ВСТУП

У поточному освітньому середовищі прогрес відбувається швидко, особливо для студентів, які є ключовим елементом процесу навчання. Щороку з’являються нові програми, курси та опції з різними функціями та призначеннями. Перед студентами постає проблема вибору найбільш відповідної освітньої програми залежно від їхніх потреб і потенціалу. Одними з головних міркувань при виборі освітньої програми є зміст курсу та репутація навчального закладу. Це дослідження важливе, оскільки потребує розробки робочої моделі, яка дозволить викладачам обробляти та класифікувати дані про студентів та їхні освітні програми відповідно до чітко визначених критеріїв. Практичний помічник, який дозволяє викладачам обробляти та впорядковувати дані відповідно до конкретних умов. Це полегшує оцінювання навчальних курсів і дозволяє швидко знайти здобувача освіти, який відповідає заданим параметрам.

Метою курсової роботи є розробка програми, яка обробляє дані про освітні програми, дозволяючи користувачам сортувати студентів за групою, ім’ям та оцінками.

У дані курсові роботі ми розберемо потрібні інструменти, методики та функціонал для реалізації поставленої нами задачі, а саме ‘Розробка програми оброблення даних з файлів та занесення даних у файли на тему: студенти’.

1 ОГЛЯД ПОТРІБНИХ КОНЦЕПЦІЙ ТА ІНСТРУМЕНТІВ ДЛЯ ВИРІШЕННЯ НАШОЇ ЗАДАЧІ ЗА ДОПОМОГОЮ ООП В МОВІ ПРОГРАМУВАННЯ JAVA

1.1 Що таке ООП. Методологія програмування

Об'єктно-орієнтоване програмування (ООП) є підходом до програмування, який організовує програмне забезпечення у вигляді взаємодіючих об'єктів. Ці об'єкти є сутностями, які можуть представляти як реальні, так і абстрактні поняття, з якими можна виконувати різні дії. ООП підкреслює важливість даних та їхньої взаємодії, що дозволяє створювати більш організовані, зрозумілі та гнучкі програми.

Основна ідея полягає в тому, щоб моделювати програму на основі реальних або уявних об'єктів, які взаємодіють між собою, подібно до того, як це відбувається у реальному світі. Це дозволяє розробникам краще структуризувати код, спрощує процес його написання та обслуговування, а також сприяє більшій наочності та логічності побудови програмних систем.

1.2 Огляд поставленої задачі курсового проєкту

Область нашого завдання включає інформацію про студентів, особливо групи та оцінки. Оскільки тема проекту «Розробка програми для обробки даних із файлів та введення даних у файли за темою «Студент», програма, що розробляється, має забезпечувати наступний функціонал:

- читання даних із текстових файлів;

- класифікація студентів за певними параметрами (середнім балом);

- розподіл учнів за категоріями відповідно до групи;

- зберегти оброблені дані в текстовий файл;

Усі ці функції необхідні, щоб мати впевненість у здійсненності завдання, яке ми виконуємо. Цю програму можна використовувати для класифікації студентів в університетських архівах.

Тепер, необхідно подумати про інструменти, які можна використовувати для виконання завдання.

1.3 Переваги використання ООП в мові Java

Об’єктно-орієнтоване програмування (ООП) у мові Java має кілька важливих переваг, які роблять його популярним підходом до розробки програмного забезпечення.

По-перше, ООП допомагає організувати майбутній код. Розробники можуть групувати дані та функції, які маніпулюють цими даними, у класи, щоб зробити свій код більш структурованим і логічним. Це робить програми, особливо великі та складні, легшими для розуміння, обслуговування та розширення.

Однією з головних переваг ООП є інкапсуляція, яка дозволяє приховати внутрішні деталі реалізації класу. Це означає, що поки ми використовуємо лише публічний інтерфейс класу, зміни в реалізації класу не вплинуть на решту вашої програми. Такий підхід зменшує ризик помилок і полегшує підтримку коду. ООП також полегшує модульність програми. Завдяки розділенню програми на окремі об’єкти або класи, кожен з яких відповідає за певну функцію, програма стає більш гнучкою та легшою для розширення.

