файлі.

6. Всі файли з даними є текстовими, форматованими ASCII-файлами.

7. Файл проекту та файли шарів проекту розташовуються в одному каталозі.

Основними вимогами до програми є наступні.

1. Програма завантажує дані з файлу-проекту певного формату.

2. Програма завантажує дані кожного векторного шару та інформацію про властивості відображення в пам'ять.

3. Після завантаження даних в пам'ять програма відображає їх в головному вікні.

4. При спробі завантаження іншого проекту головне вікно програми очищується та виділена під дані пам'ять очищаються.

5. Повинен бути передбачений пункт меню завершення роботи з програмою.

Нехай файли з даними мають наступні розширення:

– файли проектів візуалізації: .prj;

– файли шарів даних: .lay;

– файли описів властивостей об'єктів шарів: .prop.

Опис фалів взяти з додатку В.

Таким чином, відразу виділяються наступні класи:

1. Project (управляє проектом).

2. Layers (управляє шарами проекту).

3. GeoPoints (зберігає і управляє об'єктами-точками).

4. GeoPolygons (зберігає і управляє полігональними об'єктами).

5. GeoLines (зберігає і управляє лінійними об'єктами).

6. Point (клас/структура з полями X та Y для зберігання координат вузлів або точок).

Більшість властивостей та операцій об'єктів GeoPoints, GeoLines, GeoPolygons можуть бути об'єднані у класі GeoObjects. Багато властивостей та операції класів GeoLines і GeoPolygons можуть бути об'єднані в класі GeoPolygons. Управління властивостями шарів логічно винести в окремий клас GeoProperties.

Кількість об'єктів зазначених класів в програмі (n – кількість залежить від зазначеної кількості шарів певного типу в файлі проекту, m – кількість залежить від кількості вузлів лінійних/полігональних об'єктів або кількості точкових об'єктів в шарах проекту):

Project – 1;

Layers – 1;
GeoPoints – n;
GeoPolygons – n;
GeoLines – n;
Point – m.

Опис класів проекту надано в додатку Г.

Самостійно в лабораторній роботі необхідно буде зробити:

1. Використовуючи аналіз вимог та опис класів програми-візуалізатору, що представлені в теоретичній частині, розробити в середовищі BoUML діаграму класів проекту.

2. Використовуючи середовище розробки NetBeans або IntelliJ IDEA, запрограмувати методи класів (за винятком методів draw).

3. Для тестування спроектованої програми використовувати наступний код:

```
package gisviewer;

public class SimpleGisViewerTester
{
    public static void main(String[] args)
    {
        String prjFileName = "testproject.prj";
        Project m_prj = new Project();
        if ( !m_prj.isLoaded( ) )
        {
            if( m_prj.loadProject( prjFileName ) )
            {
                int count = m_prj.m_layers.count();
                for( int i=0; i<count; i++ )
                {
                    m_prj.m_layers.debugprint( );
                }
                m_prj.close();
                m_prj = null;
            }
        }
    }
}
```

В якості вихідних даних взяти файли *testproject.prj* , *layer1.lay, layer1.prop* з додатку В.

Для тестування розширте проект файлами для *layer2* та *layer3* з описом лінійних та полігональних об'єктів.

У звіті привести розроблену діаграму класів і пояснення до неї.

Програмний код реалізації діаграми класів, зміст тестових даних та результатитестування програми-візуалізатору в консольному режимі.

4.3 Зміст звіту

Підготувати звіт з виконання лабораторної роботи, який повинен включати:
– номер, тему і мету лабораторної роботи;
– теоретичні відомості;
– практична частина, яка містиме:
  a. Діаграма компонентів, побудована на основі модифікованої діаграми класів;
  b. Опис зв’язків між класами (асоціації, композиції, успадкування)
  c. Код програмних модулі;
  d. Результати тестування (вивід у консолі);
– висновки.

4.4 Питання для підготовки до захисту лабораторної роботи

1. У чому полягає різниця між діаграмою компонентів і діаграмою розгортання?

2. Як зв'язуються компоненти між собою? Які UML-елементи для цього використовуються?

3. Що таке слабке та сильне зв'язування (loose/coupling, cohesion) у контексті компонентів?

4. Яким чином у вашій системі відображено роботу з файлами .prj, .lay, .prop?

5. Які переваги має поділ системи на компоненти?

6. Які стандарти чи специфікації UML застосовувалися при створенні вашої діаграми?

7. Чи можна згенерувати код з діаграм у BoUML? Якщо так, то як?

