Крос-платформне програмування

ЛАБОРАТОРНА РОБОТА № 4

Використання файлової системи.

Елементи ООП. Серіалізація об’єктів.

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторної роботи
.

Львів

2024

Мета роботи: Ознайомлення з реалізацією принципів об’єктно-орієнтованого програмування в Java та Kotlin та засобами зберігання структурованих даних у фаловій системі. Формування елементарних навиків використання файлової системи ПК для зберігання структурованих данихпри розробці невеликих інформаційних систем.

Завдання до лабораторної роботи.

1. Описати основні можливості функціоналу запропонованої у варіанті завдання простої ***інформаційної системи*** та словесно сформулювати ТЗ (в довільній формі) на її розробку. Система повинна забезпечувати основні можливості (CRUD-операції) з опрацювання вказаної у варіанті інформації та зберігання даних у ***файлі*** у вигляді серіалізованої колекції об’єктів. А також надавати користувачеві можливість ***фільтрації та пошуку*** даних за вказаними у варіанті завдання критеріями. Взаємодію ІС з користувачем реалізувати в командному (текстовому) режимі у вигляді ***консольного*** додатку.
2. Реалізувати запланований функціонал з використанням мови ***Java***.
3. Реалізувати запланований функціонал з використанням мовою ***Kotlin***.
4. Оформити письмовий звіт про виконання роботи.

Звіт повинен містити:

- титульний аркуш:
- умову задачі відповідного варіанту;
- програму на Java;
- результати виконання кожного з елементів реалізованого функціоналу системи;
- програму на Kotlin;
- результати виконання кожного з елементів реалізованого функціоналу системи;

Короткі теоретичні відомості

В сучасних проєктах при розробці модуля для зберігання даних загальноприйнятою технікою є дотримання патерну проєктування ***Репозиторій***. Не претендуючи на точність та повноту реалізації цього шаблону в рамках нашої задачі, спробуємо зберегти основні його властивості в коді прикладів.

Терміном ***файл***, зазвичай, позначають іменовану послідовність байтів збережену в запам'ятовуючому пристрої. Проте в програмуванні часто розглядають ***файл*** як структуру даних у вигляді послідовності, у якій дані записуються і зчитуються послідовно від початку до кінця. ***Файли*** можуть зберігатися в запам'ятовуючих пристроях, або використовуватися для передачі даних між зовнішніми пристроями та програмними компонентами. Хоча в сучасних технологіях програмування в такому контексті більш вживаним є термін ***потік***. ***Потоки*** забезпечують передачу даних між програмою та пристоями вводу-виводу, фізичними файлами на диску, за допомогою мережевих з'єднань і т. п.

За способом зберігання даних, файли прийнято ділити на ***бінарні*** та ***текстові***. У першому випадку дані у файлі зберігаються у вигляді послідовності байтів, в другому, – як послідовність символів. Фізично, з точки зору файлової системи тут немає великої відмінності, бо символи також кодуються в пам'яті ЕОМ у вигляді одного чи декількох байтів даних. Але для такого подання символів розроблено багато різних способів кодування і опрацювання сивольних даних вимагає принаймні використання одного і того ж механізму кодування при записуванняі та читанні символів. Не варта також думати, що текстові файли придатні лише для зберігання текстової інформації, – тут достатньо згадати такі текстові формати представлення та зберігання даних, як ***xml***, ***json*** чи ***csv***. Такі формати зручні, оскільки дозволяють читати дані людині, чи опрацьовувати їх за допомогою різного ПЗ.

Щодо ***бінарних*** файлів, то тут також необхідний механізм інтерпретації збереженої послідовності байтів. Зазвичай, тут використовується певний фіксований набір змінних, тобто, шаблон, за допомогою якого структуруються послідовність байтів у файлах. В загальному випадку, його деколи називають ***комбінованим типом***. В алголоподібних мовах, вт. ч. Pascal, до нього використовують термін ***запис***, в C та C++ – ***структура***. В об'єктно-орієнтованих технологіяг у ролі таких шаблонів використовуються ***класи***, а окремі порції даних для зберігання представлені ***об'єктами***. Сам же процес зберігання (зчитування) байтів даних і перетворення їх в інформацію заданої структури тут називається ***серіалізацією*** (***десеріалізацією***) ***об'єктів***.

Об'єктно-орієнтовані технології програмування в якості основного засобу для створення комбінованих типів даних використовують ***класи***. ***Клас*** в ООП об'єднує набору різнотипних даних, класи підтримують ***спадкування***, ***поліморфізм***, ***узагальнення*** і інші об'єктно-орієнтовані механізми. Використання об'єктів класу в констектсі нашої задачі для елементарного зберігання комбінованих даних потребує зберігання бітової копії об'єкта з можливістю його подальшого відновлення шляхом ***серіалізації*** даних об'єкта в

файл. ***Серіалізація***, зазвичай виконується методами потокових класів, зв'язаних з файлами, або за допомогою спеціальних класів-серіалізаторів.

Об'єкти найпростіших класів, що містять лише поля з елементарними даними завжди придатні ***серіалізації,*** але, взагалі кажучи, не всі об'єкти можна серіалізувати. Зворотній процес, тобто, створення об'єкта в програмі на основі збереженої бітової копії, називають ***десеріалізацією***. В кросплатформних технологіях, таких як ***Java*** чи ***.Net (C#), серіалізація*** є безальтернативним способом зберігання комбінованих даних ще й тому, що програма не має прямого доступу до пам'яті, а створенням і розміщенням даних в пам'яті ПК тут займається віртуальна машина.

