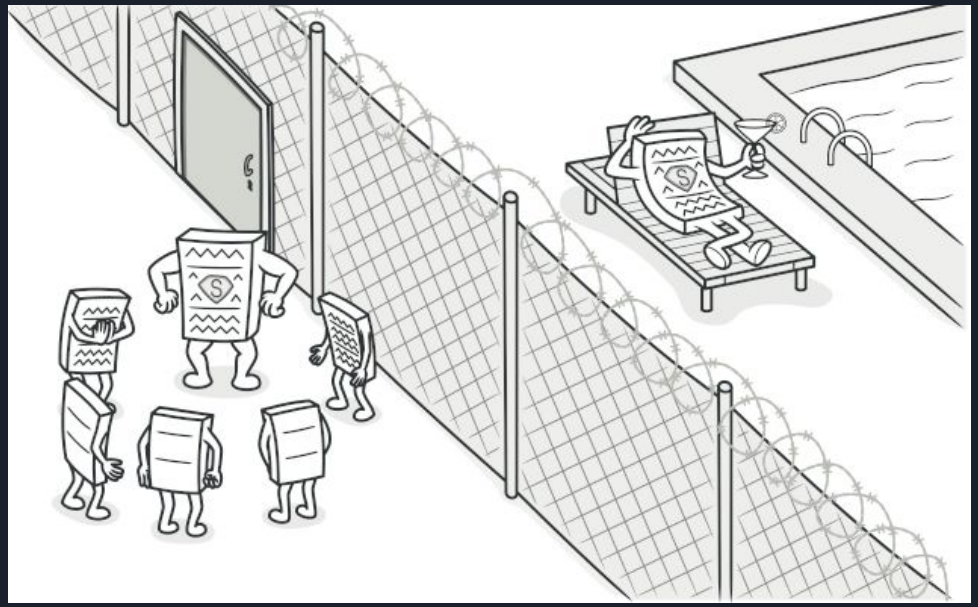
Proxy
(Design Pattern)

KAING Thierry

MENARD Dorian

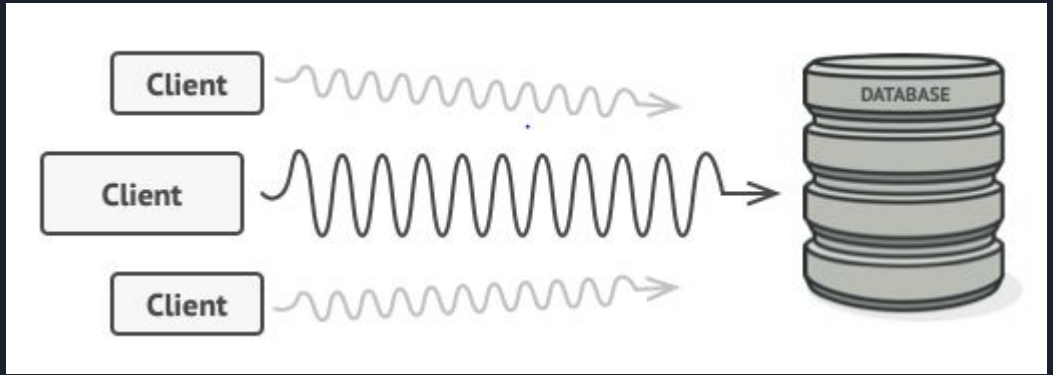
Présentation
Proxy



Design Pattern permettant de contrôler l'accès à un Objet en fonction des cas en remplaçant celui-ci et en ajoutant des traitements supplémentaire

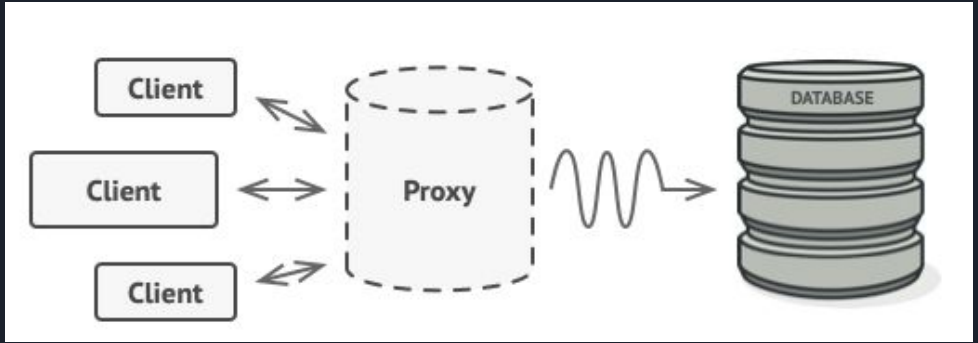
Problématique
Proxy

Objet Type Base de données qui consomme beaucoup de ressources système où l'accès est utile de temps en temps



