Крос-платформне програмування

ЛАБОРАТОРНА РОБОТА № 6

Зберігання даних з використанням Java Database Connectivity.

Створення візуального UI з використанням JavaFX.

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторної роботи
.

Львів

2024

Мета роботи: Ознайомлення з сценарієм збирання Java-проєктів Maven. Вивчення основних підходів технології Java Database Connectivity та її використання до роботи з реляційними базами даних. Формування елементарних навиків використання бібліотеки візуальних компонентів JavaFX для створення графічного інтерфейсу користувача.

Завдання до лабораторної роботи.

Для проєкту простої ***інформаційної системи***, розробленого у ході виконання лабораторної роботи № 4 додати наступний функціонал:

1. Створити реалізацію репозиторію для зберігання даних ІС у ***реляційній базі даних*** (MySQL, MS SQLServer, H2, тощо), реалізувавши увесь запланований функціонал з використанням мови ***Java***.
2. Розробити на зручний візуальний інтерфейс користувача у вигляді ***JavaFX-додатку***.

Оформити письмовий звіт про виконання роботи.

Звіт повинен містити:

- титульний аркуш:
- умову задачі відповідного варіанту;
- код репозиторію для зберігання даних на Java;
- малюнки з зображенням конструкції вікон, що використовуватимуться для взаємодії користувача з програмою;
- код обробників подій візуальних компоненнтів на Java;
- знімки екрану з результатами роботи створеної системи для кожного з елементів реалізованого функціоналу;

** За бажанням, додатково можна реалізувати аналогічний проєкт на Kotlin, переклавши основні фрагменти Java-коду за допомогою вбудованого в IntelliJ IDEA втоматичного перекладача.

Короткі теоретичні відомості

Java DataBase Connectivity

JDBC (Java DataBase Connectivity) - прикладний інтерфейс програмування, який дозволяє працювати з будь-яким сервером баз даних (пакет java.sql).

Для підключення до базу даних використовуються так звані JDBC-драйвери.

В API JDBC закладено концепцію інтерфейсів, які повинні бути реалізовані постачальниками того чи іншого сервісу. У випадку баз даних постачальник надає розробнику так званий драйвер JDBC.

За способом реалізації драйверів розділяються на 4 типи:

1. JDBC-ODBC міст. До цього типу належать драйвери, реалізовані поверх драйвера ODBC. Фактично всі виклики API JDBC транслюються у виклики ODBC, а також подальша обробка виконується API ODBC.
2. Native-JDBC. До другого типу належать драйвери, що використовують програмні частини, написані на інших мовах.
3. Network-JDBC. Цей тип драйвера повністю реалізується на Java, але при цьому виклики JDBC передаються в сетевий протокол (RMI, HTTP та ін.), що далі транслюється в конкретний протокол баз даних.
4. "Чистий" драйвер JDBC. Схожий до 3-го типу, реалізується повністю на Java, але виклики реалізуються з використанням протоколу баз даних, без мережевих протоколів.

Інтерфейс \ Клас	**Інтерфейси класи для роботи с СУБД**
DriverManager	Клас для керування драйверами в Java додатку. За його допомогою можна створити підключення до БД, використовуючи потрібний драйвер.
Connection	Інтерфейс описує методи для з’єднання з БД
Statement	Інтерфейс містить набір методів для обробки запитів до БД
PreparedStatement	Інтерфейс містить набір методів для обробки параметричних запитів до БД
CallableStatement	Інтерфейс містить набір методів для обробки роботи зі Сторед-процедурами
ResultSet	Клас представляє набір даних (результат роботи оператора SELECT)
DatabaseMetaData	Класс описує інформацію про базу даних

ResultSetMetaData	Класс описує інформацію про результуючий набір

Для отримання результуючого набору даних (екземпляру класу ResultSet), потрібно викликати метод executeQuery інтерфейсу Statement.

Методи ResultSet	**Опис**
boolean first()	Перехід на перший запис, true якщо перехід успішний.
boolean last()	Переходить на останній запис, true якщо перехід успішний.
boolean next()	Переходить на наступний запис в результуючому наборі, true якщо перехід успішний.
boolean previous()	Переходить на попередній запис в результуючому наборі, true якщо перехід успішний.
boolean absolute(int numRecord)	Переходить на вказаний запис в результуючому наборі
<Тип даних> get<Тип даних>(int index) <Тип даних> get<Тип даних>(String name)	Повертає значення поля поточного рядка, приведене до певного типу даних (getString, getInt, тощо). Має дві перегрузки, для звертання до поля за індексом (індексування починається з одиниці) і за назвою поля.
void close()	Закриває набір даних (звільняє ресурси витрачаються для обробки набору даних). Після закриття, набір даних недоступний.

JavaFX

Історія JavaFX почалася в першій половині 2000-х років, коли Кріс Олівер (Chris Oliver) з компанії SeeBeyond, розробив для створення графічних інтерфейсів нову мову F3 (Froms Follows Functions).

У 2005 році SeeBeyond була придбана компанією Sun Microsystems (на той момент розвивала мову Java). F3 перейменували в JavaFX, а Кріс Олівер продовжив роботу над новою платформою вже в рамках компанії Sun. 4 грудня 2008 року вийшов JavaFX 1.0 SDK.

Після придбання Sun Microsystems компанією Oracle в 2010 році була анонсована, а в 2011 році вийшла в реліз версія JavaFX 2.0. Скриптова мова, що була основою перших версій була вилучена, а платформа була повністю переписана фактично з нуля і почала працювати на Java.

Версія JavaFX 2.0 дозволила створювати додатки за допомогою будь-якої мови, який підтримувала JVM. Були додані нові API, інтеграція зі Swing і багато інших речей.