5 ЛАБОРАТОРНА РОБОТА №5 УДОСКОНАЛЕННЯ ПРОГРАМИ ВІДОБРАЖЕННЯ ВЕКТОРНИХ ПРОСТОРОВИХ ОБ'ЄКТІВ

5.1 Мета лабораторної роботи

Удосконалити програмний код візуалізатора просторових векторних даних шляхом розширення функціональності, оптимізації логіки обробки графічних об'єктів та покращення структури програми. Для взаємодії користувача з програмою візуалізації просторових об'єктів необхідно реалізувати графічний інтерфейс користувача (GUI), який забезпечує основні функції управління програмою, зокрема відображення вмісту шару з файлів (.lay, .prop).

5.2 Організація виконання лабораторної роботи

Використовуючи середовище розробки NetBeans (або IntelliJ IDEA), внесіть код з додатку Д в розроблений в роботі №4 проект-пакет gisviewer.

Створити програмні коди захищеного методу doLoadLayer( ) класу GeoPolygons та загальнодоступного методу draw( ) класів GeoPoints, GeoPolygons і GeoLines. Метод повинен реалізовувати графічне відображення об’єктів на вказаній панелі (наприклад, на Graphics2D у Java AWT).Повинна враховуватись

інформація з властивостей відображення (розмір, колір, стиль). Реалізація має бути узгодженою для всіх типів об’єктів (точки, лінії, полігони).Коментарі мають пояснювати логіку малювання кожного типу об’єкта.

У звіті привести створені програмні коди з коментарями, приклади тестових даних та результати тестування програм.

5.3 Зміст звіту

Підготувати звіт з виконання лабораторної роботи, який повинен включати:

– номер, тему і мету лабораторної роботи;

– теоретичні відомості;

– практична частина, яка містиме:

a. Реалізація класу GeoProperties: відображення властивостей (колір, розмір, видимість);
b. Код програмних модулів (реалізація інтерфейсу, приклад використання);
c. Результати тестування (приклади тестових даних, вивід у консолі, скріншоти результатів відображення даних в програм);

– висновки.

5.3 Питання для підготовки до захисту лабораторної роботи

1. Яка мета розробки візуалізатора просторових векторних об'єктів? Які основні вимоги ставилися до програми?
2. Які класи реалізовано в програмі? Назвіть їх призначення.
3. Як працює метод draw() у класі GeoPoints?
4. Яку роль відіграють параметри кольору, стилю, видимості в методі draw()?
5. Що робить метод doLoadLayer() у класі GeoPolygons?
6. Як можна додати підтримку нових типів геометричних об’єктів?
7. Як розрізняються об’єкти за типом при завантаженні?

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Molodets B., Hnatushenko V., Boldyriev D., Bulana, T. Information System of Air Quality Assessment Using Data Interpolation from Ground Stations. CEUR Workshop Modern Machine Learning Technologies and Data Science Workshop (MoMLeT&DS 2023), Lviv 2023. Vol. 3426. P. 233–245. Режим доступу до ресурсу: https://ceur-ws.org/Vol-3426/

2. Molodets B., Hnatushenko V., Boldyriev D. , Bulana T. Information System of Air Quality Assessment Based of Ground Stations and Meteorological Data Monitoring. CEUR Workshop Intelligent Information Technologies & Systems of Information Security (IntelITSIS 2023). 2023.Vol. 3373, P. 206–216. Режим доступу до ресурсу: https://ceur-ws.org/Vol-3373/

3. ISO/IEC/IEEE 29148:2018(E): ISO/IEC/IEEE International Standard - Systems and Software Engineering -- Life Cycle Processes -- Requirements Engineering. : IEEE, 2018.

4. Tilley S. Bundle: Systems Analysis and Design, 12th + MindTap,1 Term Printed Access Card. Course Technology, 2019. 488 p. ISBN-10: 0357237641, ISBN-13: 978-0357237649.

5. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, 2020. 432 p. ISBN: 9781492043454.

6. Bernhard Rumpe. Agile Modeling with UML: Code Generation, Testing, Refactoring. Springer, 2017. 394 p. ISBN-10: 3319588613, ISBN-13: 978- 3319588612.

7. Bhuvan Unhelkar. Software Engineering with UML. Auerbach Publications, CRC PRESS, 2018. 427 p. ISBN-10: 1138297437, ISBN-13: 978-1-138-29743-2.

8. Allen B., Bosworth A. Systems Design: Building Systems That Drive Ideal Behavior. Productivity Press, 2022. 162 p. ISBN: 9781032213101.