В коді на ***Kotlin*** також можна дотримуватися парадигми ООП. Об'єктно-орієнтована підхід тут виглядає доволі «чудернацько» у порівнянні з класикою ООП, в таких мовах як ***Java*** чи ***C#.*** Насправді, такі кардиальні відмінності Kotlin-ООП від Java-ООП мають на меті зробити код програм макимально простим. Основні правила об'єктно-орієнтованої парадигми у ***Kotlin***:

- ✓ *оголошення класу складається з імені класу, заголовка класу та тіла класу, оточеного фігурними дужками;*
- ✓ *як заголовок, так і тіло є необов'язковими; якщо клас не має тіла, фігурні дужки можна опустити;*
- ✓ *клас у Kotlin може мати первинний конструктор та один або декілька вторинних конструкторів;*
- ✓ *первинний конструктор є частиною заголовка класу, він йде після імені класу та необов'язкових параметрів типу;*
- ✓ *параметри первинного конструктора автоматично перетворюються в поля класу;*
- ✓ *якщо первинний конструктор не має анотацій або модифікаторів, ключове слово constructor можна опустити;*
- ✓ *первинний конструктор не може містити коду, код ініціалізації можна розмістити у блоках ініціалізаторів виду* **`init{…}`***;*
- ✓ *під час ініціалізації екземпляру блоки ініціалізаторів виконуються у тому ж порядку, в якому вони з'являються у тілі класу;*
- ✓ *Kotlin має синтаксис для оголошення властивостей та їх ініціалізації з первинного конструктора;*
- ✓ *в класі також можна оголошувати вторинні конструктори, вони повинні мати префікс* **`constructor`***;*
- ✓ *якщо клас має первинний конструктор, то вторинний конструктор повинен звертатися до первинного конструктора з допомогою ключового слова* **`this`***;*
- ✓ *якщо неабстрактний клас не має жодного конструктора компілятор генерує загальнодоступний первинний конструктор без параметрів;*
- ✓ *якщо всі параметри первинного конструктора мають значення за замовчуванням, компілятор генерує додатковий конструктор без параметрів, що використовує значення за замовчуванням;*

- ✓ *у Kotlin визначено чотири модифікатори видимості:* **`private`**, **`protected`**, **`internal`** *та* **`public`** *(за замовчуванням, тобто, якщо модифікатор відсутній);*
- ✓ *щоб створити екземпляр класу, викликають конструктор, як звичайну функцію, в Kotlin немає ключового слова* **`new`***;*
- ✓ *клас може бути оголошено абстрактним разом з деякими або всіма його членами, абстрактний член не має реалізації у своєму класі, екземпляри абстрактного класу створювати не можна;*
- ✓ *усі класи у Kotlin мають спільний суперклас* **`Any`***, який є суперкласом за замовчуванням для класів без оголошених супертипів;*
- ✓ *Any має три методи:* **`equals()`**, **`hashCode()`** *та* **`toString()`***, ці методи визначені для всіх класів Kotlin;*
- ✓ *за замовчуванням класи Kotlin не можна успадковувати, щоб дозволити спадкування слід позначити базовий клас ключовим словом* **`open;`**
- ✓ *в Kotlin можна використовувати інтерфейси, вони можуть містити оголошення абстрактних методів, а також реалізації методів, інтерфейс визначається за допомогою ключового слова interface;*
- ✓ *імплементація інтерфейсу оголошується черз двокрапку, так як і спадкування класів, але клас може імплементувати декілька інтерфейсів.*

Завдання.

Розробити просту інформаційну систему, яка надасть користувачеві основні можливості з опрацювання вказаної у варіанті інформації та зберігання даних у файлі у вигляді серіалізованої колекції об'єктів та можливість фільтрації та пошуку даних за вказаними у варіанті завдання критеріями. Взаємодію ІС з користувачем реалізувати в командному (текстовому) режимі у вигляді консольного додатку.

Варіанти завдань.

1. Створити програму для опрацювання файлу з даними про авіарейси. Структура даних: номер рейсу, пункт призначення, час відправлення, ціна квитка, кількість вільних місць. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід усіх даних з записаного раніше файлу. Реалізувати вивід у вигляді таблиці інформації про всі рейси до заданого користувачем пункту призначення та усі рейси, що відправляються у заданий користувачем проміжок часу.

2. Написати програму для опрацювання файлу з інформацією про складання екзаменаційної сесії студентами. Структура даних: шифр групи, прізвище студента, назва предмета, оцінка, назва предмета... (всього 5 предметів). В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід не екран усіх даних з записаного раніше файлу. Запрограмувати вивід списку студентів, які отримали "5" з усіх предметів, а також рейтингу (середньої оцінки з усіх предметів) в упорядкованому за спаданням вигляді.

3. У файл записують дані про продаж комп'ютерної техніки наступної структури: код покупця, назва товару, ціна за одиницю товару, продана кількість. Написати програму яка дозволить користувачеві створювати файл з даними, вносити в нього дані, а також виводити раніше збережені у файлі дані на екран (у вигляді таблиці). Розробити функціонал, який дозволить користувачеві за введеним з клавіатури кодом покупця виводити список куплених ним товарів та загальну вартість покупок, а за введеною назвою товару, – коди покупців, що його придбали та кількість проданих одиниць.

4. Створити програму для роботи з файлом, що містить дані про рух пасажирських потягів від деякої станції. Запис містить поля: назва кінцевої станції, номер потяга, час відправлення, кількість наявних місць по категоріях вагонів – СВ, купейний, плацкартний, загальний. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід не екран усіх даних з записаного раніше файлу. Програма повинна також виводити список потягів за вказаною назвою кінцевої станції та проміжком часу відправлення, а також відбирати лише ті з них, які мають вільні місця вказаної категорії.

5. У файл записують дані про продаж турів клієнтам туристичної агенції наступної структури: прізвище та ім'я клієнта, назва туру та його тривалість, вартість туру та відмітка про оплату. Написати програму яка дозволить

користувачеві створювати файл з даними, вносити в нього дані, а також виводити раніше збережені у файлі дані на екран (у вигляді таблиці). Розробити функціонал, що дозволить користувачеві за введеною з клавіатури назвою туру знайти усіх клієнтів, що придбали цей тур та порахувати їх кількість та загальну суму виручки від продажі турів, а також вивести список усіх клієнтів, що замовили вказаний тур, але не внесли кошти на його оплату.

6. Створити програму для опрацювання файлу з даними про оплату комунальних послуг через платіжну систему банку. Структура даних: прізвище платника, адреса платника, вид послуги, дата оплати, сума. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід усіх даних з записаного раніше файлу. В програмі передбачити вивід даних про всі оплати послуг за вказаною адресою а також вивід підсумкової таблиці з сумарною оплатою кожного виду послуг.

