A picture containing text

Description automatically generatedCairo University  
Faculty of Computers and Artificial Intelligent

**CS251**

**Software Engineering I**

Parking Garage application

Software Design

*Represented by:*

|  |  |  |
| --- | --- | --- |
| *Name* | *ID* | *Section Number* |
| Mariam Ashraf Amin Mohamed | 20200520 | S15 |
| Hoda Shafek Ahmed said | 20200619 | S15 |
| Reem Ayman Abdel-Fattah Ibrahim | 20200186 | S15 |
| Mariam Saeid Shawky | 20200521 | S15 |

*June,2022*

Does your class diagram respect or violate SOLID principles? Justify your answer.

Yes, Class diagram respect many SOLID principles Like:

* **single responsibility principle**

we can see it in the following classes

1- Configuration: This class is responsible for configuration as it selects which configuration to follow in the picking slot according to SlotSelector type, it do single operation (set configuration), has one reason for change.

2- TotalIncome: This class is responsible for total income as increase total income for garage by adding all fees, calculate and return it. There is related operation has been done has so it follows single responsibility principle.

3- TimeSetter: It has time in and time out as attribute it set them automatically when vehicle park in and park out. So, there is one operation so has one reason to be changed and achieve single responsibility principle.

4- Slot:It contains all information about slot, dimension -availability -ID, and has method to specify and return all of this so it does related operation and follows single responsibility principle.

5- ParkIn: It picks suitable slot for vehicle according to selected configuration; these two operations are related so that achieve single responsibility principle.

6- VehicleCalculation: It is responsible for calculate number of vehicles that are in grarage (single operation).

7- Vehicle: It contain all vehicle’s information (model name-identification number-model year-dimension) as this class set all this information and return them so this are related operations.

8- Calculation: It calculates parking fees for each vehicle and return it.

* **open closed principle**

we can see it in the following classes

1- IException: it has only two methods for handling output message and there is no need for adding other methods, so it is closed for modification, there are many exceptions for: ID, dimensions, money, and slot, they are subclasses: NoEnoughMoneyException, DimensionExceotion, IdException, NoFitSlotException, and NoFreeSlotException. They all extend from the base. We can’t make change in any parts but if it is important to add other methods, we can do it. How? In our case it can be done by adding subclass for new exception so this will be simple process as it is open for extension, we will just add class for exception and make it extend from base class.

2- SlotSelector: It has one operation to do as it select slot when vehicle come to park in so we don’t need to change it so it achieve closed for modification principle, SlotSelector is base class that has two subclasses extend from it FirstFitSelctor and BestFitSelctor , each of them has its own method to select slot in the way it is designed for, and if there are other ways to add it , we can do that easily by adding subclass for new selector, so that is achieve open for modification principle.

* **interface segregation principle**

1- IParkingGarage – IParkOut – IParkIn: We split these interfaces instead of making them in only one to make each of them depends on method they use so that lead to simpler classes.

2- IVehicleHandling – IvehicleCalculation: Each of them has methods they need and own class to implement it which lead to simpler classes as no class has method doesn’t use so there follows interface segregation principle.

Does your class diagram contain any design pattern(s), if yes name it and list the names of the classes involved in such pattern(s)?

Yes, class diagram has some design pattern.

**1- Strategy pattern (Behavioral)**

* SlotSelector – BestFitSelector – FristFitSelector
* IException – NoEnoughMoneyException – DimensionExceotion – IdException

NoFitSlotException - NoFreeSlotException.

**2-Adapter pattern (structural)**

TmeAdapter – TimeSetter - Calculation: TimeAdapter takes timein and timeout to calculate the time of stay so it makes attributes in TimeSetter suitable to do CalculateParkingFees in Calculation which make use of Adapter pattern.

**3- Singleton pattern (Creational)**

SlotHandling: We need one instance as all objects can see it so we can calculate number of all available slots by decreasing number of available slots when vehicle comes to parkin and increasing it when vehicle park out from garage.

VehicleHandling: We need one instance as all objects can see it so we can calculate number of all vehicles came to garage and parkin, by increasing number of vehicle when vehicle comes to parkin.

Totalincome: To calculate total income we need to make instance which seen by all objects so it has one value for all object so we can increase it by all object when paying fees, so we make use of Singleton pattern, that allow as to calculate total income for garage.