## Behavioral Patterns - Mediator Design Pattern

- Genelde GUI'ler için kullanılır.Örnek bir chat uygulaması, yada bir havalimanı kulesi. Uçakların hiçbiri birbirleri ile konuşmazlar kule ile konuşurlar. Kule karar verir
- Mediator Design Pattern nesnelerin aralarındaki iletişimin tek bir noktadan sağlanması ve koordine edilmesi gerektiği durumlarda kullanılır.Nesneler birbirleri ile doğrudan konuşmak yerine merkezi bir yapı aracılığı ile haberleşirler bu sayede nesneler arasında bağımlılık azalır. Nesneler birbirlerinin kim olduklarını bilmeden merkez aracılığı ile haberleşebilirler.

```
interface Mediator{
   //Mediator interface'i
   void notify(Component sender,String message);
   void register(Component component);
}
```

Tüm component'ler için ortak olacak abstract Component class'ı

```
abstract class Component{
   private String name;
   protected Mediator mediator;
   public Component(String name,Mediator mediator) {
      this.name = name;
      this.mediator = mediator;
   }
   public abstract void send();
   public abstract void receive(String message);
   public String getName(){
      return this.name;
   }
}
```

Yukarıda gördüğünüz üzere Component'in ismi ve mediator bilgisi yer alacak

Component Class'ları

```
class ComponentA extends Component{
    public ComponentA(Mediator mediator){
        super("Component-A", mediator);
    }
    @Override
    public void send() {
        //Component B'ye mesaj
        String message = "I am Component A and i am sending message";
        //notify'i yapacak olan mediator yani arabulucu
        this.mediator.notify(this,message);
    }
    @Override
    public void receive(String message) {
        //Component A mesaj alıyor
        System.out.println("Component A got : " + message);
    }
}
class ComponentB extends Component{
    public ComponentB(Mediator mediator){
        super("Component-B",mediator);
    }
    @Override
    public void send() {
        //Component A'ya mesaj
        String message = "I am Component B and i am sending message";
        //notify'i yapacak olan mediator yani arabulucu
        this.mediator.notify(this,message);
    }
    @Override
    public void receive(String message) {
        //Component B mesaj alıyor
        System.out.println("Component B got : " + message);
    }
}
```

Mediator Interface'inin concrete nesnesi

```
class ConcreteMediator implements Mediator{
    private final String COMPONENT_A = "Component-A";
    private final String COMPONENT_B = "Component-B";
    private Map<String,Component> registerComponents = new HashMap<>();
    public void notify(Component sender, String message) {
        String senderName = sender.getName();
        if (COMPONENT_A.equals(senderName)){
            reactOnA(message);
        }else if (COMPONENT_B.equals(senderName)){
            reactOnB(message);
        }
    }
    //Map içerisine eğer yoksa component'leri ekliyorum
    public void register(Component component) {
        this.registerComponents.put(component.getName(),component);
    }
    private void reactOnA(String message){
        System.out.println("Mediator is an action : ");
        registerComponents.get(COMPONENT_B).receive(message);
    }
    private void reactOnB(String message){
        System.out.println("Mediator is an action : ");
        registerComponents.get(COMPONENT_A).receive(message);
    }
}
```

Main

```
public static void main(String[] args) {
    Mediator mediator = new ConcreteMediator();
    Component compA = new ComponentA(mediator);
    Component compB = new ComponentB(mediator);
    mediator.register(compA);
    mediator.register(compB);
    compA.send();
    compB.send();
}
```

```
"C:\Program Files\Java\jdk1.8.0_231\bin\java.exe" ...
Mediator is an action :
Component B got : I am Component A and i am sending message
Mediator is an action :
Component A got : I am Component B and i am sending message
```