Наступними віхами в розвитку платформи стали версії JavaFX 8 і особливо JavaFX 9, яка вийшла у вересні 2017 року разом з Java 9 і привнесла в платформу модульність. І якщо раніше JavaFX поставлялася разом з Java SE, то зараз JavaFX відокремлена від основної функціональності Java SE і використовується як окремий модуль.

Остання версія фреймворку - JavaFX 15 - вийшла у вересні 2020 року. На даний момент JavaFX являє кращий спосіб для створення графічних додатків за допомогою мови Java, який прийшов на зміну AWT і Swing.

Завдання.

Розробити просту інформаційну систему, яка надасть користувачеві основні можливості з опрацювання вказаної у варіанті інформації та її зберігання в реляційній базі даних, а також пошуку даних за вказаними у варіанті завдання критеріями. Взаємодію ІС з користувачем реалізувати за допомогою віконного графічного інтерфейсу з використанням технології JavaFX.

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 IT-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.

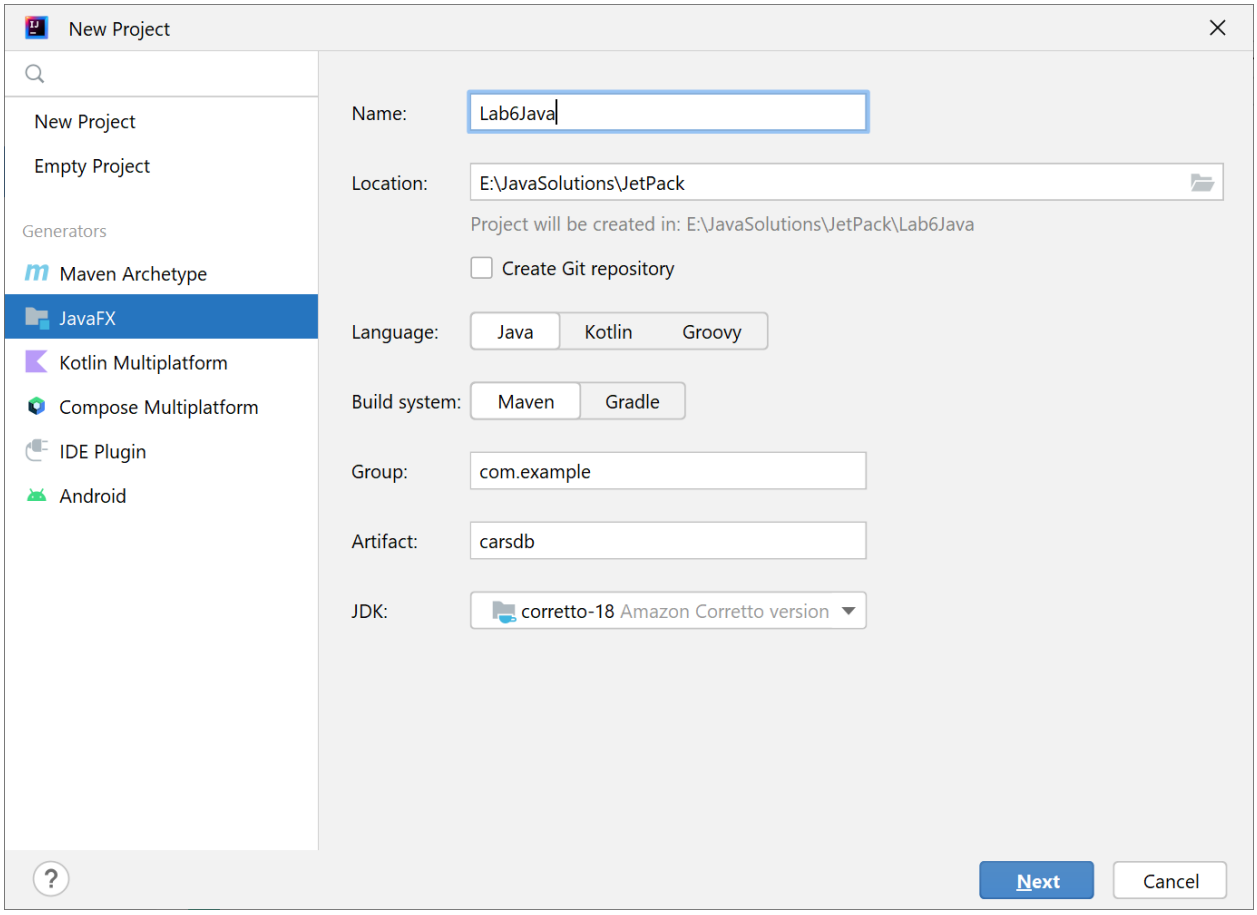
30. Створити програму для роботи з файлом, що містить дані про продаж вживаних автомобілів. Запис містить поля: марка авто, модель авто, тип кузова, об’єм двигуна, рік випуску, ціна. В програмі передбачити ввід даних користувачем та збереження їх у файл, а також вивід на екран усіх даних з записаного 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…).

Частина 1. Створення проєкту

Запускаємо IntelliJ IDEA. Створюємо новий проєкт JavaFX з використанням мови програмування Java та сценарію Maven.



На наступному кроці додаткових бібліотек не обираємо, та переходимо до створення проєкту. Запускаємо створений проєкт, щоб переконатися в його працездатності. Закриваємо запущене вікно додатку. Змінимо в проєкті дефолтні назви контролера та FXML- файлу з HelloController та hello-view на MainController та main-view відповідно, а HelloApplication на CarsShop. Ще раз перевіряємо роботу вікна, запустивши проєкт на виконання.
Код класу CarsShop далі змінювати не будемо, оскільки його завдання бути точкою входу, та виконувати запуск головного вікна нашої програми

```
package com.example.carsdb;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;
import java.io.IOException;

public class CarsShop extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(
                CarsShop.class.getResource("main-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 320, 240);
        stage.setTitle("Cars for sale");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```

Частина 2. Реалізація репозиторію для зберігання даних за допомогою JDBC.

Переходимо до створення репозиторію зі списком даних. Додамо всередину пакету, що містить згенеровані класи застосунку та контролера пакет data, в ньому реалізуємо клас моделі Car. Клас моделі можна перенести з попередньої л.р., врахувавши при цьому розміщення коду в пакетах.

