Design Patterns

Project by: Kristina Goldin 317958700, David Ben Yakov 320759921

Design pattern:	**Singleton method**	**Command pattern**	**Factory method**	**Observer pattern**
Serial number:	1	2	3	4
Relevance:	The Singleton design pattern is relevant to the sections where we need to store data to the system database and keep it consistent through the whole running time of the program. For instance, saving employees, shifts and users data.	The Command design pattern is relevant to the sections where we need to send commands from the user menu interface to the classes that implement the functions that we want to run. For instance, adding new data to the program or managing the shifts and employees.	The Factory design pattern is relevant to the sections where we want to create different classes adjusted to the user that is currently connected to the program. For instance, in our program its relevant when we want to show different menu interface to different ranks of employees.	The Observer design pattern is relevant to the sections where we want to notify employees about changes in their shifts. For instance, when assigning shift to an employee or changing characteristics of a shift, such as starting or ending time of the shift.
Classes related:	**Singleton classes:** EmployeeDataBase ShiftDataBase UserDataBase	**Command:** AddEmployeeCommand AddShiftCommand AssignShiftCommand DeleteEmployeeCommand DeleteShiftCommand NotifyEmployeeCommand PrintEmployeesCommand PrintShiftsCommand SearchEmployeeCommand SearchShiftCommand UpdateEmployeeCommand UpdateShiftCommand **Reciever:** EmployeeManagementReciever ShiftManagementReciever AssignShiftNotifier UpdateShiftNotifier **Invoker:** ShiftInvoker **Client:** MainProgram EmployeeManagementMenu ShiftManagementMenu	**Product:** Menu **Concreate products:** EmployeeMenu ManagerMenu **Creator:** MenuFactory **Client:** MainProgram	**Observable (Subject):** AssignShiftNotifier UpdateShiftNotifier **Observer:** Employee

Reason for use:	The Singleton pattern is a creational design pattern that ensures a class has only one instance across an application, with a global point of access to that instance. It’s particularly useful when you need to control access to shared resources or maintain a single point of control, such as in logging systems, database connections, caches, and thread pools. In our program, there must be exactly one instance of a class and it must be accessible to clients from a well-known access point.	Separating the sender (client) of a request from the object that processes the request. This decoupling is achieved through the encapsulation of request details in a command object, enabling clients to parameterize objects with different requests. One of the primary benefits of the Command Pattern is the decoupling of components. By isolating the sender from the receiver, changes in one part of the system do not necessitate modifications elsewhere, enhancing flexibility and ease of maintenance.	We have a superclass with multiple sub-classes and based on input, we need to return one of the sub-class. This pattern takes out the responsibility of the instantiation of a class from the client program to the factory class. Factory method is used when a class cannot predict the type of objects it needs to create.	To achieve loose coupling between objects. This allows the subject (publisher) and observers (subscribers) to interact without being aware of each other’s specific details. It promotes a flexible and maintainable system. To automatically trigger updates of changes made in one object to the other objects. This helps ensure that all dependent objects are informed and can respond accordingly to changes in the subject.