Модульність полегшує повторне використання коду, дозволяючи легко переносити окремі модулі в інші проекти. Простота тестування та налагодження також є важливою перевагою ООП. Класи інкапсулюють дані та функції, що робить більш ефективним і зручним тестування та налагодження кожного класу окремо. Це допомагає виявити та виправити помилки на ранніх етапах розробки.

Загалом, об’єктно-орієнтоване програмування у мові Java пропонує значні переваги в розробці програмного забезпечення, оскільки робить процес більш організованим, модульним і гнучким. Це дозволяє створювати високоякісні, прості для розуміння та обслуговування програми, які легко адаптуються до змін і розширюються в майбутньому.

1.4 Поняття класу та об’єкта в мові програмування Java

Класи в програмуванні є основними будівельними блоками, які визначають структуру та поведінку об’єктів.[1] Це абстракція, яка описує загальні властивості та ознаки певного типу об’єкта. Клас визначає набір атрибутів, які зберігають стан об'єкта, і методи, які визначають поведінку об'єкта. Класи діють як шаблони або заготовлений план, з яких створюються об’єкти. Класи можуть мати різні модифікатори доступу, що визначають видимість та доступність їхніх атрибутів і методів. Наприклад, модифікатори private, protected та public у деяких мовах програмування дозволяють контролювати, які частини коду можуть отримувати доступ до певних частин класу[6].

Об’єкт — це конкретний екземпляр класу з певним станом і поведінкою, визначеними класом. Об'єкти містять певні значення для атрибутів, визначених у класі, і можуть виконувати методи, описані в класі. Об’єкти створюються на основі класу за допомогою спеціальних методів, які називаються

конструкторами, які ініціалізують об’єкт і надають йому початковий стан.

Об`єкти є основними одиницями, з якими взаємодіють у програмах,

написаних у стилі ООП.

Вони взаємодіють один з одним через методи, викликаючи їх для виконання певних дій або для отримання даних. Кожен об'єкт має свою власну копію атрибутів, що дозволяє об'єктам одного класу мати різні стани. Об'єкти підтримують принципи інкапсуляції, наслідування та поліморфізму, які є основоположними в ООП.

Класи та об’єкти, таким чином, є ключовими поняттями в об’єктноорієнтованому програмуванні, що дозволяє створювати структуровані, модульні та гнучкі програми. Класи визначають загальні властивості та поведінку, тоді як об’єкти є конкретними реалізаціями цих класів, які взаємодіють один з одним під час виконання програми[5].

1.5 Опис методів та атрибутів в Java

Атрибути в мові Java - це змінні, які описують стан об'єкта. Вони представляють дані, які характеризують об’єкт і зберігаються в його пам’яті. Атрибути можуть бути простими типами даних (такими як числа, рядки та логічні значення) або складними типами даних (такими як списки, словники та інші об’єкти).[1]

Кожен об’єкт має власний набір атрибутів, які визначають його унікальні властивості. Атрибути можуть мати різні рівні доступу, які визначають, як до них можуть отримати доступ інші частини програми.

Загальні рівні доступу включають публічний, приватний і захищений

доступ. Доступ до відкритих атрибутів можна отримати будь-де в програмі, до

приватних атрибутів можна отримати доступ лише в межах класу, який їх оголошує, а до захищених атрибутів можна отримати доступ у межах класу та його підкласів.

Методи визначають поведінку об'єктів класу. Це функції, які виконують певні дії над об’єктами або їхніми атрибутами. Методи викликаються в об’єктах класів і можуть приймати параметри та виконувати роботу. Методи можуть змінювати стан об’єкта, отримувати доступ до його атрибутів і повертати значення. Як і атрибути, методи можуть мати різні рівні доступу.

Це дозволяє контролювати, які частини вашої програми можуть викликати ці методи та отримувати доступ до цього коду. Загальнодоступні методи можна викликати з будь-якого місця програми, приватні методи можна використовувати лише в класі, який їх оголошує, а захищені методи можна використовувати в класі та його підкласах.

