**Report of endterm task**

**Marketplace of cars**

**Kanay Bekzat, Kaskyrbaev Muhammed, Sagandykov Ramin, Madiyar Zhanzhigitov**

**Overview**

Our project is a marketplace aimed at buying and selling cars. That is, anyone can place their car and other users will be able to see the product with all the information. In general, the project is based on the popular marketplace Kolesa.kz, which has similar functions. For example, placing a product, finding, and changing an already placed product. Based on the existing system, we created our own with additions.

Our project works as a console into which the user enters all commands and information. At the same time, the console is the environment for working with the program. It displays all instructions, menus, information, etc.

The project also has an authorization system, that is, each user can create an account and thus carry out all actions. For this, registration and authorization functions were created that require a login and password. All information is stored in the database and with its help the data is checked. In addition, there are accounts for administrators that are designed to change any data. For example, if the user has forgotten the password, the administrator can change it. Or, when placing a product with invalid data, the administrator can delete it or change it.

To search for cars, users will be able to apply filters to narrow down their search results. Filters might include make and model, price range, mileage, year of manufacture, or location. The application will then display a list of cars that match the selected filters. Users can click on a specific car to see more information, including photos, detailed descriptions, and contact information for the seller.

Overall, the car marketplace project will provide a platform for users to buy and sell cars, with registration and authorization functionality to ensure security, the ability for sellers to publish and update their listings, and a search function that allows buyers to find cars that meet their specific criteria.

**Aim of the project**

The aim of this project is to create a robust and user-friendly car marketplace that allows buyers and sellers to easily connect and transact. The platform will provide users with the ability to register and create a secure account, which they can use to publish cars for sale, update the listings, and search for cars that match their preferences.

The primary goal is to provide a seamless and enjoyable experience for both buyers and sellers, making it easy for them to browse, compare, and purchase cars on the platform. The platform will feature robust search functionality, allowing users to filter by make, model, year, price range, and other criteria, making it easy to find the right car at the right price.

The registration and authorization process will ensure the security of the platform and the integrity of the transactions. Sellers will be able to easily publish cars for sale, including detailed descriptions, photos, and pricing information, and update their listings as needed.

Overall, the aim of this project is to provide a reliable, efficient, and user-friendly marketplace for cars, connecting buyers and sellers in a safe and secure environment.

**Requirements**

1. Registration and Authorization

* The system should allow users to register with the marketplace by providing their personal information, such as name, email address, phone number, and password.
* The system should verify the user's login and phone number through a verification process.
* The system should allow registered users to log in with their login and password.
* The system should have a secure authentication mechanism that protects user data and credentials.

1. Publishing and Changing Cars

* The system should allow registered users to publish their cars for sale on the marketplace.
* Users should be able to add car details such as make, model, year, mileage, price.
* The system should validate the car details before publishing the ad to the marketplace.
* The system should allow users to edit or delete their car ads.

1. Finding Car by Filters

* The system should provide filters to help users search for specific cars on the marketplace.
* Filters should include make, model, year, mileage, price, and location.
* The system should display search results based on the selected filters and display the cars' details.

1. Security

* The system should have a secure authentication mechanism that protects user data and credentials.

1. Performance

* The system should respond quickly to user requests and provide real-time updates on car ads.
* The system should have a high uptime and minimal downtime.

1. Usability

* The system should have an intuitive and user-friendly interface.
* The system should provide clear instructions and guidance for users.

1. Compatibility

* The system should be built using Java technology and be compatible with Java runtime environments.

**Design pattern usage  
Pattern 1:  
 Definition: Mediator** is a behavioral design pattern that reduces coupling between components of a program by making them communicate indirectly, through a special mediator object.

The Authentication class acts as the mediator, coordinating communication between the different components of the system. It has references to the JDBC class, which is responsible for managing the database, and the MenuAdmin and MenuUser classes, which provide the different interfaces for administrators and customers.

The Authentication class encapsulates the logic for user authentication and registration, and delegates tasks to the appropriate classes as necessary. For example, the login() method checks whether the user is an administrator or a customer and calls the appropriate menu method (menuAdmin.menu() or menuUser.menu()).

By using a mediator, the components of the system can communicate with each other in a decoupled way, without needing to have direct knowledge of each other's implementation details.

