**Mediator Design Pattern:**

Mediator Design pattern is one of the Behavioral Design Pattern. With the mediator design pattern, communication between objects is encapsulated with mediator object.

Object no longer communicate directly with each other, but instead communicate through the mediator. This reduces the dependencies between communicating objects, thereby lowering the coupling.

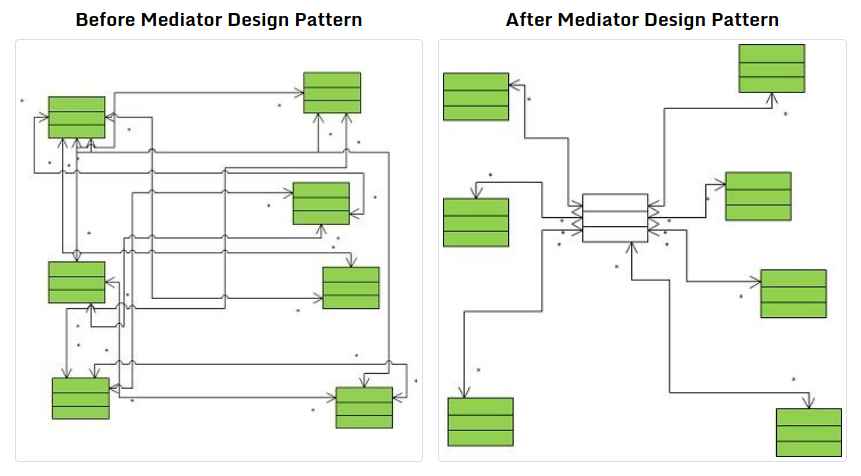
The mediator pattern promotes loose coupling by keeping objects from reffering to each other explicitly, and it lets you vary their interaction independently.

Concept

A mediator is the one who takes the responsibility of communication among a group of objects. The mediator acts as an intermediary who can track the communication between two object.

The other objects (also knows as Colleagues) in the System are also aware of this mediator and they know that if they need to communicate among themselves, they need to go through the mediator.

The advantage of using mediator is that, we can reduce the direct interconnection among the objects and thus lower the coupling.



**Design a Traffic Signal?**

When one light turns ON, rest other have to be turned OFF.

If each light will communicate with another lights to change their state then it would be highly coupled and difficult to maintain.

By introducing light mediator which is responsible to hold object reference of all other light and communication task, the design becomes flexible and de-coupled.

We can add any color light and remove it whenever we want that at runtime.

**import** java.util.HashSet;

**public** **class** MediatorDesignPatternDemo {

**public** **static** **void** main(String[] args) {

LightMediator lightMediator = **new** LightMediator();

Light red = **new** Light("Red", lightMediator);

Light green = **new** Light("Green", lightMediator);

Light yellow = **new** Light("Yellow", lightMediator);

red.turnON();

green.turnON();

yellow.turnON();

}

}

/\*\*

\*

\* When any light turns ON in traffic Signal,

\* Light Mediator turns OFF other lights.

\*

\*/

**class** LightMediator {

// Using Hashset to achieve uniqueness in light color.

HashSet<Light> trafficSignal = **new** HashSet<>();

/\*\*

\* register passed light object in LightMediator

\* It is being called by constructor of Light class.

\* We can also explicitly call this method.

\*/

**public** **void** registerLight(Light light) {

trafficSignal.add(light);

}

/\*\*

\* unregisters light from LightMediator

\*/

**public** **void** unRegisterLight(Light light) {

trafficSignal.remove(light);

}

/\*\*

\* Turns off all the lights other than

\* passed light Object

\*/

**void** turnOffAllOtherLights(Light light) {

**for** (Light l : trafficSignal) {

**if** (!(l.equals(light))) {

l.turnOFF();

}

}

System.***out***.println("------------------------------");

}

/\*\*

\* When any light turns ON, it calls this method

\* to notify mediator. Light mediator will turn OFF

\* all other light by calling turnOffAllOtherLights(light)

\* method

\*/

**public** **void** notifyMediator(Light light) {

turnOffAllOtherLights(light);

}

}

/\*\*

\* Represent light in traffic signal i.e. Red, Green or Yellow light

\*/

**class** Light {

/\*\*

\* represents the turn ON and OFF state of light

\*/

**enum** State {

***ON***, ***OFF***

}

**private** String color;

**private** State currentState;

**private** LightMediator LightMediator;

/\*\*

\* Creates Light object and register it to LightMediator

\*/

Light(String color, LightMediator LightMediator) {

**this**.color = color;

**this**.LightMediator = LightMediator;

LightMediator.registerLight(**this**);

}

/\*\*

\* Turns ON the light and notify mediator for the same

\*/

**void** turnON() {

currentState = State.***ON***;

System.***out***.printf("%s is turned %s \n", **this**, currentState.***ON***);

LightMediator.notifyMediator(**this**);

}

/\*\*

\* Turns OFF the light

\*/

**void** turnOFF() {

currentState = State.***OFF***;

System.***out***.printf("%s is turned %s \n", **this**, currentState.***OFF***);

}

/\*\*

\* It is generated using its String property

\* color's inbuilt hashcode() method

\*/

@Override

**public** **int** hashCode() {

**return** color.hashCode();

}

/\*\*

\* Uses its String property color's inbuilt hashcode() method

\*/

@Override

**public** **boolean** equals(Object obj) {

Light light = (Light) obj;

**return** color.equals(light.color);

}

/\*\*

\* returns light color

\*/

@Override

**public** String toString() {

**return** color;

}

}