1.6 Інкапсуляція в ООП та застосування її в мові програмування Java

Інкапсуляція — це комбінація даних і методів обробки цих даних в одному об’єкті або класі.[2] Тобто об’єкти містять як дані, так і методи для маніпулювання ними, забезпечуючи логічну єдність і зручний доступ до функцій, які маніпулюють цими даними. Основна мета інкапсуляції — приховати деталі реалізації від користувача. Це означає, що зовнішні користувачі можуть використовувати об’єкт лише через публічний інтерфейс, не знаючи, як саме об’єкт реалізовано всередині.

Це забезпечує високий ступінь абстракції та спрощує використання класів або об’єктів. Інкапсуляція також допомагає запобігти неправильному використанню даних.

1.7 Динамічна поведінка об'єктів в ООП та Java

Поліморфізм — це важлива концепція в об’єктно-орієнтованому програмуванні (ООП), яка стосується здатності об’єкта відповідати на той самий запит або повідомлення, використовуючи власну реалізацію. Це означає, що об'єкти одного класу можуть поводитися по-різному в залежності від конкретної реалізації відповідного методу.[3]

У контексті ООП існує дві основні форми поліморфізму: перевизначення методу і параметричний поліморфізм. Перевизначення методів дозволяє підкласам надавати власні реалізації методів, успадкованих від батьківського класу.

Це означає, що методи з однаковою назвою можуть мати різні реалізації в різних класах. З іншого боку, параметричний поліморфізм дозволяє функціям і методам працювати з об’єктами різних типів, не знаючи конкретної реалізації, що робить код більш універсальним і зручним для користувача.[5]

Поліморфізм допомагає забезпечити гнучкість і розширюваність програми, а також спрощує розробку та підтримку коду. Це дозволяє розробникам створювати загальний інтерфейс, який може використовуватися різними типами об’єктів, не змінюючи сам код. Це забезпечує значні переваги при розробці програм, оскільки зменшує потребу переписувати код і сприяє багаторазовому використанню. Це також сприяє розширюваності програми, оскільки нові функції та класи можна легко інтегрувати в існуючий код без зміни базової логіки.[3]

1.8 Спадковість класів в мові програмування Java та ООП

Наслідування в об`єктно-орієнтованому програмуванні (ООП) дозволяє створювати нові класи на основі існуючих, відомих як батьківські класи. Підкласи, або нащадки, успадковують атрибути та методи своїх батьківських класів, що дозволяє їм використовувати та розширювати їх функціональність.

Основні аспекти наслідування включають успадкування атрибутів і методів, розширення функціональності, зменшення складності коду, створення ієрархії класів та реалізацію поліморфізму. Наслідування допомагає уникнути дублювання коду, сприяє реюзабельності коду, розширює можливості функціональності програм та полегшує його розуміння та підтримку.[2]

1.9 Одна з падаригм ООП - Абстракція

Абстракція в об’єктно-орієнтованому програмуванні передбачає виділення основних аспектів об’єкта чи системи та ігнорування другорядних деталей, щоб зробити його легшим для розуміння та використання. Це дозволяє створити модель, яка узагальнює основні характеристики об’єкта або процесу, що є корисним для розробки та аналізу системи. Абстракція також розділяє інтерфейс взаємодії між об’єктом і його реалізацією, полегшуючи зміну та розширення коду без впливу на користувачів.[1]

ВИСНОВОК ДО ТЕОРЕТИЧНОЇ ЧАСТИНИ

У теоретичній частині об'єктно-орієнтованого програмування (ООП) було розглянуто основні поняття цього підходу програмування. Перше, про що варто згадати, це ідея розгляду програми як набору взаємодіючих об'єктів. Це дозволяє програмному коду моделювати об’єкти та процеси реального світу, спрощуючи розробку та розуміння системи.

Далі ми розглянули чотири фундаментальні принципи ООП: інкапсуляція, успадкування, поліморфізм і абстракція.

Інкапсуляція дозволяє приховати деталі реалізації та контролювати доступ до даних і методів класу.

Спадкування дозволяє створювати ієрархії класів, у яких підкласи успадковують функціональні можливості батьківських класів, полегшуючи повторне використання коду та створюючи загальні шаблони програм.

Поліморфізм дозволяє об'єктам різних класів відповідати на той самий запит або повідомлення своїми власними реалізаціями.