**Pattern 2:**

**Definition:** **Strategy** is a behavioral design pattern that lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.

In our example, the AutomobileFilter interface defines a family of algorithms (i.e., different filter strategies), which can be used to filter a list of automobiles. Classes such ConditionFilter, BrandFilter, ModelFilter,ReleaseYearFilter, etc. implement one of these strategies, filtering the list of automobiles based on specific characterization. By implementing the AutomobileFilter interface, the ConditionFilter, BrandFilter, ModelFilter,ReleaseYearFilter, etc. classes make its algorithm interchangeable with other algorithms that also implement the AutomobileFilter interface.

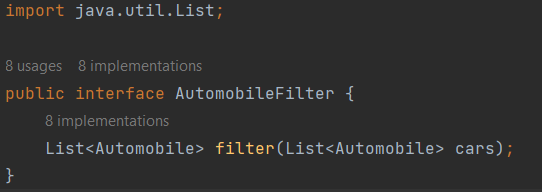
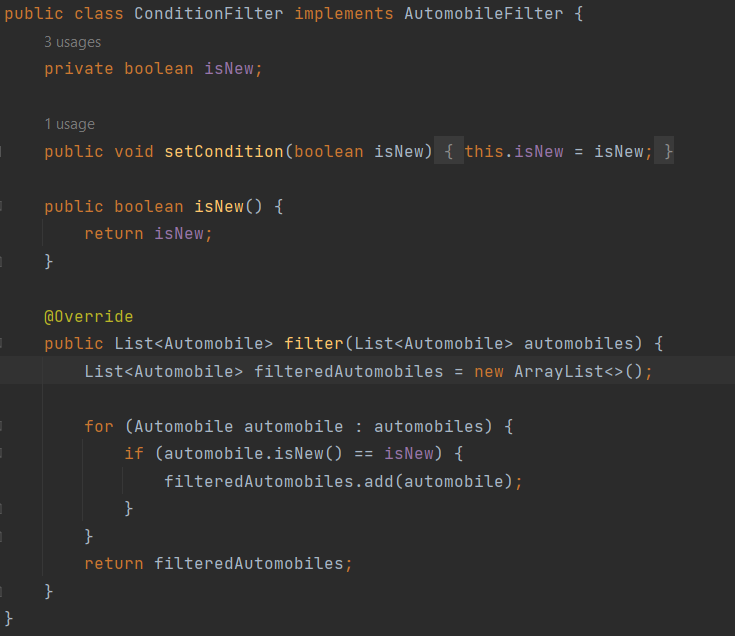
The AutomobileFilter interface acts as the Strategy interface, defining the contract for all strategies that implement it. The ConditionFilter, BrandFilter, ModelFilter,ReleaseYearFilter, etc. classes act as Concrete Strategies, implementing the AutomobileFilter interface and providing a specific filtering algorithm. The client code that uses these classes can switch between different filter strategies at runtime by providing the appropriate strategy object.  
 

Figure 1. Strategy design pattern in code

**Pattern 3:**

**Definition: Singleton** is a creational design pattern, which ensures that only one object of its kind exists and provides a single point of access to it for any other code.  
The Singleton pattern is used when only one instance of a class is required, and it must be available for each class. We have created a JDBCSingleton class. This JDBCSingleton class contains its own private constructor and its own static jdbc instance.  
The constructor must be private so that clients cannot create instances of this class themselves via the new operator. Also, the JDBC Singleton class provides a static getInstance() method to get its static instance into external code. In it, we check whether the object was created by another thread. This method is a global access point to an instance of this class.

