Object-Oriented Application Development: Auto parts catalogue

---

Introduction:

In today's automotive industry, the need for efficient and accessible solutions for purchasing and selling auto parts has become increasingly evident. As car enthusiasts seek to customise and maintain their vehicles, and mechanics strive to provide the best service to their customers, the demand for a comprehensive platform that facilitates these transactions continues to grow.

In response to this demand, we have developed an innovative Auto Parts Catalogue application designed to cater to the needs of car dealerships, mechanics' shops, and automotive enthusiasts alike. This application serves as a centralised hub where buyers can easily search for the parts they need, access detailed information about available inventory, and locate nearby sellers. Additionally, sellers can effortlessly manage their inventory, update product listings, and reach potential buyers through this platform.

By leveraging modern technology and user-friendly interfaces, our Auto Parts Catalogue application aims to revolutionise the way auto parts are bought and sold. Whether you're in search of a specific component to enhance your vehicle's performance or looking to showcase your inventory to a wider audience, our platform provides the tools and resources needed to facilitate seamless transactions and enhance the overall automotive experience.

With its intuitive features and comprehensive functionalities, our Auto Parts Catalogue application is poised to become the go-to destination for automotive enthusiasts, mechanics, and sellers alike. Join us as we embark on this journey to transform the automotive industry and redefine the way auto parts are discovered, bought, and sold.

For further testing and validation purpose, entire code base can be found here :
https://github.com/Shinzuu/AutoPartsHub

Scenario:

Adding Auto Parts to the Catalogue:

- Sellers can easily add new auto parts to the catalogue by providing relevant information such as part name, description, price, and availability.
- The application should support the uploading of images to accompany product listings, allowing sellers to showcase their inventory effectively.

2. Searching for Specific Parts:
    - Buyers can search for specific auto parts using various search criteria such as part name, category, brand, or compatibility with specific vehicle models.
    - The search functionality should provide accurate and relevant results, enabling users to quickly find the parts they need.
3. Viewing All Available Parts:
    - Users have the option to browse through all available auto parts in the catalogue, either by browsing through categories or viewing a comprehensive list of inventory.
    - Each auto part listing should display essential information such as part name, description, price, availability, and seller details.

Major Functionalities:

1. Managing Inventory:
    - Sellers can easily manage their inventory within the application, including adding new parts, updating existing listings, and removing sold items.
    - The application should provide tools for sellers to track inventory levels, monitor product availability, and receive notifications for low-stock items.
2. Displaying Information to Users:
    - The application should present detailed information about each auto part, including specifications, compatibility with different vehicle models, and any relevant technical details.
    - Users should have access to comprehensive product listings that showcase images, descriptions, pricing information, and seller contact details.

By incorporating these essential functionalities into our Auto Parts Catalogue application, we aim to provide users with a seamless and efficient platform for buying and selling auto parts. Whether you're a seller looking to showcase your inventory or a buyer in search of specific components, our application offers the tools and resources needed to facilitate successful transactions within the automotive industry.

Design:

Our Auto Parts Catalogue application is designed to efficiently manage auto parts inventory and provide seamless interactions for both administrators and normal users. The design of the application is structured around a modular architecture, promoting code reusability, scalability, and maintainability.

Classes Involved:

1. Main:
    - The Main class serves as the entry point for the application, controlling the overall execution flow.
2. Menu:
    - The Menu class manages the main menu interface, allowing users to navigate through different functionalities of the application.
3. AdminPanel:
    - AdminPanel class provides administrative functionalities for authorised users, such as accessing admin-specific features and managing inventory.
4. LocationFinder:
    - LocationFinder class facilitates locating parts or dealerships based on user input or predefined criteria.
5. Parts:
    - The Parts class serves as the parent class for all auto parts listed in the catalogue, encapsulating common attributes and methods shared by all types of auto parts.
6. AeroKit, RearWing, Wheels, Engine, Turbo, Ecu:
    - Specific types of auto parts, each inheriting from the Parts class and adding unique attributes and behaviours.

By organising our application into these classes and leveraging inheritance where appropriate, we ensure a modular and extensible design. This design approach promotes code reuse, scalability, and maintainability, laying a solid foundation for the development of our Auto Parts Catalogue application.