```
package com.example.carsdb.data;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class Car implements Serializable {
    public static final long serialVersionUID =
                                            4748325733388567091l;
    private int id;
    private String brand;
    private String model;
    private int year;
    private int engine;
    private double price;
    public Car() {
    }
```

```
@Override
public String toString() {
    return brand + ", " + model +
            ", " + year +
            ", ціна: $" + price;
}

    public Car(String brand, String model, int year,
               int engine, double price) {
        this.brand = brand;
        this.model = model;
        this.year = 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.year = year;
        this.engine = engine;
        this.price = price;
    }

    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 getYear() {
        return year;
    }
```

```
    public void setYear(int year) {
        this.year = 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;
    }
}
```

Оголосимо інтерфейс репозиторію зі списком автомобілів (інтерфейс описано у попередній л.р.):

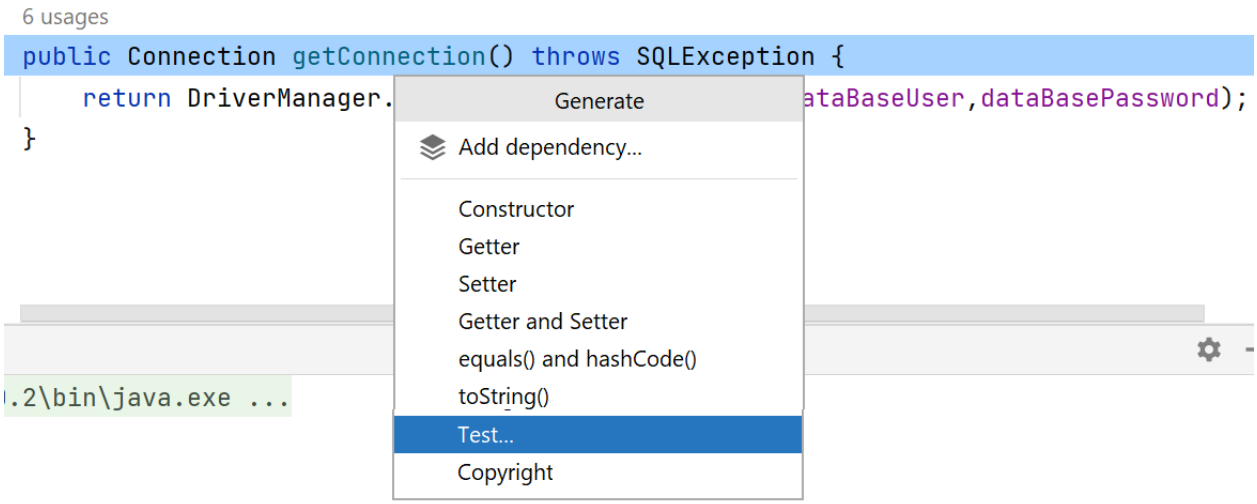
```
package com.example.carsdb.data;

import java.util.List;

public interface Repository {
    List<Car> getAll();
    Car getById(int id);
    List<Car> getAllByBrand(String brand);
    boolean addCar(Car car);
    boolean updateCar(int id, Car car);
    boolean deleteCar(int id);
}
```

Зберігатимемо дані інформаційної системи в локальній реляційній базі даних H2, яка використовується для розробки та тестування Java-програм. СКБД H2 дозволяє зберігати дані в оперативній пам’яті або у файлі на диску. Для цього до сценарію збирання у файлі pom.xml, всередину тега <dependencies> </dependencies> додаємо залежність (номер версії може відрізнятися):

```
<!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>2.1.214</version>
    <scope>compile</scope>
</dependency>
```

Створення та отримання підключення до БД реалізуємо у вигляді окремого класу, в якому задамо потрібні параметри:

```
package com.example.carsdb.data;

import java.lang.reflect.InvocationTargetException;
import java.sql.*;

public class DataBaseConnector {
    private String dataBaseUrl;
    private String dataBaseUser;
    private String dataBasePassword;
    private String driverClass;

    public DataBaseConnector(String dataBaseName) {
        this.dataBaseUrl = "jdbc:h2:file:./" + dataBaseName;
        this.dataBaseUser = "admin";
        this.dataBasePassword = "";
        this.driverClass = "org.h2.Driver";
    }

    public boolean testDriver(){
        try{
            Class.forName(driverClass)
                .getDeclaredConstructor().newInstance();
            return true;
        }catch (ClassNotFoundException e) {
            e.printStackTrace();
            return false;
        } catch (IllegalAccessException e) {
            e.printStackTrace();
            return false;
        } catch (InstantiationException e) {
            e.printStackTrace();
            return false;
        } catch (NoSuchMethodException e) {
            e.printStackTrace();
            return false;
        } catch (InvocationTargetException e) {
            e.printStackTrace();
            return false;
        }
    }

    public Connection getConnection() throws SQLException {
        return DriverManager.getConnection(dataBaseUrl,
                                dataBaseUser,dataBasePassword);
    }
}
```



Для перевірки працездатності створеного коду не конче чекати доки готової до роботи програми, це можна зробити за допомогою Unit-тестів (залежності Junit додаються в pom.xml за замовчуванням). Для створення тесту достатньо на заголовку будь-якого методу класу натиснути комбінацію клавіш Alt+Insert, а далі вибрати методи класу, які потрібно тестувати – IntelleJ IDEA сама створить клас тестів у відповідному пакеті та методи для написання тестів. Для перевірки класу `DataBaseConnector` реалізуємо та виконаємо наступні тести:

```
class DataBaseConnectorTest {

    @Test
    void testDriver() {
        DataBaseConnector testConnector  =
                    new DataBaseConnector("testDB");
        assertTrue(testConnector.testDriver());
    }

    @Test
    void getConnection() {
        DataBaseConnector testConnector  =
                    new DataBaseConnector("testDB");
        try {
            assertNotNull(testConnector.getConnection());
        } catch (SQLException e) {
            fail();
        }
    }
}
```

У випадку проходження (успішного виконання командою Run) обидвох тестів переходимо до реалізації репозиторію. Орієнтовний код репозиторію з використанням реляційної БД:

```
package com.example.carsdb.data;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class DataBaseRepository implements  Repository{

    private DataBaseConnector dataBaseConnector;

    public DataBaseRepository(
               DataBaseConnector dataBaseConnector) {
        this.dataBaseConnector = dataBaseConnector;
     try (Connection conn = dataBaseConnector.getConnection()) {
            String tableCreateStr =
                    "CREATE TABLE IF NOT EXISTS Cars\n" +
                         "(id INT NOT NULL AUTO_INCREMENT, " +
                   "Brand VARCHAR(50), Model VARCHAR(50)," +
                      "_Year INT, Engine INT, Price DOUBLE, " +
                                "PRIMARY KEY (id));";
            Statement createTable = conn.createStatement();
            createTable.execute(tableCreateStr);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    public List<Car> getAll() {
        List<Car> cars = new ArrayList<>();

 try(Connection connection = dataBaseConnector.getConnection()){
    Statement statement = connection.createStatement();
    ResultSet rs = statement.executeQuery("select * from Cars");
            while(rs.next()){
                cars.add(new Car(rs.getInt(1),
                rs.getString(2),
                rs.getString(3),
                rs.getInt(4),
                rs.getInt(5),
                rs.getDouble(6)));
            }
            rs.close();
        } catch (SQLException exception) {
          System.out.println("Не відбулося підключення до БД");
            exception.printStackTrace();
       }
        return cars;
    }
```

```
    @Override
    public List<Car> getAllByBrand(String brand) {
        List<Car> cars = new ArrayList<>();
try(Connection connection = dataBaseConnector.getConnection()){
            PreparedStatement statement =
                 connection.prepareStatement(
                "select * from Cars where Brand Like(?)"
                    );
        statement.setString(1, "%" + brand + "%");
        ResultSet rs = statement.executeQuery();
        while(rs.next()){
            cars.add(new Car(rs.getInt(1),
                    rs.getString(2),
                    rs.getString(3),
                    rs.getInt(4),
                    rs.getInt(5),
                    rs.getDouble(6)));
        }
        rs.close();
        } catch (SQLException exception) {
           System.out.println("Не відбулося підключення до БД");
          exception.printStackTrace();
        }
        return cars;
    }

    @Override
    public Car getById(int id) {
        Car car = null;
try(Connection connection = dataBaseConnector.getConnection()){
            PreparedStatement statement =
connection.prepareStatement("select * from Cars where id = ?");
            statement.setInt(1, id);
            ResultSet rs = statement.executeQuery();
            if(rs.next()){
                car = new Car(rs.getInt(1),
                        rs.getString(2),
                        rs.getString(3),
                        rs.getInt(4),
                        rs.getInt(5),
                        rs.getDouble(6));
            }
            rs.close();
        } catch (SQLException exception) {
            exception.printStackTrace();
        }finally {
            return car;
        }
    }

    @Override
    public boolean addCar(Car car) {
        int updCount = 0;
```

```
        try (Connection conn = dataBaseConnector.getConnection()) {
                PreparedStatement preparedStatement =
         conn.prepareStatement("INSERT INTO Cars (Brand, Model," +
                        "_Year, Engine, Price) VALUES (?,?,?,?,?)");
                preparedStatement.setString(1, car.getBrand());
                preparedStatement.setString(2, car.getModel());
                preparedStatement.setInt(3, car.getYear());
                preparedStatement.setInt(4, car.getEngine());
                preparedStatement.setDouble(5, car.getPrice());
                updCount = preparedStatement.executeUpdate();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
            return updCount>0;
        }

        @Override
        public boolean updateCar(int id, Car car) {
            int updCount = 0;
         try (Connection conn = dataBaseConnector.getConnection()) {
                PreparedStatement preparedStatement =
                   conn.prepareStatement( "UPDATE Cars " +
                                "SET Brand = ?, Model = ?," +
                                "_Year = ?, Engine = ?, Price = ?" +
                                "WHERE id = ?");
                preparedStatement.setString(1, car.getBrand());
                preparedStatement.setString(2, car.getModel());
                preparedStatement.setInt(3, car.getYear());
                preparedStatement.setInt(4, car.getEngine());
                preparedStatement.setDouble(5, car.getPrice());
                preparedStatement.setInt(6, id);
                updCount = preparedStatement.executeUpdate();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
            return updCount>0;
        }

        @Override
        public boolean deleteCar(int id) {
            int updCount = 0;
        try (Connection conn = dataBaseConnector.getConnection()) {
                PreparedStatement preparedStatement =
            conn.prepareStatement("DELETE FROM Cars WHERE id = ?");
                preparedStatement.setInt(1, id);
                updCount = preparedStatement.executeUpdate();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
            return updCount>0;
        }
}
```

Клас репозиторію варта також протестувати, наприклад, за допомогою тесту:

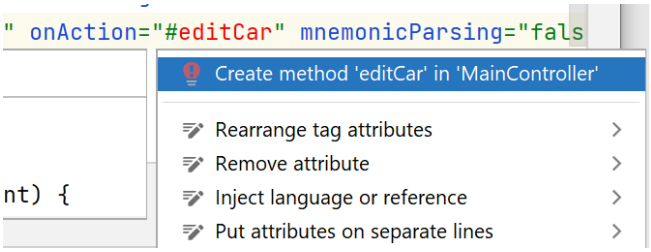
```
class DataBaseRepositoryTest {