7. Створити програму для опрацювання файлу з інформацією про запис на прийом протягом поточного тижня відвідувачів поліклініки. Структура даних: прізвище відвідувача, домашня адреса, рік народження, прізвище сімейного лікаря, день тижня, потрібна спеціалізація лікаря. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід у формі таблиці усіх даних з записаного раніше файлу. Реалізувати також вивід списку відвідувачів та підрахунок їх кількості для кожної спеціалізації на вказаний день тижня, а також вивід списку відвідувачів та їх кількості на кожен день тижня для заданої спеціалізації лікаря.

8. У файл записують інформацію про вакансії, подані ІТ-компаніями, яка містить наступні дані: назва компанії, назва позиції, мова/технологія програмування, вимоги до претендента, орієнтовна заробітна плата. Розробити програму для створення файла з даними, внесення в нього даних, а також виводу усіх даних з файла на екран (у вигляді таблиці). Розробити функціонал, який дозволить користувачеві відбирати за файла вакансії за введеною з клавіатури назвою мови програмування та діапазоном для розміру орієнтовної зарплатні, а також переглядати список усіх вакансій, поданих вказаною компанією.

9. Ріелтор зберігає у файлі інформацію з оголошень про продаж нерухомості у вигляді записів такої структури: адреса помешкання, кількість кімнат, житлова площа, загальна площа, поверх, ціна, телефон власника. Реалізувати програму, яка дозволить ріелторові створювати новий файл з даними, вносити в нього дані, а також виводити раніше збережені у файлі дані на екран. Передбачити для користувача можливість шукати у файлі та виводити на екран інформацію за такими параметрами: за введеною з клавіатури кількістю кімнат та діапазоном цін, або за кількістю кімнат та орієнтовною загальною чи житловою площі помешкання.

10. Створити програму для опрацювання файлу з даними про наявність книг у книжковому магазині. Запис містить поля: назва видавництва, назва книги, прізвище автора, рік видання, роздрібна ціна, кількість примірників на складі. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід усіх даних з записаного раніше файлу. Запрограмувати пошук книг за

прізвищем автора чи назвою (або її частиною), а також вивід списку усіх книг, які вийшли у вказаному видавництві.

11. Рекрутер записує у файлі інформацію щодо резюме, розміщених IT-фахівцями у сервісах пошуку роботи. Записи містять інформацію такої структури: прізвище та ім’я, електронна адреса, місто проживання, перелік технологій (текст, в якому перераховані технології відділяються комами), досвід роботи, бажання працювати віддалено, можливість зміни міста проживання. Реалізувати програму для створення файла з даними, внесення в нього даних, а також виводу усієї інформації з файла на екран. Забезпечити можливість відбору для офісу із заданого міста резюме, що відповідають вказаній технології, розділивши їх на три частини: претенденти, що проживають у заданому місті, претенденти, готові до віддаленої роботи та претенденти, що погодяться переїхати у вказане місто.

12. Менеджер паркінгу реєструє у файлі інформацію про парковку авто та оплату послуг їх власниками. Записи містять поля: марка та модель авто, державний номерний знак, телефон власника, година парковки (ціле число) та кількість годин, за яку вноситься передоплата. Скласти програму для вводу даних та збереження їх у файл, а також для виводу на екран усіх даних із записаного раніше файлу. Реалізувати функціонал, що дозволить менеджеру, вказавши поточний час (годину) дізнатися список авто, власникам яких доведеться доплатити за парковку та за скільки годин, а також дізнатися, чи авто може без доплати покинути паркінг, вказавши його номер.

13. Створити програму для опрацювання файлу з даними про міжміські автобусні рейси. Запис містить поля: назва міста призначення, час відправлення, вартість квитка, загальна кількість місць в автобусі, кількість проданих квитків. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід усіх даних з записаного раніше файлу. Запрограмувати визначення заповненості (у %) рейсів до вказаного міста чи обчислення сумарної виручки для кожного рейсу (за вибором користувача програми) та вивід результатів у вигляді таблиці.

14. Менеджер онлайн-магазину зберігає у файлі дані про замовлення, що надходять. Структура даних: повна назва товару, ціна, кількість, прізвище та ім’я замовника, населений пункт для доставки, спосіб доставки, здійснення оплати. Написати програму для автоматизації роботи менеджера. В програмі передбачити ввід даних менеджером та збереження їх у файлі, а також вивід усіх даних з записаного раніше файлу. Запрограмувати вивід з файлу усіх оплачених замовлень, які слід доставити у зазначене місто, та вивід усіх неоплачених замовлень за заданим способом доставки товару.

15. Створити програму для опрацювання файлу з даними про наявність книг у книжковому фонді бібліотеки. Запис містить поля: шифр книги, назва, прізвище автора, рік видання, кількість примірників. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід усіх даних з записаного раніше файлу. Запрограмувати вивід списку усіх книг, які є в

бібліотеці в єдиному примірнику, а також пошук книг за прізвищем автора чи назвою (або її частиною).

16. Дані про доступні для перегляду на онлайн-сервісі фільми зберігаються у файлі у вигляді записів такої структури: назва фільму, жанр, рік виходу на екрани, назва студії, прізвище режисера, загальна тривалість та кількість серій. Розробити програму для створення файлів з даними про фільми, запису даних у файл та для виводу збережених у файлі даних на екран та реалізувати пошук фільмів у файлі. Програма повинна за вказаним жанром виводити на екран два списки окремо одно-, а окремо багатосерійні фільми цього жанру, а також за вказаною назвою кінокомпанії та діапазоном років виводити інформацію про фільми, зняті компанією у цей період.

17. Файл містить дані про дні народження рідних та друзів. Кожен запис містить поля: ім’я, прізвище, номер мобільного, число, місяць та рік народження. Створити програму для роботи з файлом, що дозволить записувати у нього дані а також виводити збережені у файлі дані на екран. Програма повинна шукати дані про дату народження за вказаними прізвищем та ім’ям, а також виводити список даних про дні народження у вказаному місяці.

18. Файл містить дані про виконання робіт працівниками компанії протягом деякого періоду часу. Кожен запис містить поля: відділ, прізвище та ініціали, посада (позиція), кваліфікація, кількість відпрацьованих годин, оплата за годину. Створити програму для роботи з файлом, що дозволить записувати у нього дані а також виводити збережені у файлі дані на екран. Розробити функціонал для формування відомостей для оплати праці на основі даних з файла. Додатково реалізувати пошук у файлі даних за вказаними посадою та кваліфікацією працівника.

