|  |  |
| --- | --- |
| 1- Name | Singleton |
| Context | Create only one instance from database class to connect to database |
| problem | Ensure that it is never possible to create more than one instance of a database class. And provide a global point of access to this instance. |
| Solution | *Constructor is private to ensure that no other class will be able to create an instance of the database class.*  •*Define a public static method “getDBMSObject()” ,The first time this method is called , it creates the single instance of the database class and stores a reference to that object in a static private variable .* |

|  |  |
| --- | --- |
| 2- Name | Observer |
| Context | Send notification message to customers about the new offers , or employee about new update in system. |
| problem | There are hard-coded ties between objects of person class, so we cannot decoupling this objects. |
| Solution | We have created an Interface class ”Inotify” ,  Then create class ”Notification” contains a list of persons to notify of any change in offer’s state, so it should provide methods using which persons can register and unregister themselves.  Subject also contain a method to notify all the persons of any change . |

|  |  |
| --- | --- |
| 3- Name | Composite |
| Context | Compose objects into tree structures to represent whole-part hierarchies. Composite let us treat individual objects and compositions of objects uniformly. |
| problem | we needs to manipulate a hierarchical Address of "primitive" and "composite" objects. Processing of a primitive object is handled one way, and processing of a composite object is handled differently. Having to query the "type" of each object before attempting to process it is not desirable. |
| Solution | *Make a List of Address in class Address to make a chain of Areas connect with each other* |

|  |  |
| --- | --- |
| 4- Name | Strategy |
| Context | Define a group of payment methods, encapsulate each one, and make them interchangeable.  Capture the abstraction in an interface, bury implementation details in derived classes |
| problem | Every type of payment method have the same method “pay()”, but they are different in other attributes. |
| Solution | Create Interface “PaymentStrategy” represent the method signature “pay()” expectations by the customer .  Then inherit hierarchically the types of payment  this represents dynamic polymorphism.  And then all attributes for every type. |

|  |  |
| --- | --- |
| 5- Name | Decorator |
| Context | add additional cost or discount to the cost of item dynamically.  Decorators provide a flexible alternative to sub-classing for extending functionality. |
| problem | want to add cost or discount to individual item at run-time. Inheritance is not feasible because it is static. |
| Solution | Create Interface class ”Iinvoice” is the base class for both the invoice class and all decorator classes(offer,delivery), then in invoice class the method get “getcost()” implemented and ready to wrapped by decorator.  The classes(offer,delivery) decorate cost by their values. |

|  |  |
| --- | --- |
| 6- Name | Filter |
| Context | filter a set of objects using different criteria and chaining them in a decoupled way through logical operations. |
| problem | implementing multiple functionalities and most of them require different filter criteria to filter something. This cost hard coding the filters inside the functionalities. |
| Solution | Create a product object ,Criteria interface and concrete classes implementing this interface to filter a list of product objects .  FilterCriteria class use criteria objects to filter list of product objects based on various criteria and their combination. |