Абстракція допомагає узагальнити та спростити представлення об’єктів і систем, ігноруючи деталі реалізації та зосереджуючись на їхній основній функціональності. Загалом, об’єктно-орієнтоване програмування є потужним інструментом для розробки програмного забезпечення, оскільки воно дозволяє створювати більш структурований, гнучкий і багаторазово використовуваний код. ООП покращує ясність, модульність і ефективність розробки програм.

2 АНАЛІЗ МЕТОДІВ ВИРІШЕННЯ ПРАКТИЧНОЇ ЗАДАЧІ

2.1 Вибір мови програмування

Вибір мови програмування для розробки програми, яка класифікує учнів за оцінкою або групою, є важливим моментом, оскільки це впливає на ефективність, надійність і масштабованість рішення. У цьому контексті Java є одним із найкращих варіантів з кількох важливих причин. Java відома своєю незалежною платформою завдяки використанню віртуальної машини Java (JVM).

Це означає, що код, який буде написано, може працювати на будь-якій платформі з JVM, що робить Java найкращим вибором для програм, які повинні запускатися на різних операційних системах без зміни коду. Java — це повністю об’єктно-орієнтована мова програмування, яка дозволяє легко моделювати об’єкти та процеси реального світу.

Для Student Sorter об’єктно-орієнтована природа Java дозволяє створювати класи для представлення студентів і визначення їх поведінки та взаємодії. Це зберігає код організованим і легшим для розширення. Java надає потужну стандартну бібліотеку, що містить колекції, які полегшують роботу з групами об’єктів. Наприклад, клас ArrayList дозволяє легко зберігати списки студентів і маніпулювати ними. З іншого боку, класи Collections і Arrays забезпечують вбудовані методи сортування, ефективні та прості у використанні. Java має надійну типізацію, яка допомагає виявити багато помилок під час компіляції.

Це забезпечує кращий захист від помилок і робить програму більш надійною. Під час розробки програм, які маніпулюють даними, як-от оцінювання студентів, це важливо для забезпечення точності та надійності.

Хоча Java не є найшвидшою мовою програмування порівняно з іншими мовами, продуктивність дуже висока завдяки оптимізації, яку забезпечує JVM.

Для більшості завдань, таких як сортування, продуктивності Java достатньо. Java є однією з найпоширеніших мов програмування у світі, яка пропонує обширні ресурси, документацію та підтримку спільноти.[4] Тому це хороший варіант як для початківців, так і для досвідчених програмістів, оскільки ми можемо швидко знайти рішення своїх питань.

2.2 Огляд можливих середовищ проекту

При виборі середовища розробки (IDE) для нашого проекту, важливо врахувати переваги та недоліки кожного з них. Ось огляд кількох популярних середовищ для розробки на мові Java та причини, чому Intelij IDEA найкраще підходить для нашого проекту.

1. Eclipse

Переваги:

- вільний та відкритий вихідний код;

- велика кількість плагінів та розширень;

- підтримка різних мов програмування.

Недоліки:

- менш інтуїтивний інтерфейс у порівнянні з іншими IDE;

- потребує більше часу на налаштування та оптимізацію;

- може працювати повільніше з великими проектами.

2. NetBeans

Переваги:

- вільний та відкритий вихідний код;

- вбудована підтримка різних мов програмування, включаючи Java;

- хороша інтеграція з серверами баз даних.

Недоліки:

- обмежена кількість плагінів у порівнянні з Eclipse;

- менш розвинені функції для рефакторингу коду;

- менш активна спільнота розробників.

3. IntelliJ IDEA

Переваги:

- інтуїтивний та зручний інтерфейс;

- потужні інструменти для рефакторингу коду;

- інтелектуальні підказки коду та автоматичне завершення;

- інтеграція з системами контролю версій (Git, SVN);

- підтримка великої кількості плагінів та розширень;

- висока продуктивність та швидкість роботи навіть з великими проектами.

Недоліки:

- платна версія (хоча є безкоштовна версія Community Edition);

- більш високі системні вимоги у порівнянні з іншими IDE.