ДОДАТОК А ПРИКЛАД ТИТУЛЬНОЇ СТОРІНКИ ДЛЯ ЗВІТУ З ЛАБОРАТОРНИХ РОБІТ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра інформаційних технологій та комп'ютерної інженерії

ЗВІТ
З ЛАБОРАТОРНИХ РОБІТ

дисципліна *"Проектування інформаційних систем"*

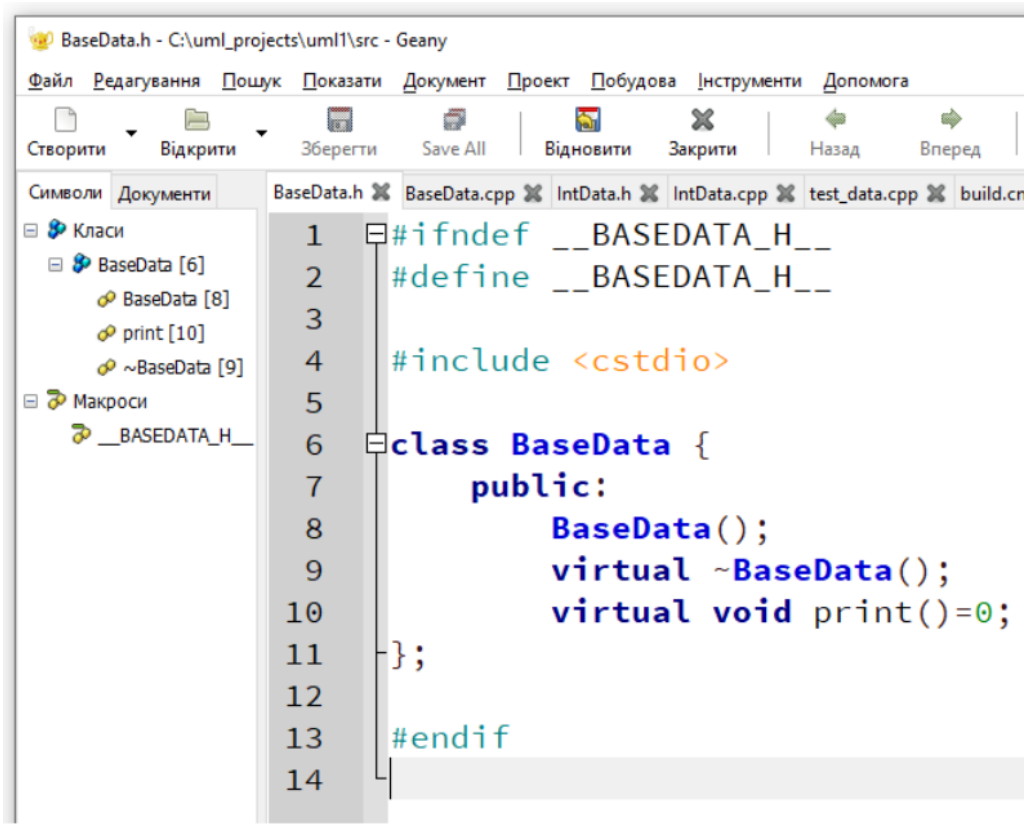
Виконавець,

здобувач гр. 126-24-1 _____ *Т.Г. Шевченко*
(підпис)

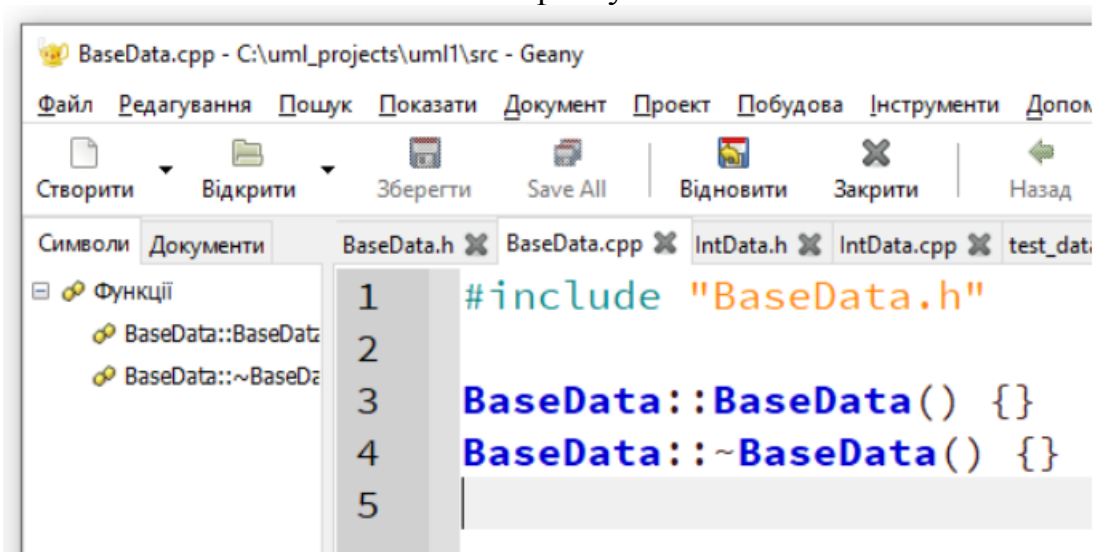
Керівник, доц. _____ *Б.В. Молодець*
(підпис)

