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

Our class diagram **respects** the SOLID principles.

Justification:

**S: Single Responsibility Principle**

Every class in your code have only one job to do:

-GarageMachine class: is the boundary class for the driver and it is responsible of output the messages to the driver on the garage machine screen.

- Machine controller class: is the controller class for the driver and it is responsible for controlling all the functions done for the driver.

- Slots class: is responsible for updating the slots details.

-Slot class: stores the details of every Slot.

- VehicleCard class: Stores the details of the cars parked-in the garage.

- ParkingMang class: Stores the cars that are currently in the garage and responsible for updating the stored cars in it.

- ManagerScreen class: is the boundary class for the manager and it is responsible of output the messages to the manager on the screen.

- ScreenController: is the controller class for the manager and it is responsible for controlling all the functions done for the manager.

## O: Open/Closed Principle

## "Open for extension", we designed the class in such a way that the new functionality can be added only when new requirements are generated.

## "Closed for modification", we developed the class and it has gone through unit testing.

## -In the Park-in method it can park in any method First come first serve or Best fit using the same code so it apply the Open/Closed principle

## I: Interface Segregation Principle

## The classes FirstComeMethod and BestFitMethod implements from the Interface ParkingMethod

**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, we used:

**Singleton In classes:**

* Slots
* ParkingMang
* MachineController
* ScreenController

**Strategy in:**

* ParkingMethod