Observer Pattern

- In the project, there are 6 classes, Main, Workspace, TSPNN, TSPPro, KMeans, and City. The Workspace class is used to create the GUI, and TSPNN, TSPPro, KMeans is our logic/implementation/algorithm classes.
- When the new city is added to the list of cities(showing them in the GUI), and based on the algorithm selected from menu items we need to find the shortest possible route that visits each city exactly once and returns to the origin city if it is both TSPNN and TSPPro. If clusters is selected in menu KMeans algorithm runs and creates the clusters based on the user input.
- So, We need to wait until the algorithm is executed completely, and according to the outputs, cities will be connected.
- In this situation, the observer partner is the best fit. The Workspace class will be implementing the Observer interface and the TSPNN, TSPPro, Kmeans will be extending the Observable class.
- So, when a new city is added or any city is dragged from one place to another, in Workspace (i.e change in data), it will call the algorithm based on selction in the menu.
- Once TSPNN/TSPPro/KMeans is done with its job, it will notify the Workspace class and the Workspace class will update the connections between the cities. In this way, the Observer partner is implemented in the project

Singleton Pattern

- In this project we have implemented singleton pattern for Blackboard, TSPNN, TSPPro, FactoryCity, FactoryConnections, MessageDisplayer classes. As objects of these classes are same/one throughout the project. So, in this situation Singleton pattern is the best fit.
- When getInstance() method is called on particular class it returns the instance of the class and only one instance is created throughout the project.

Chain of Responsibility:

- In this project we have implemented chain of responsibility for TSPNN, TSPPro, and for KMeans Classes. Where these classes implements a Handler interface and implements HadleRequest method.
- These patterns best fit in this situation because only one algorithm is selected at a time.

Decorator Pattern:

- In this project, we have implemented a decorator pattern where we can decorate a basic square with circle inside or square surrounding other squares and circle inside a square and surrounded by other squares.
- Here we have implemented two basic shapes square and circle which implements Shape interface. And also a ShapeDecorator class which implemements the Shape interface.
- There are three more decorator classes CircleSquareDecorator, CircleSquareGroupDecorator and SquareGroupDecorator which extends the ShapeDecorator class.
  So, Decorator pattern bests fits in this situation.

Factory Pattern:

- In this project, we have implemented a factory pattern where we do not want to expose the object creation logic where it is used. We have implemented the factory pattern to create the objects of class City and class Connection separately.
- Here we have implemented the class FactoryCity which implements Factory interface and the object of class City is only created in class FactoryCity.
- The same goes for implementation of factory pattern for class Connection. Here we have implemented the class FactoryConnections which implements FactoryForConnection interface and the object of class Connection is only created in class FactoryConnections.

References :

https://github.com/psyclone20/k-means-clustering : for clustering algorithm.