Дніпро
НТУ «ДП»
2025

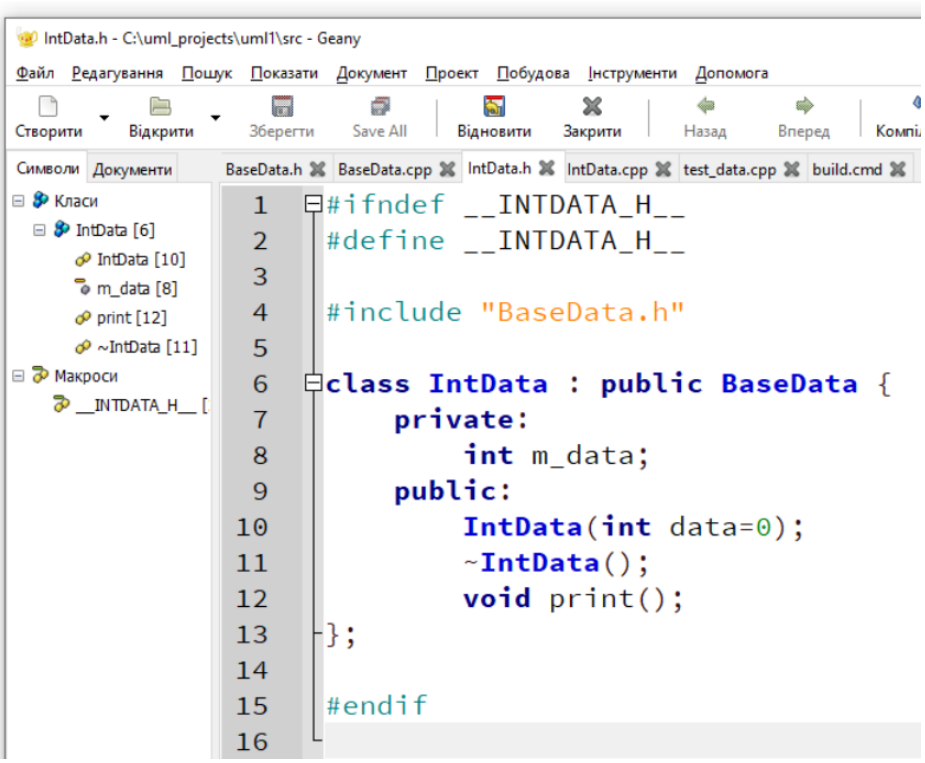
ДОДАТОК Б ЛІСТИНГ КОДУ ДЛЯ ЛАБОРАТОРНОЇ РОБОТИ 1



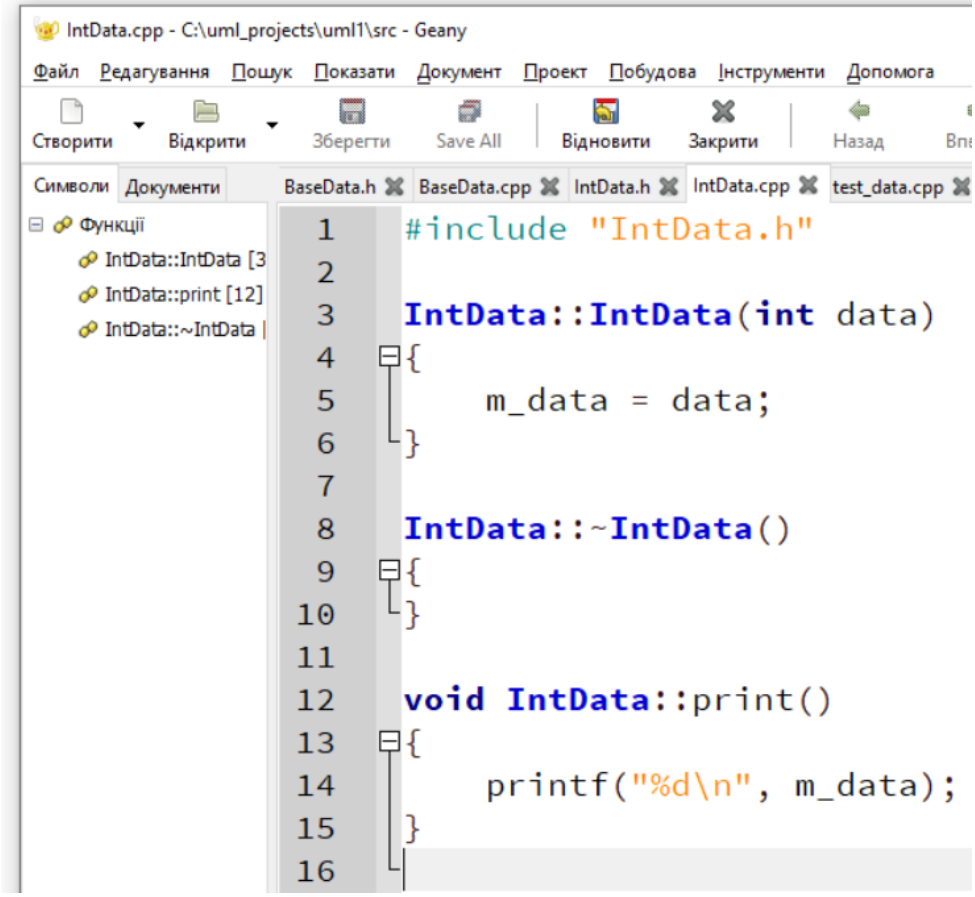
Вміст файлу BaseData.h



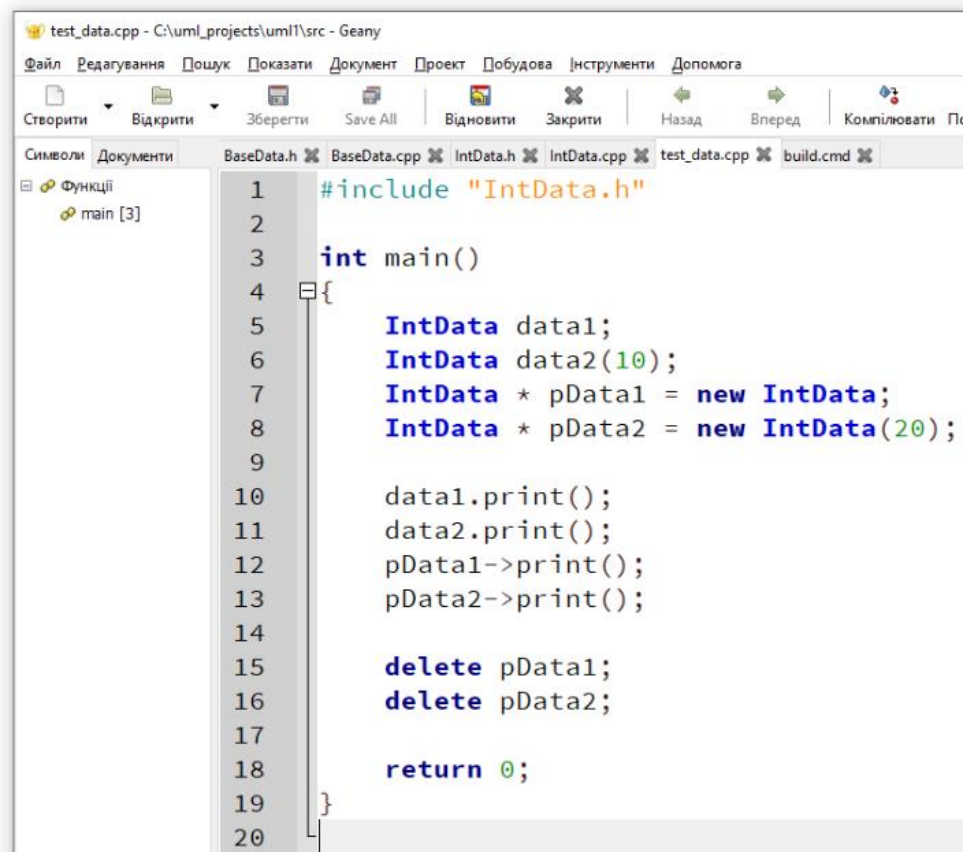
Вміст файлу BaseData.cpp



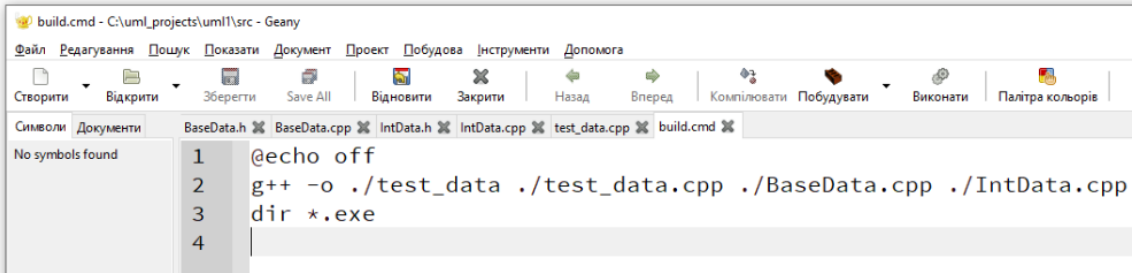
Вміст файлу IntData.h



Вміст файлу IntData.cpp



Вміст файлу test_data.cpp



Вміст файлу-скрипту збірки build.cmd

ДОДАТОК В ФОРМАТ ВХІДНИХ ФАЙЛІВ ДЛЯ ЛАБОРАТОРНОЇ РОБОТИ 1

N_Layers
Layer1.lay OBJ_TYPE
Layer2.lay OBJ_TYPE
. . .
LayerN.lay OBJ_TYPE

де N_Layers – кількість шарів відображення; Layeri.lay – файл конретного шару відображення; OBJ_TYPE – тип об'єктів шарів: POINTS, LINES, POLYGONS.

N_POINTS
X1 Y1
X2 Y2
. . .
XN YN

де N_POINTS – кількість точкових об'єктів в шарі; Xi, Yi – дійсні координати точкових об'єктів.