    @Test
    void addCar() {
        DataBaseRepository repository = new DataBaseRepository(
                new DataBaseConnector("test2"));
        repository.addCar(new Car("test brand",
                                  "test car", 2023,0,0));
        repository.addCar(new Car("test brand","test car 2",
                                     2023,10,100));
        assertNotNull(repository.getById(1));
        java.util.List<Car> cars = repository.getAll();
        assertTrue(cars.size()>0);
    }

    @Test
    void getByBrand() {
        DataBaseRepository repository = new DataBaseRepository(
            new DataBaseConnector("test3"));
        repository.addCar(new Car("test brand",
                                  "test car", 2023,0,0));
        repository.addCar(new Car("test brand",
                                 "test car 2", 2023,10,100));
        java.util.List<Car> cars =
                    repository.getAllByBrand("test brand");
        assertNotNull(repository.getById(1));
        assertTrue(cars.size()>1);
    }
}
```

Зауваження 1. Файли, які створює сервер H2-database розміщуються в кореневій папці проєкту, - після успішного проходження тестів їх можна видалити. На практиці ж unit-тести виконуються при кожному збиранні проєкту, але там використовують інший спосіб зберігання H2 баз даних, – “in-memory”.

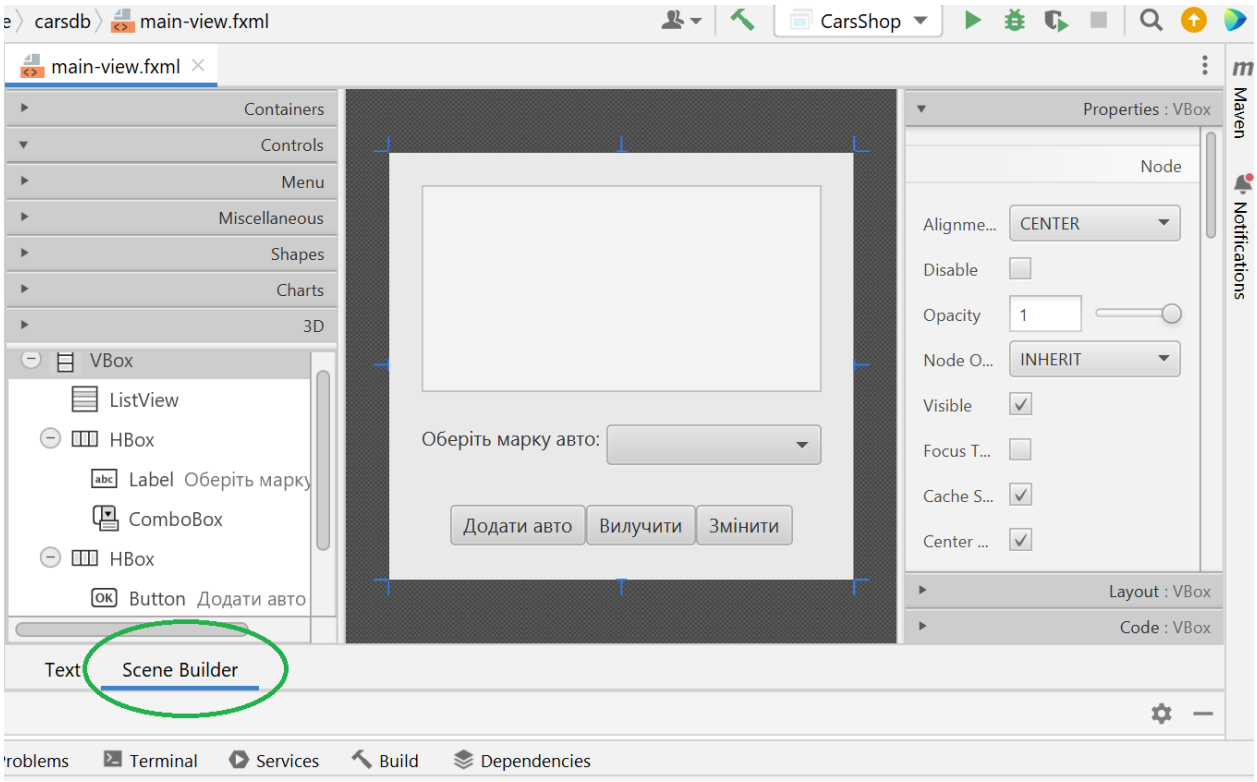
Зауваження 2. Код репозиторію придатний до використання з іншими базами даних, наприклад, MySQL. Для цього достатньо змінити лише назву драйвера та параметри підключення в класі `DataBaseConnector`. І, звичайно, не забути додати відповідну залежність у сценарій збирання в файлі pom.xml.

Частина 2. Реалізація репозиторію для зберігання даних за допомогою JDBC.

Переходимо до розробки візуальної частини. Відкриваємо файл з розміткою вікна
main-view.fxml переходимо до Scene Builder і надаємо вікну потрібного вигляду, розміщуючи на ньому потрібні візуальні компоненти. Обробники подій до кнопок та розкривного списку можна додати через редактор Scene Builder-а, або у режимі розмітки за допомогою атрибута onAction тегу відповідного компонента, наприклад, `onAction="#editCar"`



Можна автоматично створити відповідний метод в контролері, для цього слід перевести курсор в назву обробника (який позначено як помилку червоним кольором) та, натиснувши комбінацію клавіш Alt+Enter вибрати відповідний пункт з контекстного меню.



FXML – розмітка головного вікна виглядатиме, приблизно так:

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<VBox alignment="CENTER" prefHeight="264.0" prefWidth="287.0"
spacing="20.0" xmlns="http://javafx.com/javafx/17.0.2-ea"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.example.carsdb.MainController">
    <padding>
        <Insets bottom="20.0" left="20.0"
                        right="20.0" top="20.0" />
    </padding>
   <children>
      <ListView fx:id="listCars" prefHeight="200.0"
                                        prefWidth="200.0" />
      <HBox alignment="TOP_RIGHT" prefHeight="100.0"
                                        prefWidth="200.0">
         <children>
            <Label prefHeight="17.0" prefWidth="118.0"
                            text="Оберіть марку авто: " />
         <ComboBox fx:id="brandCombo" onAction="#filterByBrand"
                   prefHeight="25.0" prefWidth="137.0" />
         </children>
      </HBox>
      <HBox alignment="CENTER" prefHeight="100.0"
                                        prefWidth="200.0">
         <children>
           <Button mnemonicParsing="false" onAction="#addNewCar"
                                    text="Додати авто" />
            <Button alignment="CENTER" mnemonicParsing="false"
                    onAction="#deleteCar" text="Вилучити" />
            <Button alignment="CENTER_RIGHT" onAction="#editCar"
                     mnemonicParsing="false" text="Змінити" />
         </children>
      </HBox>
   </children>
</VBox>
```