2.2.1 IntelliJ IDEA найкраще підходить для проекту

Інтелектуальні підказки та автоматичне завершення коду в IntelliJ IDEA значно підвищують продуктивність розробників. Потужні інструменти для рефакторингу коду дозволяють швидко і легко вносити зміни, що знижує ризик помилок і підвищує якість коду. Інтеграція з системами контролю версій дозволяє ефективно співпрацювати в команді, а велика кількість плагінів та розширень дозволяє налаштувати IDE відповідно до потреб конкретного проекту.[4]

Таким чином, IntelliJ IDEA забезпечує найкращий баланс між функціональністю, зручністю використання та продуктивністю, що робить її оптимальним вибором для нашого проекту.

2.3 Порівняння реалізованих класів

Для реалізації поставленої нами задачі, тобто «Розробка програми для обробки даних з файлу по темі «Студенти» і введення даних у файл», нам необхідно використовувати переваги об'єктно-орієнтованого програмування одним із яких є – клас.

Клас в об’єктно-орієнтованому програмуванні представляє абстрактний шаблон об’єктів, які мають спільні характеристики та поведінку. Основні класи, необхідні нам для створення програми:

- Learner: клас для виклику нашої програми;

- Controller: клас який містить методи обробки для програми;

- Student:клас який містить моделі студентів і всі їх параметри.

2.4 Модель Класу Student

Для реалізації нашого завдання нам потрібен модельний клас для учнів, де потрібно створити параметри групи студента, ім'я та оцінку. Реалізація класу описана в третьому розділі. Основними атрибутами класу моделі є геттери та сетери для змінних і метод конструктора для подальшого створення моделі з параметрами.

2.5 Клас ‘Controller’

Клас Controller діє як контролер, який керує логікою програми та взаємодіє з іншими компонентами системи. У нашому коді ми відповідатимемо за обробку файлу, розподіл і класифікацію студентів за середньою оцінкою, сортування студентів за певними категоріями та запис даних у файл.

Контролер приймає файл, обробляє його, виконує необхідні операції над студентом і зберігає результати у відповідному файлі.

2.6 Клас ‘Learner’

Клас Learner відповідає за створення необхідних директорій, ініціалізацію файлів та виклик методів для обробки файлів.

Опис класу:

- створення директорій: У класі Learner створюються дві директорії: одна для вхідних файлів (InputFiles), інша для вихідних файлів (OutputFiles).

Це дозволяє організувати файли у відповідних папках, що полегшує роботу з ними;

- ініціалізація файлів: У даному прикладі ініціалізуються три вхідні файли: file1in.txt, file2in.txt та file3in.txt. Ці файли зберігаються у директорії InputFiles;

- виклик методу для обробки файлів: Метод handleFiles класу Controller викликається з параметрами директорії для вихідних файлів та масиву вхідних файлів. Цей метод відповідає за обробку файлів, сортування даних та запис результатів у вихідні файли.

- клас Learner є стартовою точкою для нашої програми. Він відповідає за налаштування необхідних директорій та файлів, а також за виклик методів, які забезпечують основну функціональність програми.

2.7 Актуальність бібліотек в середовищі Intelij IDEA

В середовищі розробки Intelij IDEA актуальність бібліотек відіграє критичну роль у забезпеченні оптимальної продуктивності, безпеки, функціональності та стійкості програм Java. Застарілі бібліотеки, як правило, несуть у собі ризик зниження продуктивності, виникнення вразливостей, обмеження функціональних можливостей та нестабільності програмного забезпечення.

Тому регулярне оновлення бібліотек до найновіших версій є невід'ємною частиною успішного розробника Java.

Intelij IDEA надає розробникам зручні інструменти для ефективного керування бібліотеками. До них належать панель Maven Projects, що відображає список використовуваних бібліотек та їх версії, а також діалогове вікно Update Dependencies, яке дозволяє легко оновлювати бібліотеки вручну або налаштовувати автоматичне оновлення.

Важливо зазначити, що не всі бібліотеки потребують негайного оновлення до найновішої версії. Деякі стабільні та добре зарекомендовані бібліотеки можуть безперебійно функціонувати протягом тривалого часу. Перед оновленням будь-якої бібліотеки необхідно ретельно оцінити потенційні ризики та переваги, щоб уникнути проблем з сумісністю або інших непередбачених наслідків.