Файл опису геометрії лінійних і полігональних об'єктів має формат:

N_Objects
M1_VERTEXES
X_1_1 Y_1_1
X_1_2 Y_1_2
. . .
X_1_M1 Y_1_M1
M2_VERTEXES
X_2_1 Y_2_1
X_2_2 Y_2_2
. . .
X_2_M2 Y_2_M2
MN_VERTEXES
X_N_1 Y_N_1
X_N_2 Y_N_2
. . .
X_N_M Y_N_M

де N_Objects – кількість об'єктів шару; Mi_VERTEXES – кількість вузлів в і-тому лінійному або полігональному об'єкті; X_j_i, Y_j_i – координати вузлів векторних об'єктів.

Формат файлу опису властивостей відображення об'єктів шарів наступний:

VISIBLE
SIZE
COLOR
STYLE

де VISIBLE – властивість відображення шару; SIZE – розмір об'єктів шару; COLOR – колір відображення об'єктів шару; STYLE – стиль відображення об'єктів шару.

Файл *testproject.prj*:

```
1
layer1.lay POINTS
```

Файл *layer1.lay*:

```
3
1.5 2.5
3.33 7
1024 999
```

Файл *layer1.prop*:

```
1
8
255
0
```

ДОДАТОК Г ОПИС КЛАСІВ ПРОЕКТУ ДЛЯ ЛАБОРАТОРНОЇ РОБОТИ 4

Ім'я класу	Опис	Операція/атрибут
Project	надає доступ до завантаженим об'єктам-шарам	m_layers
	показує стан – завантажений проект або ні	lLoad
	завантаження проекту (файл .prj)	loadProject
	повертає стан – завантажений проект чи ні	isLoaded
	закриває відкритий проект	close
	опціонально: надає статичний метод, який повертає шлях до файлу шара	extractPathName
Layers	агрегує список шарів-об’єктів	m_layers
	додає новий шлях до списку шарів	addLayer
	повертає кількість завантажених шарів проекту	count
	очищує шари від об’єктів, а проект від шарів	clear
	для відладки: виводить на консоль завантажені дані шарів (координати вузлів/об’єктів, інформацію про властивості шарів)	debugprint
GeoProperties	зберігає властивість VISIBLE	m_visible
	зберігає властивість SIZE	m_size
	зберігає властивість COLOR	m_color
	зберігає властивість STYLE	m_style

