**24LLP109 – Digital Application Development**

**Car Rental System**

Sohag Noman – F434565

**Table of content**

[**1. Introduction…………………………………………………….………3**](#Introduction)

[**2. Program Structure…………………………………………………….3**](#PS)

[**2.1 Requirements…………………………………………..…….4**](#UR)

[**2.2 Diagram…………………………………………………….….5**](#BD)

[**3. Application Development……………………………..……………..6**](#AP)

[**3.1 Classes………………………………………………….……..6**](#Class)

[**3.2 Dictionaries…………………………………………..……….7**](#Method)

[**3.3 Methods………………………………………………..………7**](#Method)

[**3.4 For Loop…………………………………………………..……7**](#FOR)

[**3.5 Conditional Statement……………………………….………8**](#Conditions)

[**4. Program functionality…………………………………………….…..11**](#PF)

[**5. Conclusion ……………………………………………………………..13**](#Con)

**Introduction**

The car rental system developed in this project demonstrates the application of Object-Oriented Programming (OOP) principles to automate and simplify vehicle rental processes.

Object-oriented programming is a way to structure code via objects, where these can interact with each other. In this coursework we move away from procedural programming where the program structure follows a linear sequence of instruction and implement OOP.

The system is designed to manage car rentals effectively by tracking stock availability, rental transactions, and customer billing. It supports a couple of features such as renting cars for different durations, handling returns and providing specialized rates for VIP customers.

**The program is intended to**:

* Automate the process of renting and returning cars.
* Show the inventory for different vehicle types (hatchback, sedan, SUV).
* Calculate and display rental costs based on rental duration.
* Issue bills upon return

**Industrial Application**   
In the auto rental market, companies rely on software solutions to manage operations effectively. This program can be deployed by family owned or very small rental shops looking for a straightforward system to:   
- Manage vehicle stock in real time.   
- Handle transactions with basic customer contact.   
- Provide unique prices for high-priority consumers.

A similar software application can be utilised in various businesses where the notion of ‘booking’ systems is implemented, because it may follow the same structure of displaying, booking, and returning.  
  
**Relation to Digital Technologies**   
This program aims to show how digital technologies can automate business processes, enabling companies to simplify customer experiences and streamline workflows. Concepts like dynamic pricing (depending on length of the rental) and client distinction (VIP prices) connect with digital transformation trends in various businesses.

**Program Structure**

The car rental system is organized into three main classes:

1. **RentalShop**: Manages vehicle inventory, rental rates, and transactions.
2. **Customer**: Represents general customers and tracks their rental transactions.
3. **VIPCustomer**: Extends Customer and applies specialized pricing for loyalty benefits.

The program was developed on VS Code using Python3 and it is divided in 2 Python files. (rentalSys.py & main.py) It runs on VS code terminal, allowing users to interact through the terminal procedurally with the shop by viewing stock, renting cars, and returning them.

**User and Functional Requirements for the Rental System**

User requirements describe what the users expect the system to do. These are high-level goals based on user needs. The key user requirements include:

A black and white sign with white text

Description automatically generated

Functional requirements describe specific behaviours or functions the system must perform to meet user requirements. For the car rental system, these include:

A black and white list of information

Description automatically generated

*Requirements are a useful tool in programming or software development as it allows the developer to:*

**A black table with white text

Description automatically generated**

**Without Requirements** a developers might assume features like multi-car rentals or advanced GUIs are necessary, leading to wasted time and energy.

**With Requirements** the developers focus on the defined scope (e.g., managing inventory and calculating rental costs), ensuring efficient development.

**This procedure provides a systematic foundation for building software, ensuring both technical quality and user experience.**

**Block diagram and Class table to explain the structure:**

A diagram of a customer

Description automatically generated

The car rental system is organized into three main classes:

1. **RentalShop**: Manages vehicle inventory, rental rates, and transactions.
2. **Customer**: Represents general customers and tracks their rental transactions.
3. **VIPCustomer**: Extends Customer and applies specialized pricing for loyalty benefits.

A black and white chart with text

Description automatically generated

**Application Development**

For this coursework 2 files with the Python extension were created. The first called rentalSys.py which contains all the classes and their respective methods, and the second one called main.py which executes the Car Rental System program.