19. Файл містить дані про курси валют комерційних банків. Кожен запис містить поля: назва банку, назва валюти, курс купівлі, курс продажу. Створити програму для роботи з файлом, що дозволить записувати у нього дані, а також виводити збережені у файлі дані на екран. Запрограмувати можливість для користувача формувати на екрані таблицю курсів валют для вказаного ним банку, а також шукати банк з найвищим курсом покупки та банк з найнижчим курсом продажу вказаної користувачем валюти.

20. Створити програму для опрацювання файлу з даними про тематичні ресурси мережі Internet (сайти). Запис містить поля: назва сайту, URL головної сторінки, тематика сайту, короткий опис, приблизна кількість сторінок на сайті, середня кількість відвідувачів за добу, рік заснування. В програмі передбачити ввід даних та збереження їх у файл, а також вивід усіх даних з записаного раніше файлу. Запрограмувати вивід списку сайтів заданої тематики та пошук і вивід даних про сайти, опис яких містить заданий користувачем текст.

21. У файл записують дані про продаж побутової техніки наступної структури: назва товару, виробник, модель, ціна за одиницю товару, кількість одиниць на складі. Розробити програму, яка дозволить користувачеві створювати файл з даними, вносити в нього дані, а також виводити раніше збережені у файлі

дані на екран (у вигляді таблиці). Реалізувати функціонал, що дозволить менеджерові за введеною з клавіатури назвою товару виводити список пропозицій з вказанням виробників, моделей та ціни, а також шукати у файлі товари, кількість яких на складі менша за вказане число.

22. Ріелтор зберігає у файлі інформацію з оголошень про оренду нерухомості у вигляді записів такої структури: адреса помешкання, кількість кімнат, житлова площа, поверх, вартість оренди за місяць, чи включена у вартість оренди оплата комунальних послуг, телефон власника. Розробити програму, яка дозволить ріелторові створювати новий файл з даними, вносити в нього дані, а також виводити раніше збережені у файлі дані на екран. Забезпечити користувачеві можливість за введеною з клавіатури кількістю кімнат та вартістю оренди (діапазон від-до) вивести на екран два списки: помешкання з заданими параметрами, у вартість оренди яких включено комунальні послуги, та помешкання, в яких комунальні платежі доведеться сплачувати додатково.

23. Страхова компанія записує у файл дані про продані страхові поліси. Дані мають таку структуру: прізвище та ім’я клієнта, адреса, номер мобільного телефона, вид страхування, вартість полісу, дата початку страхового періоду, термін дії полісу. Розробити програму, яка дозволить менеджерам компанії створювати файли з даними, вносити в них дані, а також виводити раніше збережені у файлі дані на екран (у вигляді таблиці). Окрім цього програма повинна виводити підсумкову таблицю загальної вартості полісів за різними видами страхування, а також знаходити усі поліси клієнта за його прізвищем чи номером телефону.

24. Створити програму для опрацювання файлу з інформацією реєстрацію відвідин, протягом поточного тижня, клієнтів салону краси. Структура даних: прізвище відвідувача, домашня адреса, рік народження, день тижня, вид послуги, наявність абонементу. В програмі передбачити ввід даних користувачем та збереження їх у файлі, а також вивід у формі таблиці усіх даних з записаного раніше файлу. Реалізувати також вивід списку відвідувачів та підрахунок їх кількості для кожного виду послуги на вказаний день тижня, а також вивід для вказаного виду послуги списку відвідувачів та їх кількості на кожен день тижня.

25. Створити програму для опрацювання файлу з даними для оплати електропостачання мешканцями міста. Структура даних: прізвище платника, номер лічильника, № квартири, № будинку, назва вулиці, попереднє значення лічильника, поточне значення лічильника, вартість 1 кВт спожитої електроенергії. В програмі передбачити ввід даних, збереження їх у файл, вивід усіх даних з записаного раніше файлу, а також вивід за вказаним прізвищем платника, або його адресою (вулиця, будинок, квартира) а також формування «платіжок» для мешканців заданого будинку чи вулиці.

26. У файлі зберігають інформацію про тематичні публікації в мережі Internet. Кожен запис містить: назву сайту, URL публікації, назву публікації, прізвище та ім’я автора, дату публікації, тематику, короткий опис(анотацію). Написати програму для формування файлу з даними. В програмі передбачити ввід даних та збереження їх у файл, вивід усіх даних з записаного раніше файлу,

а також вивід переліку усіх публікацій вказаного автора, а також пошуку публікацій, опис яких містить заданий користувачем текст.

27. У файл записують інформацію про вакансії, подані різними компаніями у сервіси пошуку роботи. Інформація містить наступні дані: назва компанії, професія, посада, вид зайнятості (постійна робота, тимчасова, часткова зайнятість), вимоги до претендента, пропонована зарплатня. Розробити програму для створення файла з даними, внесення в нього даних, а також виводу усіх даних з файла на екран (у вигляді таблиці). Реалізувати функціонал, що дозволить користувачеві відбирати з файла вакансії за введеною з клавіатури назвою посади та мінімальним значенням орієнтовної зарплатні, а також переглядати список усіх вакансій за вказаним видом зайнятості.

28. У файлі зберігають дані про виконання робіт підрядниками компанії. Кожен запис містить поля: назва компанії-підрядника, юридична адреса, номер банківського рахунку, вид робіт, обсяг виконаної роботи, вартість робіт за одиницю вимірювання. Створити програму для роботи з файлом, що дозволить записувати у нього дані, виводити збережені у файлі дані на екран, а також формувати підсумковий звіт для фінансових розрахунків з підрядниками. В програмі передбачити також пошук відомостей виконавця вказаного виду робіт.

29. Створити програму для роботи з файлом, що містить дані про прокат автомобілів. Запис містить поля: марка авто, модель авто, тип кузова, коробка передач, кількість місць, об’єм багажника, об’єм двигуна, рік випуску, вартість доби оренди. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід на екран усіх даних з записаного раніше файлу. Програма повинна шукати авто за заданими клієнтом технічними параметрами (тип кузова, кількість місць, об’єм багажника, к/п, тощо) або за маркою, роком випуску та орієнтовною ціною добової оренди.

