SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Faculty of Informatics and Information Technologies

Ilkovičova 2, 842 16 Bratislava 4

**OOP Project – Luxury Cars**

First and Last Name: Marko Stahovec

Date: 11.4.2021

Supervisor: Ing. Pavle Dakić

Contents

[1. Project intent 3](#_Toc69053512)

[2. Class and dependencies diagram 3](#_Toc69053513)

[3. Completed criteria 4](#_Toc69053514)

[3.1 Inheritance 4](#_Toc69053515)

[3.2 Polymorphism 4](#_Toc69053516)

[3.3 Aggregation 4](#_Toc69053517)

[3.4 Encapsulation 4](#_Toc69053518)

[3.5 Other criteria 4](#_Toc69053520)

[3.5.1 DataAccessObject design pattern 4](#_Toc69053521)

[3.5.2 Observer design pattern 5](#_Toc69053522)

[3.5.3 Custom Exception 5](#_Toc69053523)

[3.5.4 Custom Handlers 5](#_Toc69053524)

[3.5.5 Multithreading 6](#_Toc69053525)

[3.5.6 Runtime type identification 6](#_Toc69053526)

[3.5.7 Nested interface 6](#_Toc69053527)

[3.5.8 Lambda expression 7](#_Toc69053528)

[3.5.9 Default method implementation 7](#_Toc69053529)

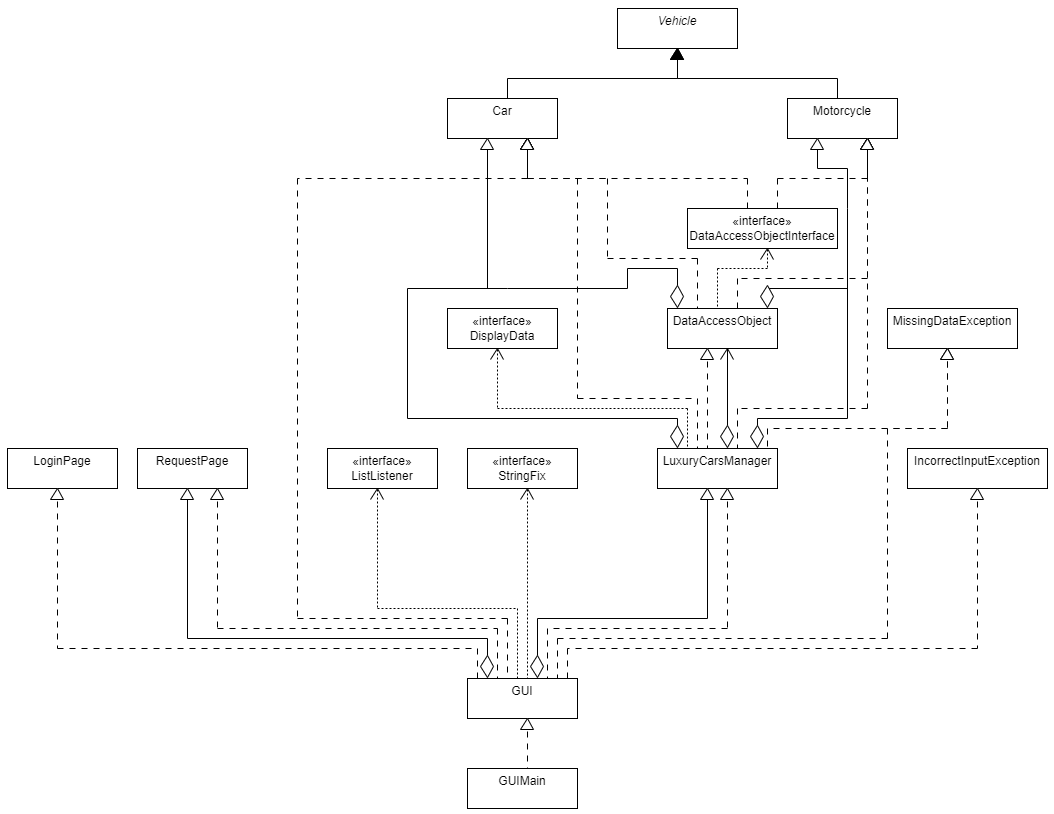
[3.5.10 Serialization 7](#_Toc69053530)