The **RentalShop** is the first class in our rentalSys.py file, and the keyword ‘class’ is used to define the class.

A computer screen shot of a program

Description automatically generated

In Python, a class is a user-defined blueprint from which objects are made. When we create a class, we are making a new type of object. If, for example, there was a class River, we could create instances of the class as Volga, Seine, and Nile.) This object is a container for variables and functions. Variables in an object are called attributes and functions are called methods. (In the River class example, all the instances will have the same structure and share all class attributes defined within the class River.) ***We use classes and objects because they allow us to organise complex code, as well as reusable code.***

The RentalShop class implements the \_\_init\_\_ method or constructor, which, in Python, is used to initialize objects of a class. This method is called by python when the object is created. The task of this \_\_init\_\_ is to initialize (assign values) to the attributes of the class when an object of the class is created. In python the class itself does not have any attribute, we only create the attribute when we define the init method. The first parameter ‘self’ is a reference to the class itself. The "self" parameter is used to pass the object of the class as the first argument whenever we call a method of an object that was made from a class.

The ‘self.stock’ and ‘self.prices’ refer to the attributes that the rental shop has. They are both stored in dictionaries {}. Dictionaries are a very useful data type built in Python. They can also be referred as ‘associative arrays’ because they store data as a set of “ *key:value”* pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}. The ‘stock’ dictionary refers to the initial inventory of cars with their respective amount, while the ‘prices’ dictionary is a nested dictionary. This means it contains inside itself two more dictionaries with ‘daily’ and ‘weekly’ prices, with the key being the type of car inside quotation marks and the value being a data type, in this case an integer. The purpose of using a dictionary to enable the pricing logic is mainly because :

* Can retrieve prices based on conditions (Based on car type, duration)
* Allows modification. If pricing changes, we only need to update the dictionary.
* Allows simpleness, enhancing readability for any user.
* It’s an iterable data type. It has a collection of objects that can be iterated over with the help of a for-loop

A screen shot of a computer code

Description automatically generated

‘Def’ is a Python keyword that needs to be called when we want to create a method. After the ‘def’ keyword, we name our method **display\_inventory().** The parenthesis ‘()’, also called argument, is the most important part within the function that contains the parameters we pass into it. Since methods always need at least one parameter, conventionally in Python, it is the reference to the class itself. The ‘self’ parameter refers to the current instance of the RentalShop class, allowing access to instance variables like self.stock and self.prices.

The purpose of this method is to:

* Display car stock in our RentalShop
* Show the prices for car type and rates
* Provide information about VIP customer rates

The print() function prints a message that shows “Available Cars and Prices:”

**For** is a keyword that can be used to iterate over a sequence. In this case the for loop does the following actions:

* Create a variable car\_type inside the loop that will store the key from the dictionary.
* Count is a built-in string function that returns the number of occurrences of a string type element appearing inside self.stock.
* Items() is a built-in function for dictionaries that returns an iterable view of key-value pairs.

The for loop automatically unpacks each key-value pair from items() and assign them into the variables car\_type and count.

The rest are print() functions that print the messages inside the parenthesis.

A computer screen with text

Description automatically generated

This method is named **rent\_car().**The purpose of this method is to handle the logic of renting a car to a customer.

* The first parameter is obviously ‘self’ refers to the instance of ‘RentalShop’ class.
* The second parameter ‘car\_type’ , is a string type variable that refers to the type of car the customer wants to rent.
* The third parameter ‘days’ , is an integer type variable that refers to the number of days the customer wants to rent.

Inside the method there is a conditional statement with 2 ‘if’ keyword that checks 2 things simultaneously:

* Check if the requested car type exists in stock: self.stock . If it does not exist, it prints ‘Invalid car type’ and exits the function returning none. (In Python ‘None’ is a keyword to say “there is nothing meaningful to return, so I return nothing”)
* Check stock availability: Checks the current stock for the requested car type. If stock is 0, it prints a message that states the type of car is out of stock and exits the function returning none.

