Cairo University  
Faculty of Computers and Artificial Intelligent

**CS251 - Software Engineering I**

Project Name

Software Requirements Specifications (SRS)

Team Names

Month & Year

# Team

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Name** | **Email** | **Mobile** |
| 20200575 | Mirette Shenouda Maher | [miretteshenouda01@gmail.com](mailto:miretteshenouda01@gmail.com) | 01223898053 |
| 20200567 | Monica Saeed Habib | monicasaeed12@gmail.com | 01226489661 |
| 20200686 | Youstina Saadawy Thabet | [youstinasaadawy@gmail.com](mailto:youstinasaadawy@gmail.com) | 01207227565 |
| 20200386 | Catherine Ramy Mikhail | [catherineramy02@gmail.com](mailto:catherineramy02@gmail.com) | 01201700544 |

* **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, our class diagram contains design pattern using Strategy design Patterns**

It features design concepts/principles like:

* Encapsulate what varies, Code to an Interface, Delegation and Favor Delegation over Inheritance.
* we implemented class configuration which has 2 behaviors " best fit - first come " where Client only makes use of the public interface “configuration” of Algorithm and is not tied to concrete subclasses (Best fit and first Come). and can change its behavior by switching among the various concrete algorithms.
* inheritance will help the developer split both functions, so he’s not forced to use both.
* **Does your class diagram respect or violate SOLID principles? Justify your answer.**

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

**Single responsibility principle (SRP):** A class should have one and only one reason to change, meaning that a class should have only one job.

**Examples:**

* Class “GarageCalculations” only do the system calculations such as total income and garage fees.
* Class “MyGarage”: responsible for parkin and parkout functions.
* Class “FirstCome”: will use the first free slot available from the parking garage slots.
* Class “BestFit”: will find the slot with the minimum dimension to hold the vehicle.

**Open/closed principle (OCP):** Objects or entities should be open for extension but closed for modification.

***Inheritance:***

- we created class “configuration” because we have 2 different configurations (BestFit , FirstCome) that inherit from this class to setup the garage.

***Delegation:***

We created and object “myconfig” from “configuration” class and sent it to class “MyGarage” so that any changes in class configuration will not affect the created garage.

Also, we created an object “calculations” from “GarageCalculations “and we sent it to “MyGarage” class. So, if any changes happened to the function “calcFees” or we added any other functions in this class it will be easily handled in class “MyGarage”

**Interface segregation (ISP):** A client should never be forced to implement an interface that he does not use, or clients shouldn’t be forced to depend on methods they do not use.

**Examples of interfaces found in our system:**

* Class “Configuration”: In case of adding another configuration to our system such as (BestFit , FirstCome) it can be easily inherited from “Configuration”.
* Class “Vehicle”: In case we want to use this garage for different types of vehicles rather than cars, it can be easily implemented by inheriting from class “Vehicle”.
* Class “Slot”: if our garage will be used by different vehicles rather than cars, we can have different slot dimensions to suit each of them.