Design Pattern

1- MVC

The **Model View Controller** (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.
MVC is more of an architectural pattern, but not for complete application. MVC mostly relates to the UI / interaction layer of an application. You're still going to need business logic layer, maybe some service layer and data access layer.

- The **Model** contains only the pure application data; it contains no logic describing how to present the data to a user.
- The **View** presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.
- The **Controller** exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.

The pattern is visible on our Class diagram, with the different packages.

2- **Strategy**

Strategy is a **behavioural design pattern** that lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable. **We use it to determine what role the party leaders will have.**

3- Factory

The Factory Method design pattern **defines an interface for creating an object, but let subclasses decide which class to instantiate**. **In the project we use it to create several clients with different and random behaviour.**

4- Bridge

The bridge pattern is used to separate abstraction from its implementation so that both can be modified independently.
This pattern involves an interface which acts as a bridge between the abstraction class and implementer classes and also makes the functionality of implementer class independent from the abstraction class. Both types of classes can be modified without affecting to each other.
Here we have used it on most of our classes, in order to be able to change the class in the event of a problem while keeping the methods of the interface.

5- Singleton

Singleton Design pattern is one of the simplest design patterns. This pattern ensures that a class has only one instance and provides a global point of access to it.
A singleton design pattern in C# is one of the widely used & most popular design patterns around the corner. By using this design pattern, a class will have only a single instance of the program that provides a global point of access to it seamlessly. In other words, we can say that the singleton pattern follows is a class that allows only a single instance of itself to be created and usually gives simple access to that instance.
The constructor is private so that it can only be accessed in the GetInstance command, when it is accessed, either the function returns the existing instance of the Object set as a static variable, or it creates one and places it in variable. Therefore, a single object is instantiated and can be called anywhere. **Here we will use it for the chef, unique and who sometimes needs to be picked up at any time in the kitchen by other cooks.**

6- Observer

This pattern is used when there is one too many relationships between objects such as if one object is modified, its dependent objects are to be notified automatically.
Observer Design Pattern is allowed a single object, known as the subject, to publish changes to its state and other observer objects that depend upon the subject are automatically notified of any changes to the subject's state.
The project specification asks the clerk to react when a customer wants to talk to him as well as when a waiter is too busy to perform certain tasks.
We have therefore decided to apply the Observer Design Pattern to the Room Clerk / Waiter and Room Clerk / Customer pairs.
The Room Clerk then implements the Observer interface while the Servers and Clients implement the Observable interface.