Rental rate and pricing logic:

If the first 2 conditions are valid, that means the requested car type is valid and it’s in stock, so the function reduces stock of the car type by 1. Then, there is another conditional expression ("weekly" if days >= 7 else "daily") to decide which rate to apply.

* "weekly": If the rental duration is 7 days or more.
* "daily": If the rental duration is less than 7 days.

**‘Rate = self.prices**’ Retrieves the corresponding rate for the car\_type from the self.prices dictionary. When the conditional expression is evaluated, it picks either ‘weekly’ or ‘daily’ key from the self.prices dictionary.

Conditional statement in Python is an essential part of programming in Python that allow us to provide logic within our program. They allow us to make decisions based on the values of variables or in this case, input of the user. After selecting the correct rate type, it retrieves the rate of the specific car type.

A computer screen with text

Description automatically generated

The purpose of the return\_car method is to handle the customer returning a rented car. It updates the inventory, calculates the total rental cost, and displays the rental details and the total bill to the customer.

A computer screen with text and images

Description automatically generated A computer screen with text on it

Description automatically generated A computer screen with white text

Description automatically generated

The **Customer** and **VIPCustomer**: Interact with the shop to inquire about stock (inquire\_stock), rent cars (rent\_car), and return cars (return\_car).

A computer screen shot of a black screen

Description automatically generated

The VIPCustomer class is declared as a subclass of the Customer class using Python's ***inheritance.***   
The VIPCustomer class specialize the Customer class by adding a new feature: VIP price for car rentals / or discounted rates.  
Inheritance eliminates duplicating code from the Customer class. All properties and methods of Customer are available to VIPCustomer.

Main is a special function in Python, and it acts as the starting point of execution for any program. When we run the main.py in the terminal, we can execute whatever is inside that file. Since our classes and methods are inside the rentalSys.py file, we can import our rentalSys.py file as a module with the respective classes.



A module in Python is just a file consisting of Python code, and any Python file can be referenced as a module.

A black background with colorful text

Description automatically generated

The main function serves as the entry point for the car rental program and initializes the system.

shop = RentalShop() creates an instance of the RentalShop class

A screen shot of a computer code

Description automatically generated

The following actions are executed here :

* Greet user and ask their name
* Determine customer type (VIP customer object has special rates)

A computer screen shot of a program code

Description automatically generated

* A while loop allows the menu to be displayed, until user exits.
* If-else conditional statement handles the available options

**User Interaction**:

The customer interacts procedurally through the command line. Their input prompts request the users to perform specific actions like viewing stock, renting or returning cars.

**Program functionality**

To test that the program works, we can go into terminal and select the directory where the program is stored. In this case, Test folder.

A screen shot of a computer

Description automatically generated

Inside the folder we can call the main file that will launch our program, asking user to prompt their name.

A black background with white text

Description automatically generated

Once inserted the name, the program ask to the user what type of customer they would like to become.

A screen shot of a computer

Description automatically generated

If ‘regular’ is selected, the program displays all the available options. Depending on the user’s choice, they can:

1. View available cars.
2. Rent a car.
3. Return a car.
4. Exit the system.

In this case options 1 shows the available cars and the prices.

A screen shot of a computer

Description automatically generated

If selected 2, the program asks to entre car type and number of days.

When user wants to return the car, they can choose option 3 and the program will issue the bill for the rented car.

A screen shot of a black screen

Description automatically generated

Let’s check the VIP customer rate :

A screenshot of a computer program

Description automatically generated

In this case user has selected to be a VIP customer.

A black background with white text

Description automatically generated

Upon returning the car :

A screen shot of a black screen

Description automatically generated

This reflect the VIP rates implemented inside the VIPCustomer class:

A screen shot of a computer program

Description automatically generated

**Conclusion**

The rental system works, but has limitations. It lacks a database for permanent storage, meaning all data is lost when exited. The system assumes a single customer engages at a time, limiting multi-user environments. The program runs through CLI. Additionally, there’s no mechanism for handling damaged vehicles or late returns, which are common in real-world situations.  
  
Improvements could include integrating a database for inventory and rental history, enhancing input validation, and allowing multi-user support through a web or mobile interface. Adding features for damage handling, late fees, and online payments would further improve usability.