Explanation:

In our Auto Parts Catalogue application, the code has been structured to adhere to object-oriented principles and promote modularity, reusability, and maintainability. Below is a detailed explanation of the code, discussing the rationale behind design decisions, addressing encountered problems, and highlighting the usage of inheritance, overloading, overriding, and polymorphism:

1. **Design Rationale:**
    - The application is designed with a modular architecture to separate concerns and promote code organization. Each class is responsible for a specific functionality, making the codebase easier to understand and maintain.
    - Inheritance is utilized to model the relationship between generic auto parts (represented by the Parts class) and specific types of auto parts (such as AeroKit, RearWing, etc.). This allows for code reuse and promotes extensibility, as new types of auto parts can be easily added without modifying existing code.
2. **Encountered Problems and Solutions:**
    - During development, one challenge encountered was efficiently managing the inventory of auto parts and ensuring seamless interactions between buyers and sellers. This was addressed by implementing a centralized Parts class to represent generic auto parts and utilizing child classes for specific types of parts, allowing for better organization and management of inventory.
    - Another challenge was implementing user authentication for administrative functions. This was resolved by creating separate AdminPanel class to handle administrative functionalities.
3. **Usage of Inheritance, Overloading, Overriding, and Polymorphism:**
    - **Inheritance:** Inheritance is used to model the relationship between the Parts class (parent) and its subclasses (AeroKit, RearWing, etc.). This enables the child classes to inherit common attributes and methods from the parent class while adding their own unique functionalities.
    - **Overloading:** Overloading is utilized in certain methods to provide multiple implementations with different parameter lists. For example, methods in the Parts class may be overloaded to handle different types of parameters, such as integers or strings.
    - **Overriding:** Overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. This allows for customization of behavior in child classes while maintaining the same method signature.
    - **Polymorphism:** Polymorphism is demonstrated through method overriding, where the same method name is used across different subclasses with varying implementations. This allows for dynamic dispatch, where the appropriate method implementation is selected at runtime based on the actual object type.

Test:

Search Method:

```
==================================
        Menu
==================================
'-->Navigate by inserting index<--'
1. Browse parts
2. Find nearest dealership
3. Contact
4. Admin login
==================================
Insert number between 1 to 4:  1
----------------------------------
          ==================================
              Browsing Window
          ==================================
          '-->Navigate by inserting index<--'
          1. Browse all
          2. Browse by type
          3. Browse by brand
          4. Search by name
          5. Go back to menu
          ==================================
          Insert number between 1 to 5:  4

          ----------------------------------
          Input parts name(must be 1:1): R33 GTR
          ----------------------------------
                    .                          ->
                    |      AeroKit
                    +---------------------------->
                    +---------------------------->
                    | - ID: 1032
                    | - Name: R33 GTR
                    | - Type: Aero Kit
                    | - Brand: Nissan
                    | - Price: 2100$
                    | - Color available: Cyan
                    | - Available parts: 5
                    +---------------------------->
                Instert 1 to find the Nearest dealer
                    2 to go back to menu
                Insert here: 2
```