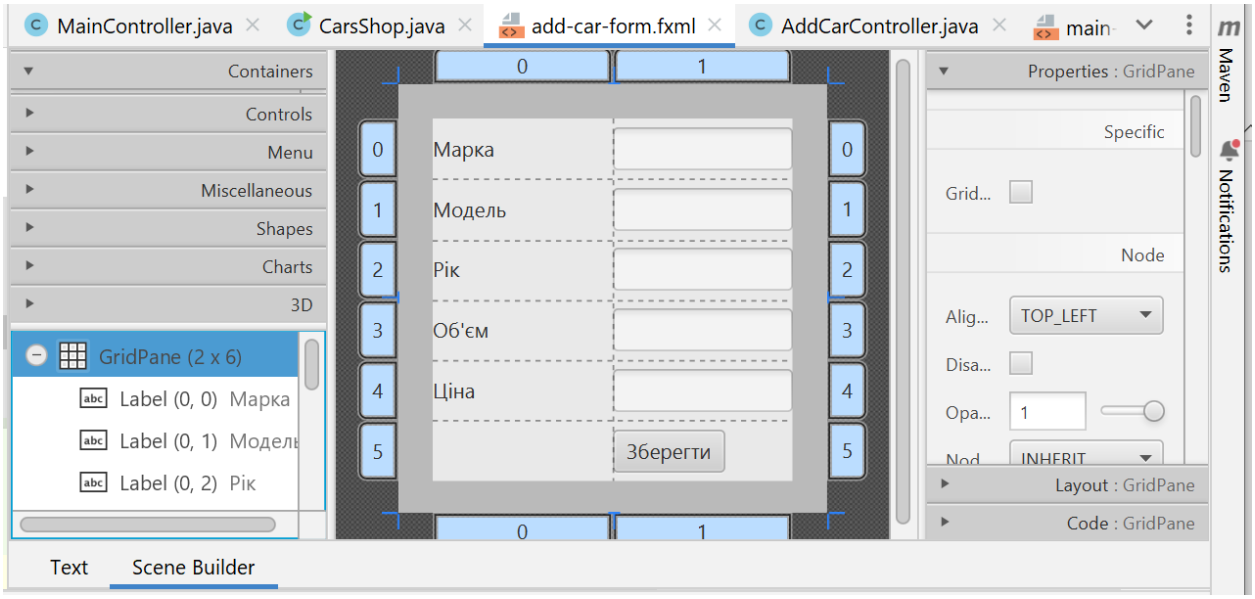
Переходимо до редагування контролера. Логіка роботи контролера передбачає заповнення вікна даними на старті, тому контролер повинен імплементувати інтерфейс Initializable. Орієнтовний код контролера головного вікна

```
package com.example.carsdb;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
```

```
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.ComboBox;
import javafx.scene.control.ListView;
import javafx.stage.Stage;
import com.example.carsdb.data.Car;
import com.example.carsdb.data.DataBaseConnector;
import com.example.carsdb.data.DataBaseRepository;
import com.example.carsdb.data.Repository;

import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;

public class MainController implements Initializable {

    @FXML
    ListView listCars;
    @FXML
    ComboBox brandCombo;
    private Repository repository;

    @Override
    public void initialize(URL url,
                ResourceBundle resourceBundle) {
        repository = new DataBaseRepository(
                new DataBaseConnector("carsShopDB"));
        updateListsView();
    }

    public void updateListsView() {
        List<Car> cars = repository.getAll();
        ObservableList<Car> carsList =
                 FXCollections.observableList(cars);
        listCars.setItems(carsList);
        List<String> brands = new ArrayList<>();
        brands.addAll(
                cars
                .stream()
                .map(car->car.getBrand())
                .distinct()
                .toList()
        );
        brands.add("all");
        ObservableList<String> brndList =
                FXCollections.observableList(brands);
        brandCombo.setItems(brndList);
        brandCombo.getSelectionModel().select(brands.size()-1);
    }
```

```
    @FXML
    public void deleteCar(ActionEvent actionEvent) {
        Car toDelete =
    (Car) listCars.getSelectionModel().getSelectedItem();
        repository.deleteCar(toDelete.getId());
        updateListsView();
    }

        @FXML
    public void addNewCar(ActionEvent actionEvent) {
        Stage newWindow = new Stage();
        FXMLLoader loader =
            new FXMLLoader(CarsShop.class.getResource(
                        "add-car-form.fxml"
                )
        );
        Parent root = null;
        try {
            root = loader.load();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        newWindow.setTitle("Зареєструвати авто");
        newWindow.setScene(new Scene(root, 250, 220));
        AddCarController secondController =
                            loader.getController();
        secondController.set_repository(repository);
        secondController.set_mainController(this);
        newWindow.show();
    }

    public void filterByBrand(ActionEvent actionEvent) {
        String brand =
(String)brandCombo.getSelectionModel().getSelectedItem();
        List<Car> cars = null;
        if("all".equals(brand)) {
            cars = repository.getAll();
        }else{
            cars = repository.getAllByBrand(brand);
        }
        ObservableList<Car> carsList =
                    FXCollections.observableList(cars);
        listCars.setItems(carsList);
    }
    public void editCar(ActionEvent actionEvent) {
        // TODO
        // редагування вибраного зі списку елемента
        // пропонується розробити самостійно
        // за прикладом створення вікна для додавання
        // нового елемента в список
    }
}
```