2.8 Вимоги до курсового ПЗ

Розроблена програма є інструментом для обробки текстових файлів, що містять інформацію про студентів. Вона виконує наступні функції:

- читання даних про студентів: Програма здатна читати текстові файли з інформацією про студентів. Формат файлів повинен бути чітко визначений (група, ім'я, оцінка, розділені пробілами);

- обробка даних: Програма обробляє дані про студентів, зберігаючи їх у внутрішніх структурах;

- валідація даних: Програма перевіряє коректність формату вхідних даних. У разі виявлення помилок, програма повинна повідомити користувача;

- розподіл студентів за успішністю: На основі оцінок, програма розподіляє студентів за трьома категоріями: високий бал, середній бал, низький бал. Межі цих категорій задані в 3-ому розділі;

- сортування студентів: Програма дозволяє сортувати списки студентів за різними критеріями (групою та оцінкою). Критерії сортування повинні бути визначені користувачем;

- збереження результатів: Програма зберігає відсортовані списки студентів у окремих текстових файлах. Назви файлів повинні бути зрозумілими та відображати категорію успішності студентів.

2.8.1 Технічні вимоги

Програма повинна мати можливість працювати в середовищі, яке підтримує:

- об'єктно-орієнтоване програмування;

- стандартні бібліотеки для роботи з файлами та текстом;

- структури даних для зберігання та обробки інформації про студентів;

- механізми сортування даних;

- взаємодію з користувачем для отримання налаштувань.

2.8.2 Додаткові вимоги

- програма повинна бути розроблена з урахуванням зрозумілості та простоти використання;

- повинна бути передбачена обробка помилок та винятків під час виконання програми;

- забезпечити документування коду для покращення підтримки та модифікації програми.

ВИСНОВОК ДО АНАЛІТИЧНОЇ ЧАСТИНИ

В аналітичній частині роботи було детально розглянуто ключові аспекти розробки програми для сортування студентів за оцінками та прізвищами. Було проаналізовано вимоги до програми, визначено основні етапи розробки та обґрунтовано вибір мови програмування Java для реалізації цього проєкту.

Збір та зберігання даних про студентів, розробка алгоритмів сортування та створення зручного інтерфейсу користувача є важливими компонентами цієї задачі. Вибір Java виявився оптимальним завдяки її платформній незалежності, об'єктно-орієнтованій природі, потужним стандартним бібліотекам, сильної типізації, високій продуктивності, широкій підтримці спільноти та інтеграції з провідними інструментами розробки, такими як IntelliJ IDEA.

Аналіз вимог до програми показав, що коректність даних, ефективність сортування, масштабованість та зручність використання є ключовими факторами, що впливають на успішність розробки та впровадження даного рішення.

3 ПРАКТИЧНЕ СТВОРЕННЯ ПРОГРАМИ

3.1 Середовище розробки

Коли було почато розробку програми Java, було обрано середовище IntelliJ IDEA.

Таке рішення було прийнято з кількох причин:

По-перше, IntelliJ IDEA відома своєю потужністю та розширюваністю. Було швидко виявлено, що це ідеальний інструмент для завдань на всіх етапах розробки. Різні функції, такі як автоматичне завершення коду, вбудований контроль версій і підтримка мови Java, тепер допомагають мені вирішувати завдання ефективно та без особливих зусиль.

По-друге, IntelliJ IDEA надає потужні інструменти візуального налагодження, аналізу коду та рефакторингу, які допомагають покращити якість коду та скоротити час розробки.

Як користувач із досвідом використання інших середовищ розробки, я також знайшов інтерфейс IntelliJ IDEA інтуїтивно зрозумілим і простим у використанні. Його гнучкість і налаштування дозволили мені адаптувати робоче середовище до моїх потреб і стилю розвитку.

Підсумовуючи, вибір IntelliJ IDEA для розробки програм на мові Java виявився вдалим. Його потужність, зручність і масштабованість дозволили ефективно вирішувати проблеми та досягати бажаних результатів.

3.2 Створення проекту

Після відкриття застосунку створюємо новий проект під назвою “CourseProject” і відкриваємо файл main.java.