[4. List of most important commits 8](#_Toc69053531)

# Project intent

Production, distribution and sale of cars is a set of processes that require a consistent organization and order. It is a massive system in which a large number of people indirectly work on the same product. The automobile sales are usually realized by a car manufacturing that makes their cars, but there are also companies that deal exclusively by selling or work closely with automakers that supply them with finished products. And therefore, it is necessary to rely on the correct implemented system. Purchase of luxury cars is usually provided for customers to scoop their dreams through different parameters or packages of equipment. In such situations, the dealer must be in direct contact with car manufacturers to ensure maximum customer satisfaction. In exceptional cases, the authorized marks and the possibility of verifying motor vehicles in specific service. It is necessary for certain variability of various brands and an adequate parameter database that describe the cars and the possibility of communicating with the manufacturer for the production of personalized product.

# Class and dependencies diagram



# Completed criteria

## Inheritance

As you can see on the diagram, the inheritance is present when segregating class **Car** and class **Motorcycle** from their superclass called **Vehicle.** They **inherit** all the attributes that class Vehicle possesses.

## Polymorphism

In this project, polymorphism was applied when **constructing** cars, since all the cars in the program were constructed using class Car, but their branding and parameters were all **different**.

## Aggregation

Aggregation was applied directly with classes **LuxuryCarsManager, Motorcycle** and **Car**, because both Motorcycle and Car have unidirectional association with LuxuryCarsManager, while being able to exist individually.

## Encapsulation

## Encapsulation mechanism was used for example on a class **Car**, since its parameters were declared **private** and in the program they are being accessed by special **get** and **set** methods.

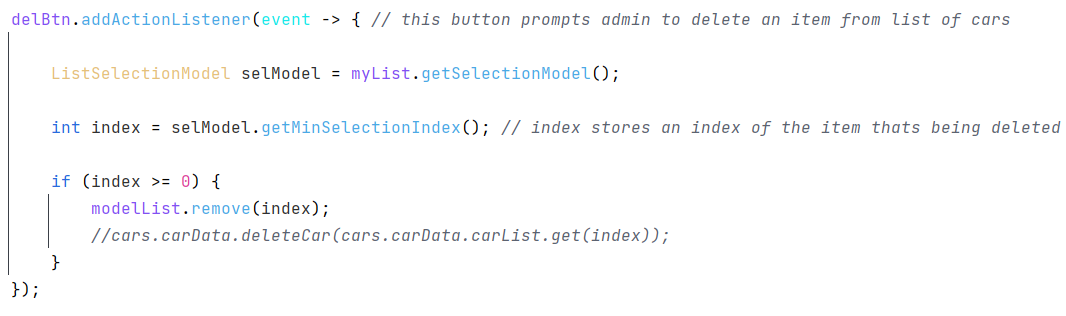
## Other criteria

### DataAccessObject design pattern

DataAccessObject was used as a separate class and interface as a **mechanism to access and delete data** from current database. Their dependencies are shown in the diagram above.

### Observer design pattern

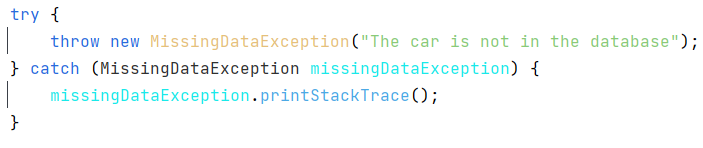
Observer design pattern was indirectly used when creating ActionListeners for specific buttons using method **addActionListener** with lambda expression, such as:



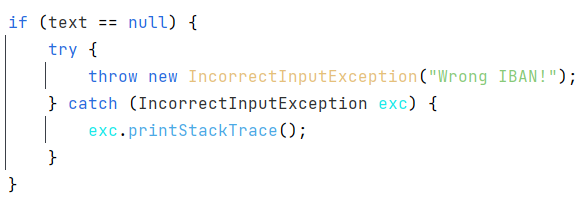
### Custom Exception

There are 2 custom exceptions:

1. **MissingDataException** – is thrown when manipulating with incorrect vehicle object



1. **IncorrectInputException** – is thrown when entering a wrongful input



### Custom Handlers

Every event that occurs in this application is handled manually using **custom** listeners, as can be seen on **criteria 3.5.2**, where is an example handler for deletion button.

### Multithreading

Multithreading feature was implemented within the **request** **section** of the application, where if user requests a vehicle, the application remains **fully** **functional**, while another thread prepares the requested vehicle.



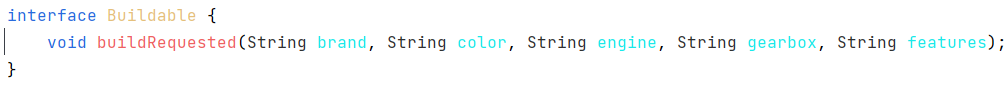
### Runtime type identification

Runtime type identification was used in backend class called **LuxuryCarsManager**, where it served as a runtime identifier for the vehicle data to be displayed.



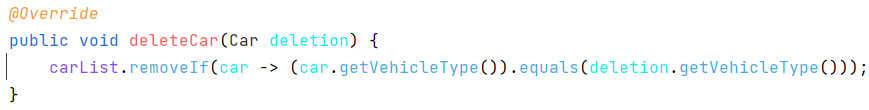
### Nested interface

**Buildable** interface is a nested interface in **DisplayData** interface, where it serves as an addition for class **LuxuryCarsManager**. It is a single-method interface, whomst method builds a requested vehicle and stores it into a car list or motorcycle list accordingly.



### Lambda expression

There are multiple lambda expressions implemented, for example method **deleteCar** in class DataAccessObject operates on a single-line statement using lambda expression in order to delete a car from its arraylist.



### Default method implementation

Default method implementation can be found in **DisplayData** interface, where method called **displayMissingData** defines standard behavior if it is not overriden.



### Serialization

Serialization mechanism is part of a **menubar** on top-left part of the application, where **load** and **save** items are present to ensure simplistic yet effective state **persistence**.

# List of most important commits



Main classes and first GUI draft:

<https://github.com/OOP-FIIT/oop-2021-uto-18-a-dakic-MarkoStahovec/commit/24d7f585ffc2f01e133b9e602a004177a5578f60>

Multithreading using lambda expression:

<https://github.com/OOP-FIIT/oop-2021-uto-18-a-dakic-MarkoStahovec/commit/c363e252ffcac477fe0f2d5839b86cbf54828015>

Serialization:

<https://github.com/OOP-FIIT/oop-2021-uto-18-a-dakic-MarkoStahovec/commit/35b24854bcf562cf68b58a937d9e1ea80102ecec>

Project restructure, icon fixing and database:

<https://github.com/OOP-FIIT/oop-2021-uto-18-a-dakic-MarkoStahovec/commit/74b09df7b4eb0eb647bdccfd20510529bc78839d>

DataAccessObject design pattern addition:

<https://github.com/OOP-FIIT/oop-2021-uto-18-a-dakic-MarkoStahovec/commit/1050a70386a921e7c6375abececf325698db4229>

Icons for main menu:

<https://github.com/OOP-FIIT/oop-2021-uto-18-a-dakic-MarkoStahovec/commit/c676209336d015c87fa91963fbb22bf2f4dfac20>

Motorcycle request:

<https://github.com/OOP-FIIT/oop-2021-uto-18-a-dakic-MarkoStahovec/commit/813c8e1a963349a467e2176200a07a14a96c4094>

# Possible improvements

If I had more time, I would remake serialization mechanism to be automatic, so user wouldn’t have to load current state manually, making the most of serialization itself.