Solution Proxy
Proxy

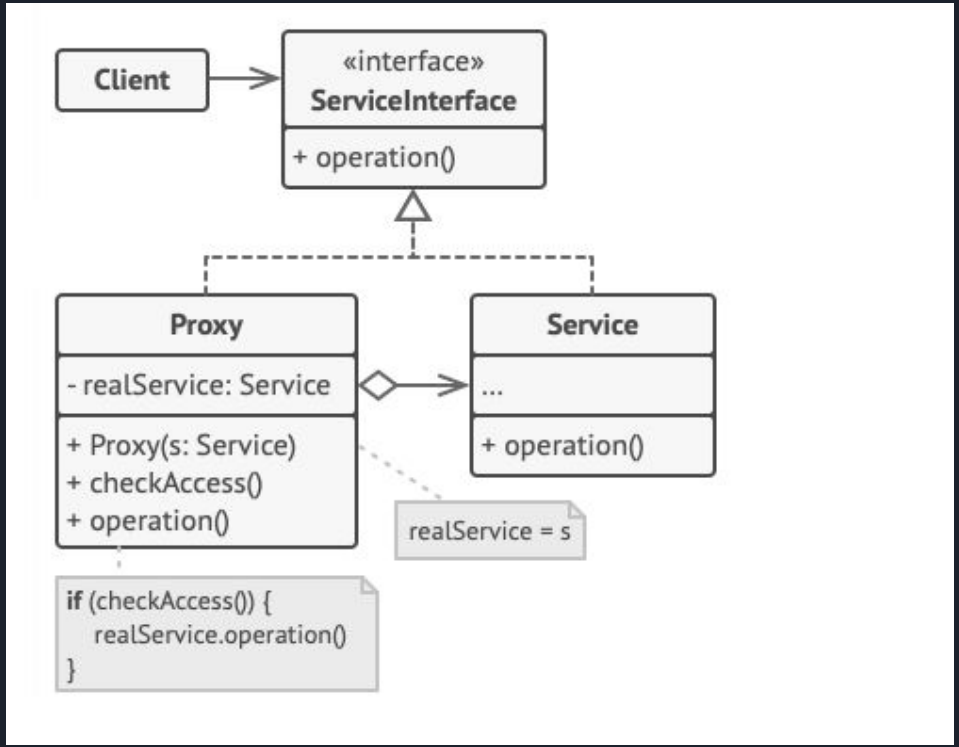
Ici, nous pouvons voir la position du Proxy parmi toutes les entités ainsi que l'importance qu'il peut apporter



Structure proxy

Proxy

Ici, nous avons une Class Proxy implémentant la même Interface que le Service original



Exemple de Code

Interface of Internet

```java
public interface Internet
{
    public void connectTo(String serverhost) throws Exception;
}
```

RealInternet.java

```java
public class RealInternet implements Internet
{
    @Override
    public void connectTo(String serverhost)
    {
        System.out.println("Connecting to "+ serverhost);
    }
}
```

Exemple de Code

```
public class ProxyInternet implements Internet
{
    private Internet internet = new RealInternet();
    private static List<String> bannedSites;

    static
    {
        bannedSites = new ArrayList<String>();
        bannedSites.add("abc.com");
        bannedSites.add("def.com");
        bannedSites.add("ijk.com");
        bannedSites.add("lnm.com");
    }

    @Override
    public void connectTo(String serverhost) throws Exception
    {
        if(bannedSites.contains(serverhost.toLowerCase()))
        {
            throw new Exception("Access Denied");
        }

        internet.connectTo(serverhost);
    }

}
```

Exemple de Code

Client.java

```
public class Client
{
    public static void main (String[] args)
    {
        Internet internet = new ProxyInternet();
        try
        {
            internet.connectTo("geeksforgeeks.org");
            internet.connectTo("abc.com");
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

Proxy

Cas d’utilisation Proxy

Proxy

- **Lazy Initialization** : objet qui consomme beaucoup de mémoire
- **Access Control** (protection proxy) : récupérer l’objet uniquement pour des utilisateurs spécifiques
- **Logging Requests** (logging proxy) : garder un historique des requêtes à un service object

Conclusion
Proxy

Avantages

- Gérer le temps de vie du service objects
- Possibilité de créer le proxy sans changer le code du service ou du client

Inconvénients

- Code plus complexe par l'implémentation de plusieurs class
- Les réponses du services peuvent prendre plus de temps

Sources

https://refactoring.guru/design-patterns/proxy

https://www.geeksforgeeks.org/proxy-design-pattern/