**Hierarchy of the project**

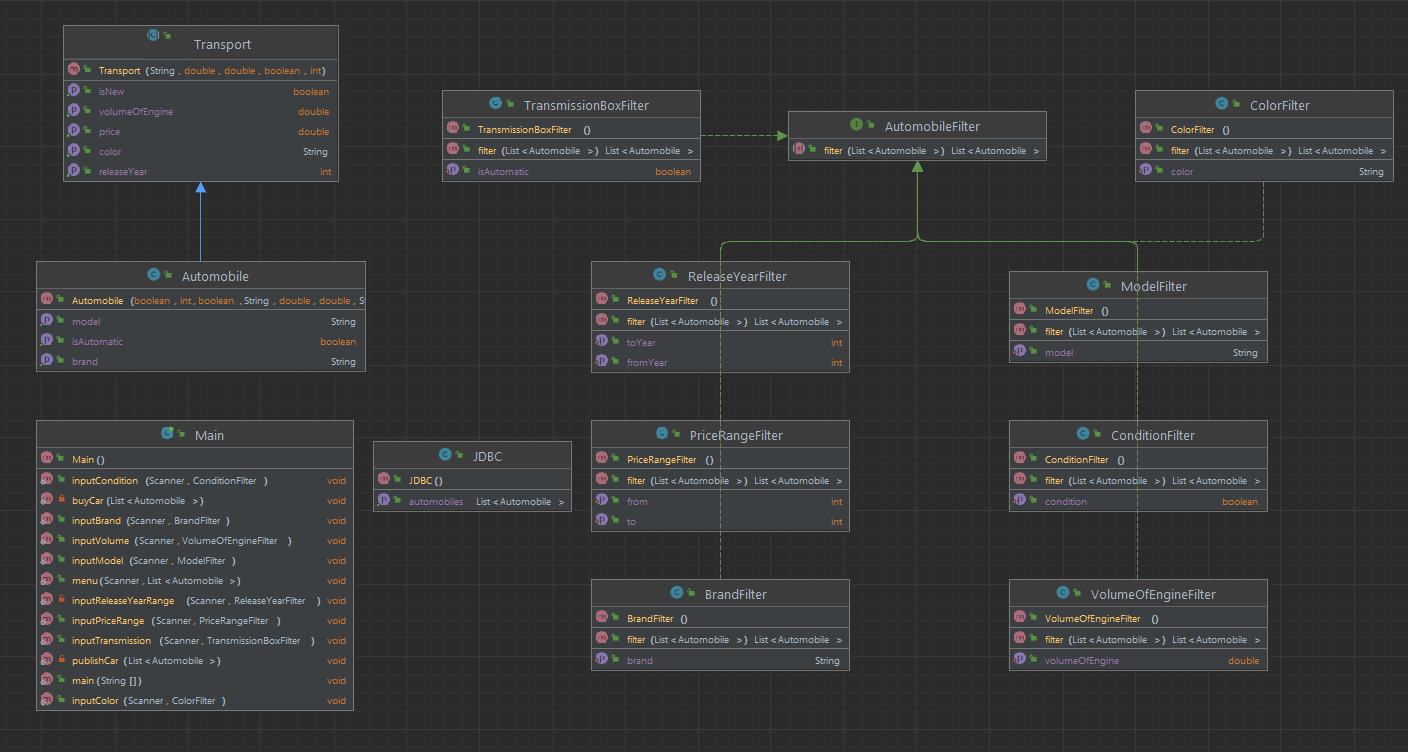


Figure 2. Hierarchy of the project

This is the class hierarchy in our project. The biggest one is car filters, which inherits 8 filters. Then comes the Transport hierarchy and the independent Main and JDBC classes.

**Java Database Connectivity**

To begin with, we created a server in Pg Admin 4. After that, we established a connection in IntelliJ IDEA. We created a separate class JDBC to do actions related to the database.

JDBC class perform the following tasks:

1. Establish a connection to a database server
2. Send SQL statements to the database
3. Retrieve and process the results returned by the database
4. Manage transactions between the database and the application

The getConnection() method establishes a connection to the database using the DriverManager class and returns the connection object. This method is called by other methods in the class that need to interact with the database.