Код обробника для кнопки додавання нового авто в список містить створення додаткового вікна. Для коректної роботи поданого вище коду слід розробити форму відповідного вигляду за допомогою Scene Builder-a. FXML-файл можна додати до проєкту з контекстного меню папки з ресурсами, він повинен розміщуватися у підпапці, що відповідає пакету класу програми. Контролер можна додати у відповідний пакет як звичайний Java-клас і зв'язати з формою в FXML-розмітці, відповідним атрибутом, наприклад
`fx:controller="com.example.carsdb.AddCarController"`



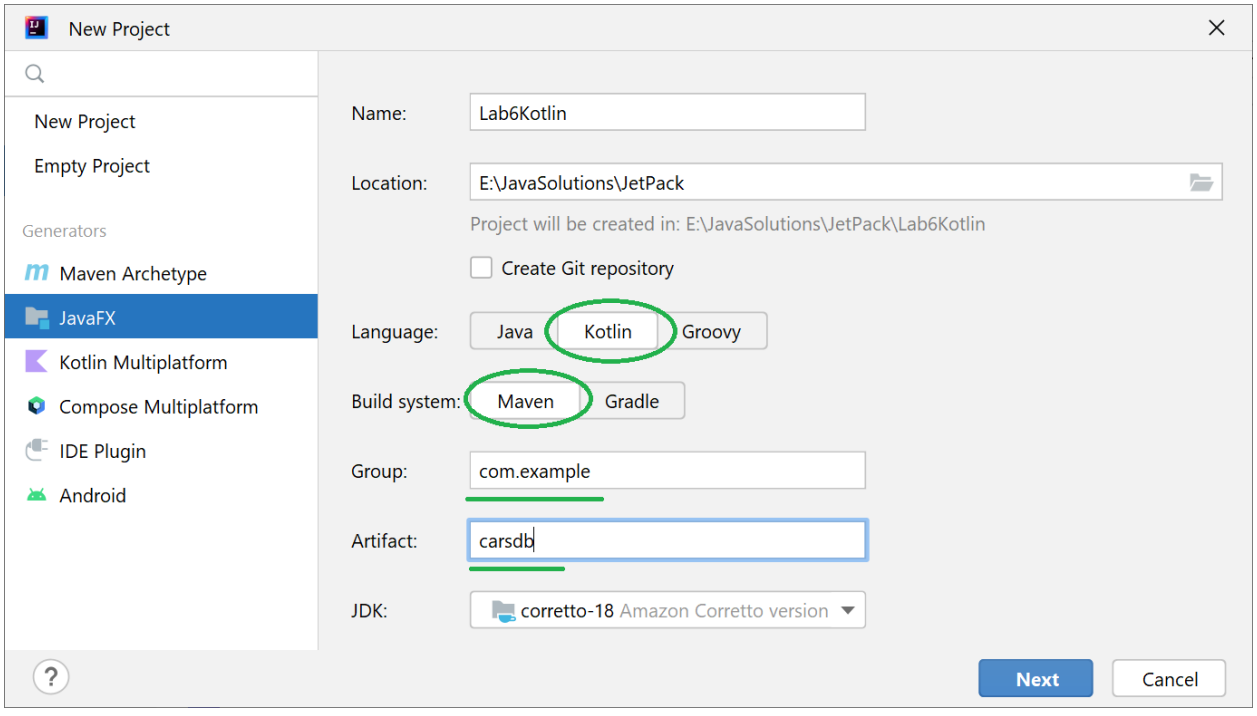
Орієнтовний код контролера форми додавання нового авта в список:

```
package com.example.carsdb;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import com.example.carsdb.data.Car;
import com.example.carsdb.data.DataBaseRepository;
import com.example.carsdb.data.Repository;
public class AddCarController {
    private Repository _repository;
    private MainController _mainController;
    public void set_mainController(MainController
                                   _mainController) {
        this._mainController = _mainController;
    }
    public void set_repository(Repository _repository) {
        this._repository = _repository;
    }

    @FXML
    TextField brand;
    @FXML
    TextField model;
```