30. Створити програму для роботи з файлом, що містить дані про продаж вживаних автомобілів. Запис містить поля: марка авто, модель авто, тип кузова, об’єм двигуна, рік випуску, ціна. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід на екран усіх даних з записаного раніше файлу. Програма повинна вибирати список авто за такими групами параметрів: марка, модель та діапазон для ціни або марка, модель та період випуску (у роках, від… до…).

Приклад розв'язування задачі 30

30. Створити програму для роботи з файлом, що містить дані про продаж вживаних автомобілів. Запис містить поля: марка авто, модель авто, тип кузова, об'єм двигуна, рік випуску, ціна. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід на екран усіх даних з записаного раніше файлу. Програма повинна вибирати список авто за такими групами параметрів: марка, модель та діапазон для ціни або марка, модель та період випуску (у роках, від… до…).

До побудови коду прикладів спробуємо застосувати принцип ***модульності***, розділивши функціонал програми між двома незележними компонентами: перший відповідатиме виключно за зберігання даних (запис-зчитування), другий – за взаємодію програми з користувачем. Незалежність цих компонентів дасть нам можливість розробляти кожен з них окремо, в т. ч. заміняти його на іншу реалізацію з таким же функціоналом, наприклад, реалізувати зберігання даних в реляційній базі даних, чи розробити візуальний інтерфейс для взаємодії з користувачем (див. додатки до розділу).

В сучасних проєктах при розробці модуля для зберігання даних загальноприйнятою технікою є дотримання патерну проєктування ***Репозиторій***. Не претендуючи на точність та повноту реалізації цього шаблону в рамках нашої задачі, спробуємо зберегти основні його властивості в коді прикладів.

Програма на Java.

Код на ***Java*** буде містити декілька файлів, оскільки нам потрібно описати структуру даних у вигляді окремого ***типу*** (***класу***), а для того, щоб він був доступний решті коду, його потрібно оголосити зі специфікатором ***public***. В ***Java*** файл з кодом може містити лише один ***public***-клас, а його назва повиння співпадати з іменем класу. Спочатку створимо файл ***Car.java***, в якому реалізуємо клас ***Car*** з описом структури даних з інформацією про окреме авто:

```
package com.amcrossplatform.lab4;
import java.io.Serializable;

public class Car implements Serializable {
  // статичне поле, що забезпечує визначення належності
  // до цього класу десервалізованих з файлу об'єктів
  public static final long serialVersionUID = 1110111;
  // поля класу, що міститимуть дані про окреме авто
  private int id;
  private String brand;
  private String model;
  private int productionYear;
```

```
private int engine;
private double price;
// методи класу забезпечують доступ до
// полів класу (запис і зчитування даних)
public int getId() {
  return id;
}
public void setId(int id) {
  this.id = id;
}
public String getBrand() {
  return brand;
}
public void setBrand(String brand) {
  this.brand = brand;
}
public String getModel() {
  return model;
}
public void setModel(String model) {
  this.model = model;
}
public int getProductionYear() {
  return productionYear;
}
public void sepProductionYear(int year) {
  this.productionYear = year;
}
public int getEngine() {
  return engine;
}
public void setEngine(int engine) {
  this.engine = engine;
}
```

```
public double getPrice() {
  return price;
}
public void setPrice(double price) {
  this.price = price;
}
// метод toString описує та забезпечує подання
// інформації, що міститься в екземплярі класу
// в текстовому вигляді, зокрема, при виводі на консоль
@Override
public String toString() {
  return id + ") " +brand + " " + model +
      ", " + productionYear +
      ", ціна: $" + price;
}
// конструктори класу дозволяють створювати
// екземпляри класу (об'єкти) для зберігання
// даних про окреме авто
public Car() {
}
public Car(String brand, String model,
           int year, int engine, double price) {
  this.id = 0;
  this.brand = brand;
  this.model = model;
  this.productionYear = year;
  this.engine = engine;
  this.price = price;
}
public Car(int id, String brand, String model,
           int year, int engine, double price) {
  this.id = id;
  this.brand = brand;
  this.model = model;
```

```
        this.productionYear = year;
        this.engine = engine;
        this.price = price;
    }
}
```

Клас ***Car*** містить зайве, на перший погляд, поле ***id***. Потреба в ньому з'являється у випадку зберігання даних в реляційній БД, – там воно використовується в якості первинного ключа таблиці з даними. У випадку серіалізації даних у файл, його обслуговування створюватиме нам додатковий клопіт (див. код репозиторію). Проте це поле в класі необхідне для взаємозамінності репозиторіїв з різними способами зберігання даних. Для створення коду класу в IntelliJ IDEA достатньо правильно описати його поля, – методи класу, подані в коді вище, можна згенерувати автоматично за допомогою команди ***Code=>Generate*** (комбінація клавіш ***Alt+Insert***).

Процеси серіалізації-десеріалізації екземплярів класу контролюються за допомогою інтерфейсу ***Serializable***. Імплементація цього інтерфейсу не зобов'язує сам клас реалізовувати якісь специфічні методи, але «інформує» решту коду про те, що про об'єкти такого класу можна записувати у файл. Якщо в класі ***Car*** не імплементувати цей інтерфейс, ми не зможемо серіалізувати колекцію даних про автомобілі.