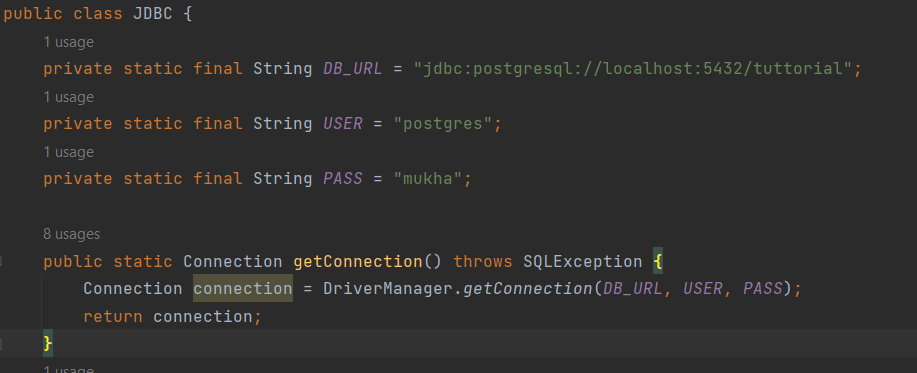


Figure 3. JDBC connection

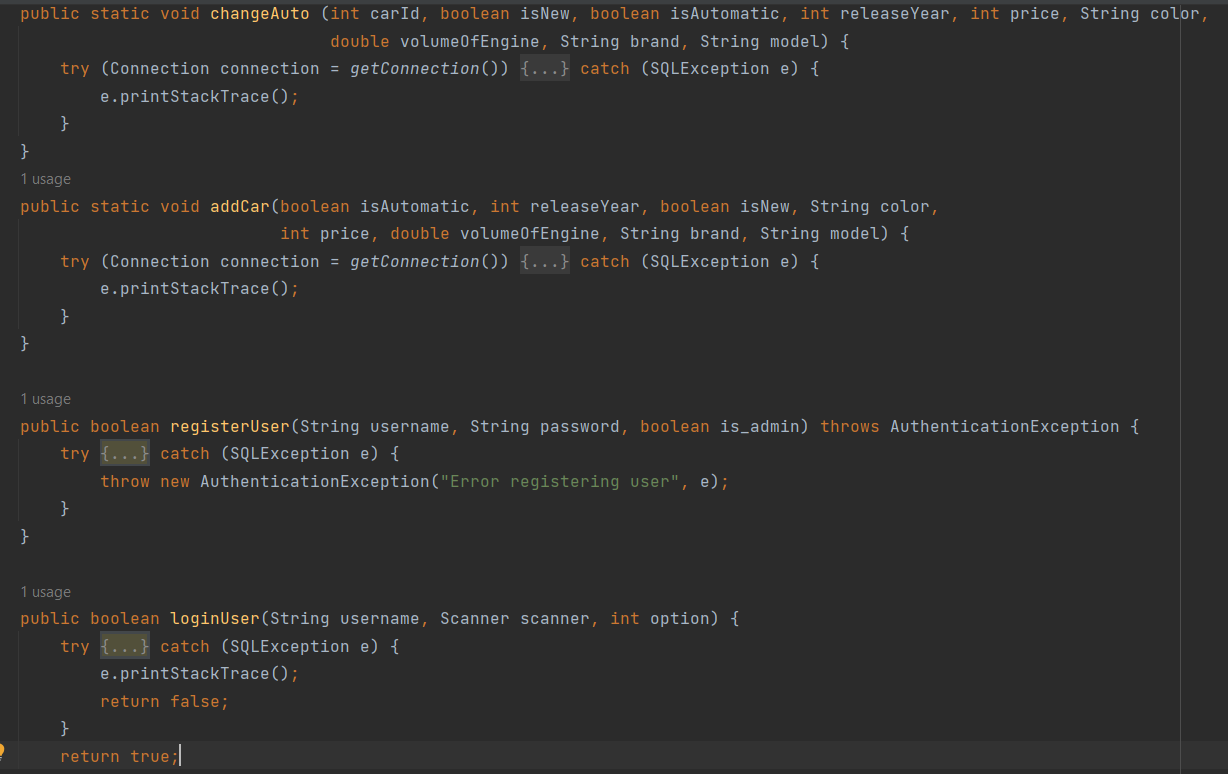
By implementing the JDBC that connects to a PostgreSQL database this class performs various operations such as updating and inserting automobile information, registering users, and logging in users.  


Figure 4. Work with database

For example, the addCar() and changeAuto() methods use prepared statements to execute SQL statements that add or modify automobile information in the database. These methods take parameters representing the values to be added or modified and use these parameters to set the values in the prepared statement before executing it.