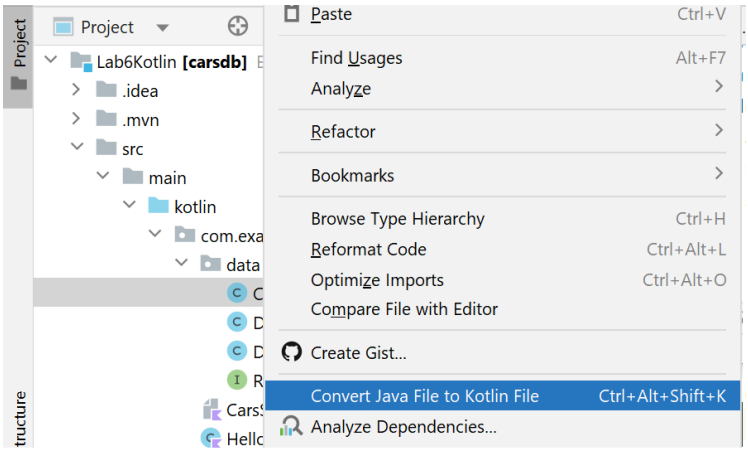
```
    @FXML
    TextField year;
    @FXML
    TextField engine;
    @FXML
    TextField price;
    public void addCarToFile(ActionEvent actionEvent) {
        String brand_ = brand.getText();
        String model_ = model.getText();
        String year_ = year.getText();
        String engine_ = engine.getText();
        String price_ = price.getText();
        Car newCar = new Car(brand_,model_,
         Integer.parseInt(year_), Integer.parseInt(engine_),
                 Double.parseDouble(price_) );
        _repository.addCar(newCar);
        final Node source = (Node) actionEvent.getSource();
        final Stage stage =
                (Stage) source.getScene().getWindow();
        _mainController.updateListsView();
        stage.close();
    }
}
```

Частина 4 (необов'язкова). Kotlin with JavaFX (quick start).

Створюємо JavaFX проєкт з використанням сценарію збирання Maven та мови Kotlin. Назву та групу артифакту бажано задати такими ж, як і проєкті з частини 1, це дозволить використовувати тіж самі пакети для класів з кодом. Перевіряємо роботу згенерованого шаблонного коду. Перейменовуємо файли проєкту аналогічно до проєкту з першої частини роботи і ще раз перевіряємо працездатність коду.



2. Наповняємо проєкт файлами з проєкту з частини 1. Їх можна копіювати, відкривши відповідну папку в провіднику операційної системи. Скопійовані Java-класи перекладаємо за допомогою вбудованої в IntelliJ IDEA утиліти, що викликається з контекстного меню позначки класу у вікні проєкту. При перекладі з Java деякі фрагменти можуть інтерпретуватися неточно, в результуючому коді вони підкреслюються червоним, натиснувши на такому фрагменті комбінацію клавіш Alt+Enter можна отримати підказку щодо способів вирішення проблеми та обрати потрібний варіант.



3. Файли з FXML-розміткою вікон перекладати не потрібно, – обидві мови використовують однакову розмітку. Аналогічно сценарій збирання у файлі pom.xml, сюди слід додати лише залежність для H2. Код головного класу застосунку також можна не заміняти перекладом з Java-проєкту.

4. Утиліта-перекладача дещо надуживає так званими Nullable-типами (позначаються символом "?" після вказаного типу). Так при перекладі інтерфейсу репозиторію типи усіх методів та параметрів перекладаються на Kotlin як Nullable. В той же час в класі-реалізації цього інтерфейсу усі методи перекладені без Nullable. Цю розбіжність слід узгодити. Наприклад, замість отриманого перекладу інтерфейсу у формі

```
interface Repository {
   fun getAll( ) : List<Car?>?
   fun getById(id: Int): Car?
   fun addCar(car: Car): Boolean
   fun updateCar(id: Int, car: Car): Boolean
   fun deleteCar(id: Int): Boolean
}
```

Скористатися варіантом, в якому усі методи відповідають їх реалізації в класі `DataBaseRepository :`

```
interface Repository {
    fun getAll(): List<Car>
```

```
    fun getAllByBrand(brand: String): List<Car>
    fun getById(id: Int): Car
    fun addCar(car: Car): Boolean
    fun updateCar(id: Int, car: Car): Boolean
    fun deleteCar(id: Int): Boolean
}
```

5. Аналогічна ситуація зустрічається в контролері щодо візуальних компонентів, оголошених в FXML-розмітці:

```
@FXM
var listCars: ListView<*>? = null
@FXM
var brandCombo: ComboBox <*>? = null
```

В цьому варіанті їм надається Nullable-тип та присвоюється значення null, після чого FXML-Loader не може помістити туди компоненти, створені на основі розмітки. Цей фрагмент коду можна переписати з використанням специфікатора lateinit:

```
@FXML
lateinit var listCars: ListView<Car>
@FXML
lateinit var brandCombo: ComboBox<String>
```

заодно уточнивши типи елементів списку.
А от щодо методу ініціалізації в класі MainController ситуація протилежна, – в ньому параметр `resourceBundle: ResourceBundle` для коректного запуску програми якраз повинен допускати значення null, тобто тут слід додати «?» після назви типу:

```
override fun initialize(url: URL,
                  resourceBundle: ResourceBundle?) {   ...   }
```

6. Утиліті-перекладачу не вдається перекласти фрагмент Java-коду, в якому вибирається список наявних брендів автомобілів, оскільки в там використано Java StreamAPI, Цей фрагмент можна реалізувати, наприклад, виписавши бренди усіх автомобілів в допоміжну множину (кожен елемент увійде одинираз). Тобто, метод заповнення списків в MainController може бути таким:

```
fun updateListsView() {
    val cars = repository!!.getAll()
    val carsList = FXCollections.observableList(cars)
    listCars!!.setItems(carsList)
    val brands: MutableList<String> = ArrayList()
    val brandsSet: MutableSet<String> = HashSet()
    for (car in cars) {
        brandsSet.add(car.brand!!)
    }
    for ( s in brandsSet) {
        brands.add(s)
    }
    brands.add("all")
    val brndList = FXCollections.observableList(brands)
```

```
        brandCombo!!.setItems(brndList)
        brandCombo!!.selectionModel.select(brands.size - 1)
}
```

Після узгодження усіх згаданих вище «труднощів перекладу» отримаємо робочий застосунок