	задає значення властивості VISIBLE	setVisible
	повертає значення властивості VISIBLE	getVisible
	задає значення властивості SIZE	setSize
	повертає значення властивості SIZE	getSize
	задає значення властивості COLOR	getSize
	повертає значення властивості COLOR	getColor
	задає значення властивості STYLE	setStyle
	повертає значення властивості STYLE	getStyle
GeoObjects	зберігає властивості об’єктів шару	m_properties
	статичне константне поле типу шару	POINTS = 0
		LINES = 1
		POLYGONS = 2
	завантаження даних шару з файлу .lay	loadLayers
	завантаження інформації про властивості шару з файлу .prop	loadProperties* (викликається з loadLayers)
	повертає властивості об’єктів шару	getProperties
	операція завантаження даних шару з файлу .lay	doLoadLayer** (викликається з loadLayers)
	операція малювання об’єктів шару у вікні GUI	draw***
	операція видалення об’єктів шару	clear***
	для відладки: виводить на консоль інформацію про об’єкти шару	debugprint***
GeoPoints	агрегує список об’єктів типу Point	m_pts
	реалізує абстрактні	doLoadLayer,

	методи	clear, debugprint, draw
GeoPolygons	агрегує список об’єктів шару (кожний елемент списку – список об’єктів типу Point)	m_obj*
	реалізує абстрактні методи	doLoadLayer, clear, debugprint, draw
GeoLines	реалізує абстрактний метод	draw
Point	зберігає значення координати X	x
	зберігає значення координати Y	y
	задає значення координат точки	Point

* – операція/атрибут класу розміщені в захищеній частині;
** – операція розміщена в захищеній частині класу, абстрактна;
*** – абстрактна операція, загальнодоступна.

ДОДАТОК Д ЛІСТИНГ КОДУ ДЛЯ ЛАБОРАТОРНОЇ РОБОТИ 5

```
package gisviewer;

class Point {
  public double x;
  public double y;

  Point(double _x, double _y) { x = _x; y = _y; }
}
```

Код модуля Point.java

```
package gisviewer;

class GeoProperties {
  private int m_visible = -1;
  private int m_size = -1;
  private int m_color = -1;
  private int m_style = -1;

  public void setVisible(int flVisible) { m_visible = flVisible; }
  public int getVisible() { return m_visible; }
  public void setSize(int size) { m_size = size; }
  public int getSize() { return m_size; }
  public void setColor(int color) { m_color = color; }
  public int getColor() { return m_color; }
  public void setStyle(int style) { m_style = style; }
  public int getStyle() { return m_style; }
}
```

Код модуля GeoProperties.java

```
package gisviewer;

import java.awt.Graphics;

class GeoLines extends GeoPolygons {

  @Override public void draw(Graphics gr) { }
}
```

Код модуля GeoLines.java

```java
package gisviewer;


import java.util.ArrayList;

class Layers {
    private ArrayList<GeoObjects> m_layers = new ArrayList<GeoObjects>();

    public boolean addLayer(String fileNameLayer, int layerType)
    {
        boolean res = false;
        GeoObjects obj = null;

        switch(layerType)
        {
            case GeoObjects.POINTS:    obj = new GeoPoints(); break;
            case GeoObjects.LINES:     obj = new GeoLines(); break;
            case GeoObjects.POLYGONS:  obj = new GeoPolygons(); break;
        }

        res = obj.loadLayer(fileNameLayer);
        if(res) m_layers.add(obj);

        return res;
    }

    public int count()
    {
        return m_layers.size();
    }

    public void debugprint()
    {
        int count = this.count();
        for(int i=0;i<count;i++)
        {
            GeoObjects obj = m_layers.get(i);
            obj.debugprint();
        }
    }

    public void clear()
    {
        int count = this.count();
        for(int i=0;i<count;i++)
        {
            GeoObjects obj = m_layers.get(i);
            obj.clear();
        }
        m_layers.clear();
    }
}
```

Код модуля Layers.java

```
package gisviewer;

import java.io.*;
import java.util.*;

class Project {
  public static String extractPathName(String fileName, char slash)
  {
      String res = "";
      int i, count = fileName.length();
      int pos = 0;
      for(i=0;i<count;i++) if(fileName.charAt(i)==slash) pos=i;
      for(i=0;i<=pos;i++) res+=fileName.charAt(i);
      return res;
  }

  public boolean loadProject(String fileName)
  {
      Scanner fin = null;
      flLoad = false;
      try
      {
        File file = new File(fileName);
        String path = Project.extractPathName(file.getAbsolutePath(), '/');

        fin = new Scanner(file);

        int countLayers = fin.nextInt();

        for(int i=0; i<countLayers; i++)
        {
            String layerFileName = path + fin.next();
            String layerType = fin.next();

            if(new File(layerFileName).isFile())
            m_layers.addLayer(layerFileName, Integer.parseInt(layerType));
        }
        fin.close();
        flLoad = true;
      }
      catch(FileNotFoundException e) { }

      return flLoad;
  }

  public boolean isLoaded() {  return flLoad; }

  public void close() {  flLoad = false;   m_layers.clear(); }

  public Layers m_layers = new Layers();
  private boolean flLoad = false;
}
```