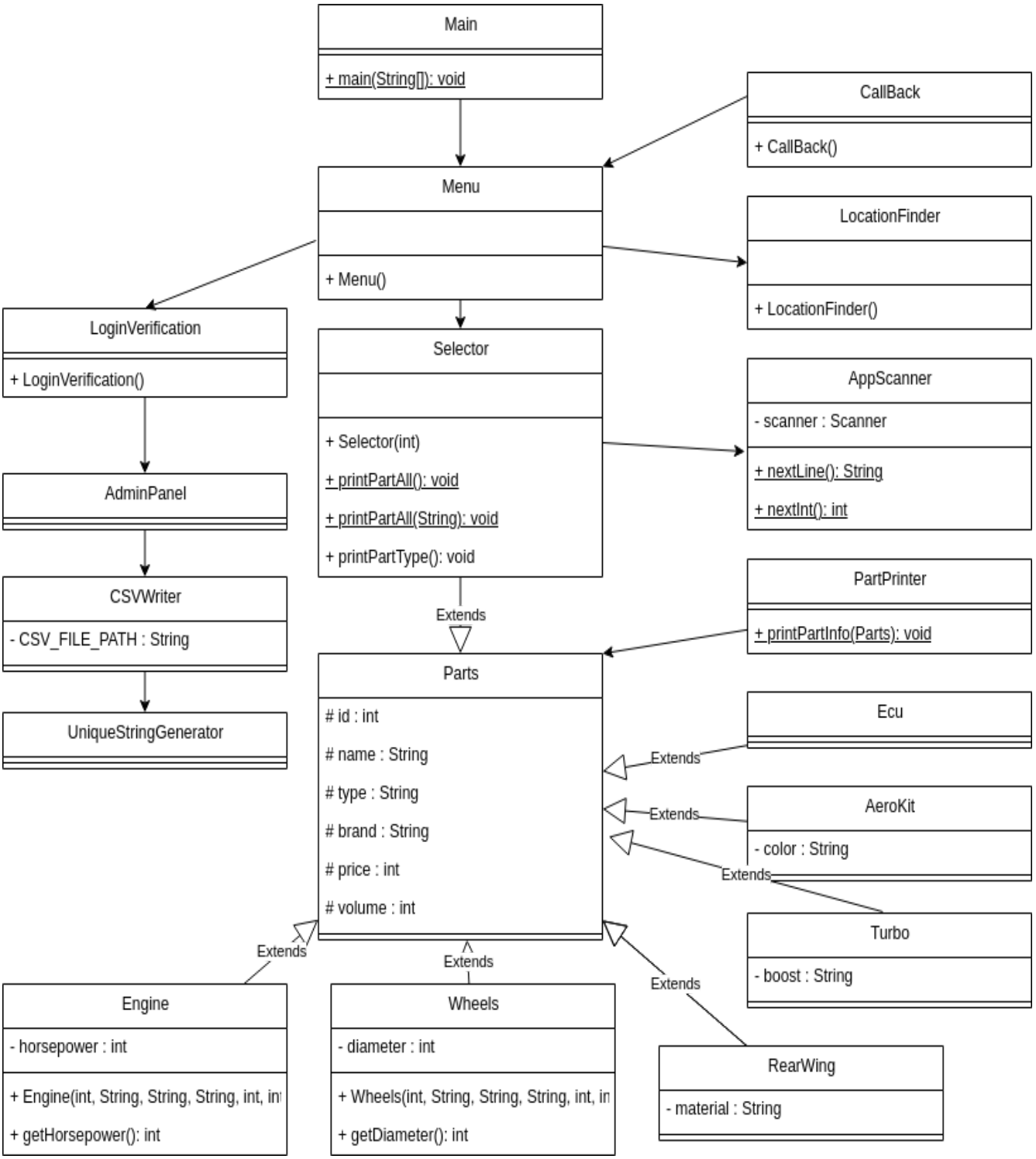
Conclusion:

Upon completing the Auto Parts Catalogue project, it's important to reflect on the process and outcomes. Here's a summary of the project's findings and areas for improvement:

1. **Key Findings and Achievements:**
    - The Auto Parts Catalogue application was successfully developed to address the need for a centralised platform for managing auto parts inventory and facilitating transactions within the automotive industry.
    - The modular architecture and adherence to object-oriented design principles ensured a scalable and maintainable application, laying a strong foundation for future enhancements.
    - Implementation of features such as user authentication, inventory management, and location-based services showcased proficiency in applying advanced programming concepts effectively.
2. **Reflection on the Design and Implementation Process:**
    - The design and implementation process involved meticulous planning and iterative development to meet project requirements effectively.
    - Collaborative teamwork and effective communication played a crucial role in achieving project milestones and overcoming challenges encountered during development.
    - Adhering to software engineering best practices, such as modularization and abstraction, contributed to the clarity and maintainability of the codebase, facilitating future updates and enhancements.
3. **Lessons Learned and Areas for Improvement:**
    - Throughout the project, valuable lessons were learned about the importance of thorough planning and effective communication in ensuring project success.
    - While the development process was executed well, there may be opportunities to further optimise certain functionalities or enhance user experience based on user feedback.
    - Continuous improvement and refinement are essential for future iterations of the Auto Parts Catalogue application, with a focus on addressing user needs and staying abreast of industry trends.

In conclusion, the Auto Parts Catalogue project represents a significant achievement in leveraging technology to streamline processes within the automotive industry. Moving forward, the insights gained from this project will inform future endeavours, guiding the development of innovative solutions to address evolving industry needs effectively.

Diagram:


Main
+ main(String[]): void
CallBack
+ CallBack()
Menu
+ Menu()
LocationFinder
+ LocationFinder()
LoginVerification
+ LoginVerification()
Selector
+ Selector(int)
+ printPartAll(): void
+ printPartAll(String): void
+ printPartType(): void
AppScanner
- scanner : Scanner
+ nextLine(): String
+ nextInt(): int
AdminPanel
CSVWriter
- CSV_FILE_PATH : String
UniqueStringGenerator
PartPrinter
+ printPartInfo(Parts): void
Extends
Parts
id : int
name : String
type : String
brand : String
price : int
volume : int
Ecu
Extends
AeroKit
- color : String
Extends
Turbo
- boost : String
Extends
Engine
- horsepower : int
+ Engine(int, String, String, String, int, in
+ getHorsepower(): int
Extends
Wheels
- diameter : int
+ Wheels(int, String, String, String, int, in
+ getDiameter(): int
Extends
RearWing
- material : String