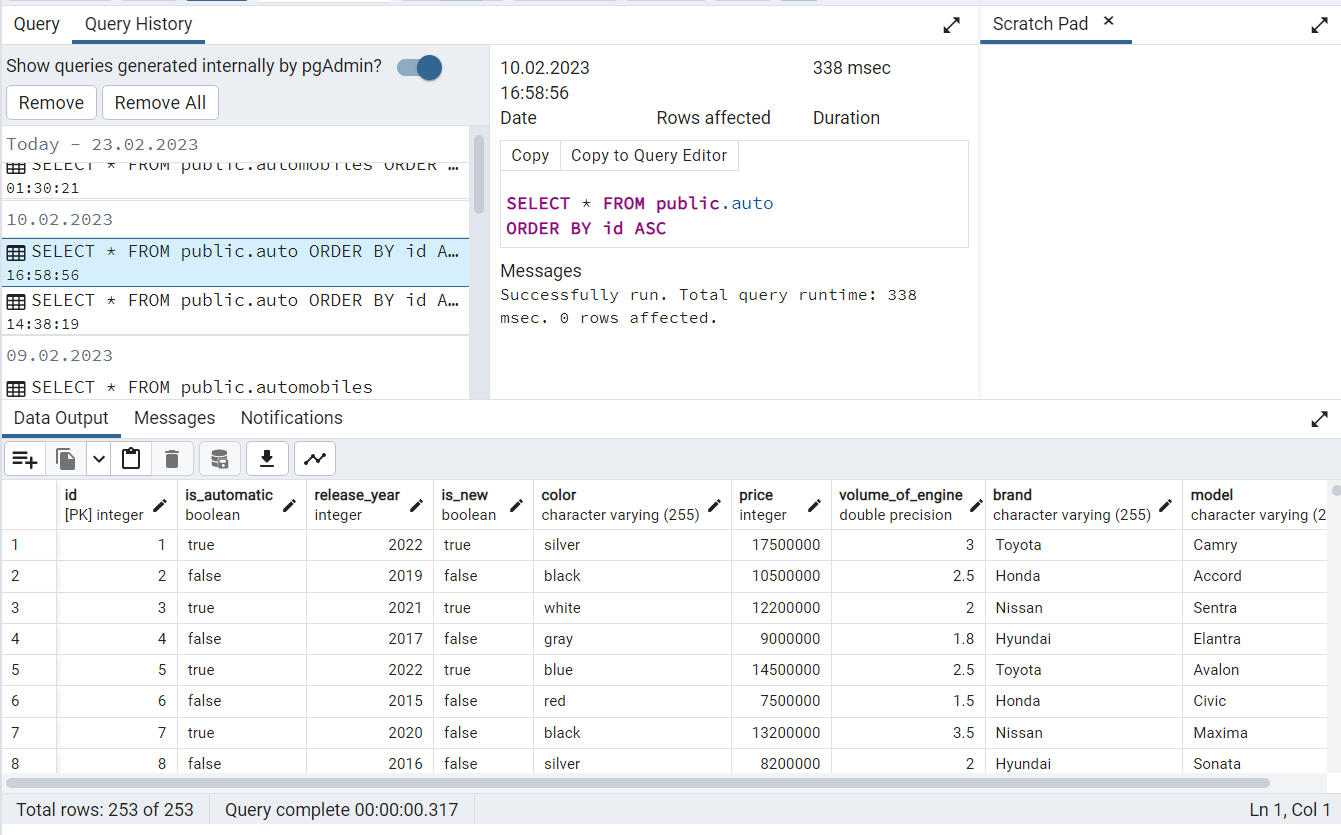


Figure 5. Example of database

From Figure 5 you can see what the database looks like from the developer's point of view. Here you can see all the cars placed on the marketplace, as well as the history of adding. This can be very useful when working with the program.

**References:**

1. Refactoring Guru. (n.d.). Java Design Patterns. Retrieved from <https://refactoring.guru/ru/design-patterns/java>
2. DigitalOcean. (n.d.). Java Design Patterns: Example Tutorial. Retrieved from <https://www.digitalocean.com/community/tutorials/java-design-patterns-example-tutorial>
3. JavaTpoint. (n.d.). Java Design Patterns. Retrieved from https://www.javatpoint.com/design-patterns-in-java#:~:text=Java%20Design%20Patterns,2)%20Structural%20Pattern