Для використання репозиторіїв з різними способами зберігання даних та їх можливості їх перемикання в програмі слід забезпечити класам репозиторіїв підтримку однакового інтерфейсу, тобто, набору методів. В ООП для таких цілей використовуються ***інтерфейси*** – спеціальні «класи», які в термінах абстракних методів лише оголошують потрібний функціонал. Для роботи в програмі використовують класи, що ***імплементують інтерфейс***, тобто містять програмну реалізацію оголошених методів. Інтерфейс репозиторію до нашої задачі може виглядати так:

```
package com.amcrossplatform.lab4;

import java.util.List;


// Інтерфейс описує набір методів, якими
// повинен володіти клас, щоб програма могла
// використовувати його в якості репозиторію
public interface Repository {
  // усі методи за замовчуванням вважаються відкритими
  // та абстрактними (не містять реалізації)
  List<Car> getAll();
```

```
  Car getById(int id);
  boolean addCar(Car car);
  boolean deleteCar(int id);
  List<Car> getByYear(String brand, String model,
                      int yearFrom, int yearTo);
  List<Car> getByPrice(String brand, String model,
                       float priceFrom, float priceTo);
}
```

Далі розробку проєкту можна розділити, розробляючи окремо різні реалізації самого репозиторію та код основної програми для взаємодії з користувачем. Для написання коду основної програми достатньо використовувати змінну оголошеного вище інтерфейсу. Наприклад, консольний її варіант на ***Java*** може виглядати так:

```
package com.amcrossplatform.lab4;

import java.util.List;
import java.util.Scanner;


public class Program {
  // поле для класу для підключення потрібної
  // імплементації репозиторію
  private static Repository repository;
  // головний метод (програма)
  public static void main(String[] args) {
    // оголошуємо колекцію даних інформаційної системи
    // та заповнюємо її списком з репозиторію,
    // список може бути порожнім, якщо файл репозиторію
    // щойно створено
    List<Car> cars = repository.getAll();
    // сканер для читання даних з консолі
    Scanner input = new Scanner(System.in);
    // меню програми та діалог з користувачем
    int choice;
```

```
while (true) {
  System.out.println("Оберіть потрібну дію:");
  System.out.println("1 - переглянути всі записи");
  System.out.println("2 - додати авто");
  System.out.println("3 - шукати моделі за ціною");
  System.out.println("4 - шукати моделі за віком");
  System.out.println("5 - вилучити дані про авто");
  System.out.println("0 - завершити роботу програми");
  choice = Integer.parseInt(input.nextLine());
  if(choice == 0) break;
  switch(choice){
    case 1:
      // оновлюємо список авто
      cars = repository.getAll();
      // виводимо дані про авта на консоль
      for (Car c: cars){
        System.out.println(c);
      }
      break;
    case 2:
      Car car = inputNewCar();
      repository.addCar(car);
      break;
    case 3:
      System.out.println("Задайте марку авта:");
      String brand = input.nextLine();
      System.out.println("Задайте модель авта:");
      String model = input.nextLine();
      System.out.println("Задайте ціну:\nмінімум");
      float minPrice =
              Float.parseFloat(input.nextLine());
```

```
            System.out.println("максимум");
            float maxPrice =
                    Float.parseFloat(input.nextLine());
            cars = repository.getByPrice(brand, model,
                                minPrice, maxPrice);
            cars.stream().forEach(System.out::println);
            break;
        case 4:
            System.out.println("Задайте марку авта:");
            String carBrand = input.nextLine();
            System.out.println("Задайте модель авта:");
            String carModel = input.nextLine();
            System.out.println("Задайте рік випуску:\nвід");
            int minYear = Integer.parseInt(input.nextLine());
            System.out.println("до");
            int maxYear = Integer.parseInt(input.nextLine());
            cars = repository.getByYear(carBrand, carModel,
                                        minYear, maxYear);
            cars.stream().forEach(System.out::println);
            break;
        case 5:
            System.out.println("Вкажіть id авта, " +
                "дані про яке потрібно вилучити:");
            int id = input.nextInt();
            Car found = repository.getById(id);
            if(found == null){
              System.out.println("Авто з таким id немає");
            }
            else{
              if(repository.deleteCar(id)){
                System.out.println("Дані успішно вилучено");
```

```
            }
          }
          break;
        }
      }
    }

    // Допоміжний метод, для вводу з консолі
    // даних про авто
    private static Car inputNewCar() {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Введіть дані про авто.");
      System.out.println("Марка авто:");
      String brand = scanner.nextLine();
      System.out.println("Модель:");
      String model = scanner.nextLine();
      System.out.println("Рік випуску:");
      int year = scanner.nextInt();
      System.out.println("Об\'єм двигуна:");
      int engine = scanner.nextInt();
      System.out.println("Ціна авто:");
      float price = scanner.nextFloat();
      return new Car(brand, model, year, engine, price);
    }
}
```

Поданий вище код проєкту вже є готовою програмою, але для її роботи потрібна будь-яка імплементація інтерфейсу ***Repository***, екземпляр якої потрібно зберегти в поле ***repository*** класу ***Program***. Наприклад, репозиторій на основі поданого далі коду класу ***FileRepository*** можна підключити до програми інструкцією

```
repository = new FileRepository("carforsale.dat");
```

на початку методу ***main***.

Для опрацювання колекцій з даними в коді репозиторію використовується популярна технологія ***Java StreamAPI***. Зручність її використання можна оцінити на прикладі виводу списку автомобілів в коді основної програми, де в різних пунктах використано класичний (алгоритмічний):

```
for (Car c: cars){
    System.out.println(c);
}
```

та потоковий (StreamAPI)

```
cars.stream().forEach(System.out::println);
```

стилі перегляду елементів списку.

