Cairo University  
Faculty of Computers and Artificial Intelligent

**CS251**

**Software Engineering I**

Parking Garage Application

Use of SOLID Principles and Design Patterns

Aly Eyad Aly SayedAhmed

John Gamil Samir Hakim

Silvana Yacoub GabAllah

Yara Mohamed Saad

May 2022

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

A) The Class Diagram Respects Solid Principles, such as:

* **Single Responsibility Principle (SRP)**, most evident in:
  + **DateLog:** This class handles only responsibilities related to Time Capture of Vehicles, it does not use this time to calculate fees, it only focuses on capturing time.
  + **Dimensions:** This class only only handles storing dimensions of Vehicles and Slots, and Determining if they fit
  + **ParkingSpace:** This class is only responsible for storing ParkingSpace information, it does not handle any parking functions, only sets the vehicle of a parking space (since the vehicle is a private member of this class)
  + **SpotFindAlgorithm:** This interface (and classes that implement it) only perform one function, which is executing some Algorithm to find a fitting spot for a Vehicle.
* **Open Closed Principle (OCP), most evident in:**
  + **SpotFindAlgorithm:** This interface is open for extension and closed for modification, where concrete Algorithms can implement this interface, and all usages of these algorithms are done via the interface itself (coding to an interface rather than an implementation).

**Q) 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).**

A) The class diagram contains multiple design patterns, such as:

* **Singleton:** Used in the **Garage class** to allow only one instance of the garage to exist, and provide only one method of access to that instance, by:
  + Making the Garage Constructor Private
  + Defining a private static Garage Variable to represent the single Instance
  + Defining a public static method getInstance to get that single Garage instance
* **Strategy:** Used in **SpotFindAlgorithm** to define a family of Spot Finding Algorithm, encapsulating them, and allow them to be interchangeable. This usage maps to the structure of the Strategy Pattern as follows:
  + **Host:** ParkInController, the class using the algorithm
  + **Algorithm:** SpotFindAlgorithm
  + **Concrete Algorithms:** FirstComeFirstServe, BestFit
  + **setAlgorithm:** SetFinder
  + **performAlgorithm:** performFind
  + **strategy:** spotFinder
  + **operation:** getSpot