Код модуля Project.java

```
package gisviewer;
import java.awt.Graphics;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

abstract class GeoObjects {
  static final int POINTS = 0;
  static final int LINES = 1;
  static final int POLYGONS = 2;
  private GeoProperties m_properties = new GeoProperties();

  protected boolean loadProperties(String fileName)
  {
      Scanner fin = null;
      boolean res = false;
      try
      {
        fin = new Scanner(new File(fileName));
        m_properties.setVisible( Integer.parseInt( fin.next() ) );
        m_properties.setSize( Integer.parseInt( fin.next() ) );
        m_properties.setColor( Integer.parseInt( fin.next() ) );
        m_properties.setStyle( Integer.parseInt( fin.next() ) );
        fin.close();
        res = true;
      }
      catch(FileNotFoundException e) { }

      return res;
  }

  protected abstract boolean doLoadLayer(String fileName);

  public boolean loadLayer(String fileName)
  {
     if(doLoadLayer(fileName))
     {
         String fileNameProp = "";
         int count = fileName.length()-3;
         for(int i=0;i<count;i++) fileNameProp += fileName.charAt(i);
         fileNameProp += "prop";

         return loadProperties(fileNameProp);
     }
     else return false;
  }

  public abstract void draw(Graphics gr);

  public abstract void clear();
  public abstract void debugprint();
  public GeoProperties getProperties()  {  return m_properties; }
}
```

Код модуля GeoObjects.java

Модуль GeoPoints.java

```
package gisviewer;
import java.util.ArrayList;
import java.awt.Graphics;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

class GeoPoints extends GeoObjects {
  private ArrayList<Point> m_pts = new ArrayList<Point>();

  @Override protected boolean doLoadLayer(String fileName)
  {
       Scanner fin = null;
       boolean res = false;
      try
      {
        fin = new Scanner(new File(fileName));
        int countObj = fin.nextInt();
        for(int i=0; i<countObj; i++)
        {
            double x = Double.parseDouble( fin.next() );
            double y = Double.parseDouble( fin.next() );
            m_pts.add(new Point(x,y));
        }
        fin.close();
        res = true;
      }
      catch(FileNotFoundException e) { }
      return res;
  }

  @Override public void draw(Graphics gr) {  }

  @Override public void clear() { m_pts.clear(); }

  @Override public void debugprint()
  {
      int count = m_pts.size();
      for(int i=0;i<count;i++)
      {
        Point pt= m_pts.get(i);
        System.out.println(pt.x+"\t"+pt.y);
      }

       System.out.println("Properties:");
       System.out.println("VISIBLE = " + this.getProperties().getVisible());
       System.out.println("SIZE = " + this.getProperties().getSize());
       System.out.println("COLOR = " + this.getProperties().getColor());
       System.out.println("STYLE = " + this.getProperties().getStyle());
  }
}
```

Код модуля GeoPoints.java

```
package gisviewer;

import java.util.ArrayList;
import java.awt.Graphics;

class GeoPolygons extends GeoObjects {
  protected ArrayList<ArrayList<Point>> m_obj;

  @Override protected boolean doLoadLayer(String fileName) { return false; }

  @Override public void draw(Graphics gr) { }

  @Override public void clear()
  {
      int count = m_obj.size();
      for(int i=0;i<count;i++)
      {
          ArrayList<Point> pts = m_obj.get(i);
          pts.clear();
      }
      m_obj.clear();
  }

  @Override public void debugprint()
  {
      int count = m_obj.size();
      for(int i=0;i<count;i++)
      {
          System.out.println("Object "+ (i+1));

          ArrayList<Point> pts = m_obj.get(i);
          int count2 = pts.size();
          for(int j=0;j<count2;j++)
          {
              Point pt= pts.get(i);
              System.out.println(pt.x+"\t"+pt.y);
          }
      }
  }
}
```

Код модуля GeoPolygons.java

Навчально видання

Олевський Віктор Ісакович
Молодець Богдан Володимирович

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Методичні рекомендації до виконання лабораторних робіт
для здобувачів ступеня бакалавра
спеціальності F6 Інформаційні технології

В авторській редакції

Електронний ресурс.
Підписано до видання 18.02.2025. Авт. арк. 2,1

Національний технічний університет «Дніпровська політехніка»
49005, м. Дніпро, просп. Дмитра Яворницького, 19