Реалізація репозиторію зв збервганням даних у файлі:

```
package com.amcrossplatform.lab4;

import java.io.*;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

import java.util.OptionalInt;

import java.util.function.Predicate;

import java.util.stream.Collectors;


public class FileRepository implements Repository{
  private String fileName;
  private List<Car> cars;
  // констуктор класу, що дозволяє створити екземпляр
  // репозиторію асоційований з заданою назвою файла
  public FileRepository(String fileName) {
    this.cars = new ArrayList<>();
```

```
  this.fileName = fileName;
  if ((new File(fileName)).exists()){
    this.reloadData();
  }
}
// метод для отримання списку усіх автомобілів
@Override
public List<Car> getAll() {
  reloadData();
  return cars;
}
// допоміжний метод виконує “синхронізацію” даних,
// які використовує програма з даними у файлі
private void reloadData() {
  if ((new File(fileName)).exists()) {
    try (FileInputStream fileInputStream =
                new FileInputStream(fileName)) {
      try (ObjectInputStream objectInputStream =
         new ObjectInputStream(fileInputStream)) {
        cars = (List<Car>) objectInputStream
                                        .readObject();
      } catch (ClassNotFoundException e) {
        throw new RuntimeException(e);
      }
    } catch (FileNotFoundException e) {
      throw new RuntimeException(e);
    } catch (IOException e) {
      throw new RuntimeException(e);
    }
  }
}
```

```
@Override
public Car getById(int id) {
  reloadData();
  Optional<Car> foundCar = this.cars.stream()
      .filter(new Predicate<Car>() {
        @Override
        public boolean test(Car car) {
          return car.getId()==id;
        }
      })
      .findFirst();
  Car car = null;
  if(foundCar.isPresent()){
    // якщо авто з таким id знайдено,
    // то метод поверне відповідний
    // об'єкт, якшо ні, то значення null
    car = foundCar.get();
  }
  return car;
}
// додавання даних про нове авто
@Override
public boolean addCar(Car car) {
  int id = 0;
  reloadData();
  if(this.cars.size()>0) {
    OptionalInt maxId = this.cars.stream()
        .mapToInt(c -> c.getId()).max();
    if (maxId != null) {
```

```
      id = maxId.getAsInt();
    }
  }
  car.setId(id+1);
  cars.add(car);
  try {
    save();
  } catch (IOException e) {
    return false;
  }
  return true;
}
// метод для запису списку автомобілів у файл допомагає
// “синхронізувати” дані, які містить список з даними
// у файлі, перезаписуючи його
private void save() throws IOException {
  try(FileOutputStream fileOutputStream =
                new FileOutputStream(fileName)){
    try(ObjectOutputStream objectOutputStream =
            new ObjectOutputStream(fileOutputStream)){
      objectOutputStream.writeObject(this.cars);
    }
  }
}


@Override
public boolean deleteCar(int id) {
  // оновлюємо список автомобілів
  reloadData();
  // шукаємо авто за заданим id
```

```
    Optional<Car> foundCar =
cars.stream().filter(c->c.getId()==id).findFirst();
    // якщо авто з таким id є в списку, видаляємо
    // його та зберігаємо змінений мписок у файлі
    if(foundCar.isPresent()){
      Car car = foundCar.get();
      cars.remove(car);
      try {
        save();
      } catch (IOException e) {
        throw new RuntimeException(e);
      }
      return true;
    }
    return false;
  }

  @Override
  public List<Car> getByYear(String brand, String model,
                             int yearFrom, int yearTo) {
    reloadData();
    List<Car> result = cars.stream()
        .filter(new Predicate<Car>() {
          @Override
          public boolean test(Car car) {
            return car.getBrand().equalsIgnoreCase(brand)
              && car.getModel().equalsIgnoreCase(model)
              && car.getProductionYear() >= yearFrom
              && car.getProductionYear() <= yearTo ;
          }
        })
```

```
        .collect(Collectors.toList());
    return result;
  }

  @Override
  public List<Car> getByPrice(String brand, String model,
                        float priceFrom, float priceTo) {
    reloadData();
    List<Car> result = cars.stream()
        .filter(new Predicate<Car>() {
          @Overrid e
          public boolean test(Car car) {
            return car.getBrand().equalsIgnoreCase(brand)
              && car.getModel().equalsIgnoreCase(model)
              && car.getPrice() >= priceFrom
              && car.getPrice() <= priceTo ;
          }
        })
        .collect(Collectors.toList());
    return result;
  }
}
```

Програма на Kotlin.

В прикладі на ***Kotlin*** також дотримуватимемось парадигми ООП. Опишемо в нашому прроєкті клас Car для зберігання даних про окреме авто. За озвученими вище правилами, його уод може виглядати так:

```
package com.amcrossplatform.lab4

import java.io.Serializable

// оголошення класу та первинний конструктор
class Car ( var brand: String,
            var model: String,
            var year: Int = 0,  var engine: Int = 0,
            var price: Float = 0.0f): Serializable {
    // тіло класу
    // поле класу
    var id: Int = 0
    // решта полів класу оголошено в первинному
    // конструкторі
    // конструктор без переметрів
    constructor():this("Unknown", "Unknown"){
    }
    // метод для текстового представлення даних
    // про окремий екземпляр класу
    override fun toString(): String {
        return id.toString() + ") " + brand + "," + model +
                ", " + year +
                ", ціна: " + price +
                " грн"
    }
    // ще один конструктор
    constructor(id: Int, brand: String, model: String,
         year: Int, engine: Int, price: Float): this(brand,
         model, year, engine, price) {
        this.id = id
```

```
    }
}
```

Серіалізація об’єктів в ***Kotlin*** організована за допомогою спеціальних класів-серіалізаторів. Це, з одного боку, дозволяє підтримувати різні формати серіалізації, а з іншого – виносить процес сам процес за межі класу з даними. В нашому коді ми використовуватимемо класичну бінарну серіалізацію та відповідні файлові потоки ***Java***.

Клас ***Car*** в ***Kotlin*** можна було оголосити як клас даних (***Data Class***). В такому випадку його код був би значно коротшим, але тоді він міститиме лише члени, які додаються до датакласів за замовчуванням.

Враховуватимемо перспективу різних способами зберігання даних та можливість використання декількох репозиторіїв. ООП в ***Kotlin*** дозволяє використовувати ***інтерфейси***, так само, як і вінших об’єктно-орієнтованиї мовах. Інтерфейс репозиторію до нашої задачі на ***Kotlin*** може виглядати так:

```
package com.amcrossplatform.lab4

interface Repository {
    fun getAll(): List<Car>
    fun getById(id: Int): Car
    fun addCar(car: Car): Boolean
    fun deleteCar(id: Int): Boolean
    fun getByYear(carBrand: String, carModel: String,
                  minYear: Int, maxYear: Int): List<Car>
    fun getByPrice(brand: String, model: String,
              minPrice: Float, maxPrice: Float): List<Car>
}
```