3.3 Код реалізації

Після успішного написання програмного коду, ми розмістили його в додатку для зручності та подальшого використання. Усі функції та методи описані в коді були ретельно протестовані та перевірені на відповідність вимогам проекту. Для полегшення навігації та розуміння коду, також було додано коментарі та пояснення до ключових частин. Див. Додаток А.

3.4 Наповнення файлів

Після написання коду створюємо три текстові файли і наповнюємо їх наступним чином. Приклад того як наповнюються файли зображені на рисунку 3.1, рисунку 3.2 та рисунку 3.3.

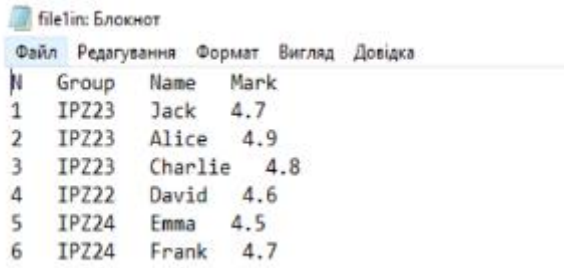

Рисунок 3.1 – Вміст першого файлу “file1in”

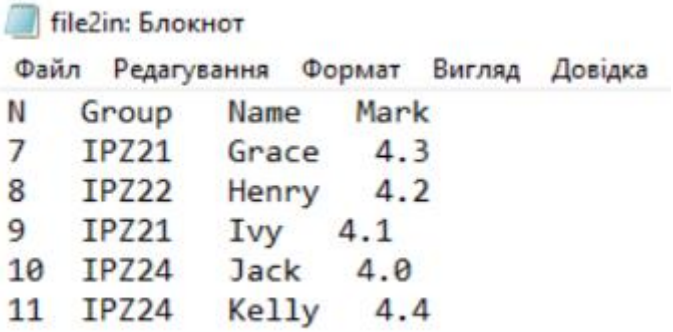

Рисунок 3.2 – Вміст другого файлу “file2in”

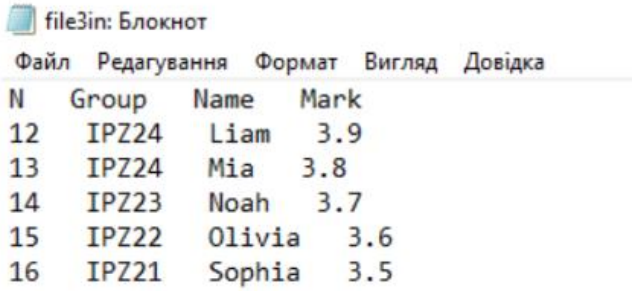

Рисунок 3.3 – Вміст третього файлу “file3in”

Всі збіги з дійсними людьми випадкові, дані взяті випадковим чином для реалізації експерименту.

3.5 Результат програми

Програма має в своєму функціоналі два режими, після натискання цифри 1 ми сортуємо студентів по оцінках, якщо 2 то по групі та прізвищу.

Результат роботи першого режиму зображено на рисунку 3.4.

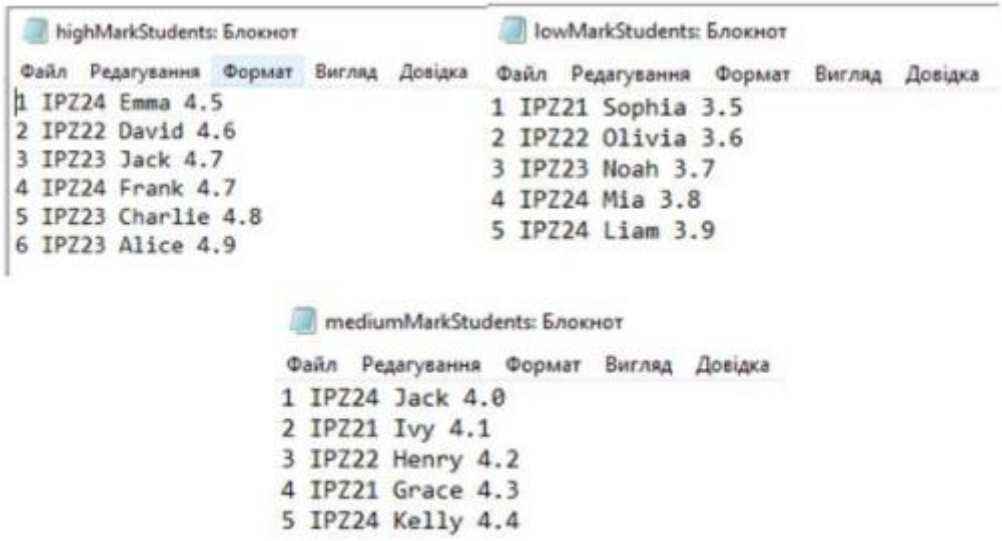

Рисунок 3.4 – Результат роботи сортування студентів за оцінками

Результат роботи другого режиму зображено на рисунку 3.5.

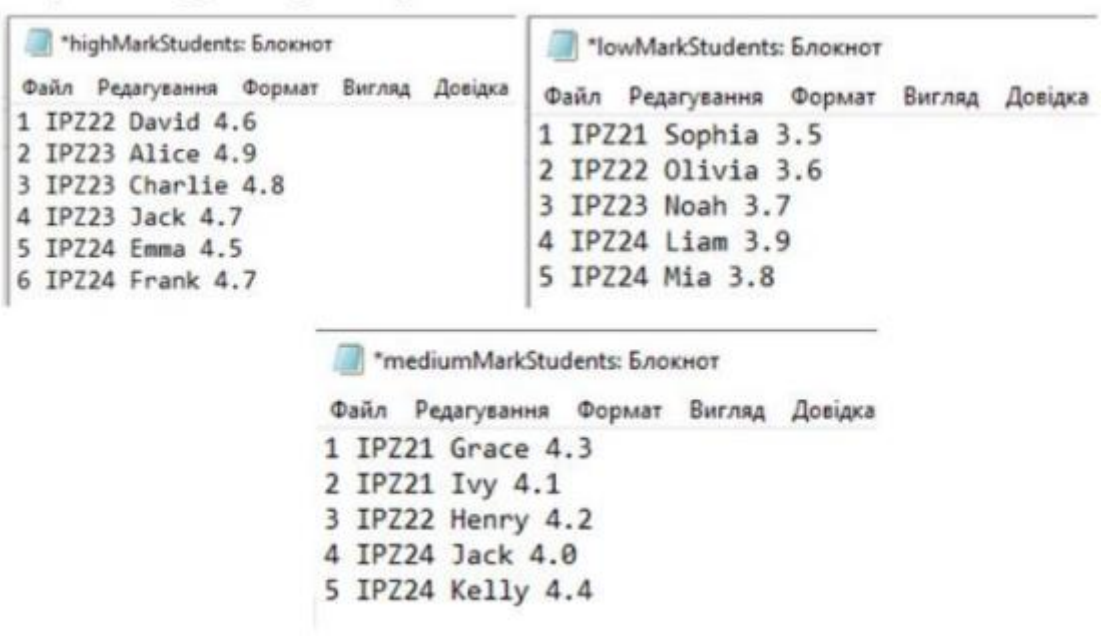

Рисунок 3.5 – Результат роботи сортування студентів по групах

ВИСНОВКИ

Результатом курсового проекту стала успішна розробка програми класифікації студентів за оцінкою та групою, яка була реалізована мовою програмування Java в середовищі розробки IntelliJ IDEA.

Під час розробки були завершені всі основні етапи проектування, включаючи аналіз вимог, вибір технології, реалізацію алгоритму та тестування програмного забезпечення.

Ця програма дозволяє ефективно та надійно класифікувати студентів за визначеними критеріями, таким чином забезпечуючи точність та оптимальну продуктивність навіть із великими обсягами даних. Вибір Java як мови програмування виправданий рядом переваг, серед яких: Незалежність від платформи, об’єктно-орієнтована природа, потужна стандартна бібліотека, надійна типізація та висока продуктивність.

Крім того, використання IntelliJ IDEA як середовища розробки значно підвищило ефективність процесу розробки завдяки інструментам автозаповнення коду, рефакторингу, інтеграції з системами контролю версій та іншим корисним функціям.