Відразу реалізуємо один репозиторій у вигляді класу. Код, як зазначалося вище використовує бінарну серіалізацію та потокові класи з пакету java.io. Застосування ***Java StreamAPI*** до пошуку даних перекладено з java-реалізації репозиторію на ***Kotlin*** за допомогою транслятора, вбудованого в IntelliJ IDEA.

```
package com.amcrossplatform.lab4

import java.io.*

import java.util.stream.Collectors

class FileRepository(private val fileName: String) :

Repository {

  private var cars: MutableList<Car>
```

```
// ініціалізатор створює порожній список для
// того, щоб не було звертання до null-об'єктв
init {
  cars = ArrayList()
}
// метод повертає актуальний список автомобілів
override fun getAll(): List<Car> {
  reloadData()
  return cars
}
// метод синхронізує список автомобілів, зчитуючи
// актуальні дані з файла
private fun reloadData() {
  if (File(fileName).exists()) {
    try {
      FileInputStream(fileName).use {
        fileInputStream -> try {
          ObjectInputStream(fileInputStream).use {
            objectInputStream ->
            cars = objectInputStream
                 .readObject() as MutableList<Car>
          }
        } catch (e: ClassNotFoundException) {
          throw RuntimeException(e)
        }
      }
    } catch (e: FileNotFoundException) {
      throw RuntimeException(e)
    } catch (e: IOException) {
      throw RuntimeException(e);
    }
```

```
  }
}

override fun getById(id: Int): Car {
  return Car()
}

override fun addCar(car: Car): Boolean {
  var id = 0
  if (cars.size > 0) {
    val maxId = cars.stream().mapToInt {
                                c: Car -> c.id }.max()
    if (maxId != null) {
      id = maxId.asInt
    }
  }
  car.id = id + 1
  cars.add(car)
  try {
    save()
  } catch (e: IOException) {
    return false
  }
  return true
}
// метод використовується лише всередині класу в
// методі додавання нового запису, тому може бути закритим
@Throws(IOException::class)
private fun save() {
  FileOutputStream(fileName).use {
    fileOutputStream ->
```

```
    ObjectOutputStream(fileOutputStream)
          .use { objectOutputStream ->
               objectOutputStream.writeObject( cars )
    }
  }
}
override fun deleteCar(id: Int): Boolean {
  // оновлюємо список автомобілів
  reloadData()
  // шукаємо авто за заданим id
  val foundCar = cars.stream().filter {
           c: Car -> c.id == id }.findFirst()
  // якщо авто з таким id є в списку, видаляємо
  // його та зберігаємо змінений мписок у файлі
  if (foundCar.isPresent) {
    val car = foundCar.get()
    cars.remove(car)
    try {
      save()
    } catch (e: IOException) {
      throw RuntimeException(e)
    }
    return true
  }
  return false
}
override fun getByYear(
  brand: String, model: String,
  yearFrom: Int, yearTo: Int
             ): List<Car> {
  reloadData()
```

```
    return cars.stream()
      .filter { car ->
        (car.brand.equals(brand, ignoreCase = true)
            && car.model.equals(model, ignoreCase = true)
            && car.year >= yearFrom) && car.year <= yearTo
      }
      .collect(Collectors.toList())
  }
  override fun getByPrice(
    brand: String, model: String,
    priceFrom: Float, priceTo: Float
                ): List<Car> {
    reloadData()
    return cars.stream()
      .filter { car ->
        (car.brand.equals(brand, ignoreCase = true)
            && car.model.equals(model, ignoreCase = true)
        && car.price >= priceFrom) && car.price <= priceTo
      }
      .collect(Collectors.toList())
  }
}
```

Код основної програми з консольним текстовим інтерфейсом користувача можна реалізувати у вигляді такої функції:

```
package com.amcrossplatform.lab4

fun main(args: Array<String>) {
  var repository: Repository
  // створюємо екземпляр репозиторію для зберігання даних
  repository = FileRepository("carforsale.dat")
  // оголошуємо колекцію даних інформаційної системи
  // та заповнюємо її списком що міститься в репозиторії,
  // список може бути порожнім, якщо файл репозиторію
```

```
// щойно створено
var cars: List<Car> = repository.getAll()
// меню програми та діалог з користувачем
var choice: Int
while (true) {
  println("Оберіть потрібну дію:")
  println("1 - переглянути всі записи")
  println("2 - додати авто")
  println("3 - шукати моделі за ціною")
  println("4 - шукати моделі за віком")
  println("5 - вилучити дані про авто")
  println("0 - завершити роботу програми")
  choice = readln().toInt()
  if (choice == 0) break
  when (choice) {
    1 -> {
      // оновлюємо список авто
      cars = repository.getAll()
      // виводимо дані про авта на консоль
      for (c in cars) {
        println(c)
      }
    }

    2 -> {
      val car = inputNewCar()
      repository.addCar(car)
    }

    3 -> {
      println("Задайте марку авта:")
```

```
  val brand = readln()
  println("Задайте модель авта:")
  val model = readln()
  println("Задайте ціну:\nмінімум")
  val minPrice = readln().toFloat()
  println("максимум")
  val maxPrice = readln().toFloat()
  cars = repository.getByPrice(
              brand, model, minPrice, maxPrice)
  cars.stream().forEach { x: Car? -> println(x) }
}

4 -> {
  println("Задайте марку авта:")
  val carBrand = readln()
  println("Задайте модель авта:")
  val carModel = readln()
  println("Задайте рік випуску:\nвід")
  val minYear = readln().toInt()
  println("до")
  val maxYear = readln().toInt()
  cars = repository.getByYear(
            carBrand, carModel, minYear, maxYear)
  cars.stream().forEach { x: Car? -> println(x) }
}
5 -> {
  println(
    "Вкажіть id авта, " +
        "дані про яке потрібно вилучити:"
    )
    val id = readln().toInt()
```

```
            val found = repository.getById(id)
            if (found == null) {
              println("Авто з таким id не знайдено")
            } else {
              if (repository.deleteCar(id)) {
                println("Дані успішно вилучено")
              }
            }
          }
        }
      }
    }

    // Допоміжна функція, для вводу з консолі
    // даних про авто
fun inputNewCar(): Car {
    println("Введіть дані про авто.")
    println("Марка авто:")
    val brand = readln()
    println("Модель:")
    val model = readln()
    println("Рік випуску:")
    val year = readln().toInt()
    println("Об\'єм двигуна:")
    val engine = readln().toInt()
    println("Ціна авто:")
    val price = readln().toFloat()
    return Car(brand, model, year, engine, price)
}
```

Створений проєкт, звісно, можна модифікувати, наприклад, реалізувавши репозиторій для роботи з реляційною базою даних (замість збереження даних у файлі) чи розробити візуальний інтерфейс користувача замість консольного.