Patron proxy

- Intermédiaire qui se substitue à un fournisseur de services (une autre classe), le mandataire ajoute une indirection à l'utilisation du substitué.
- Le substitut est utilisé principalement pour effectuer des contrôles (accès aux services).
- Il peut être utilisé pour simplifier l'utilisation d'un objet complexe.

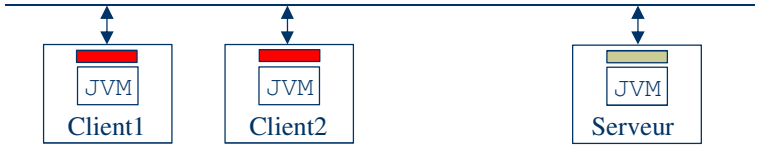
Patron proxy

- Le mandataire agit sur le substitué pour le client par procuration. On parle aussi de substitut ou subrogé (surrogate).
- Le mandataire a la même interface que le substitué.
- Cela sert à contrôler, différer, optimiser, sécuriser, accéder à distance …

Exemple : appels distants

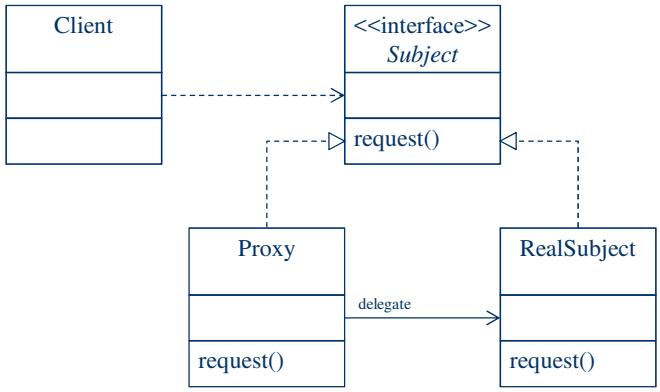
- Les clients déclenchent des méthodes côté serveur… sans le voir car ils déclenchent des méthodes d'un mandataire côté client qui se charge de l'appel distant :
  - transmission des paramètres et du résultat,
  - gestion des exceptions levées côté serveur à destination d'un client.

Exemple : appels distants



- Le mandataire est téléchargé par les clients depuis le serveur.

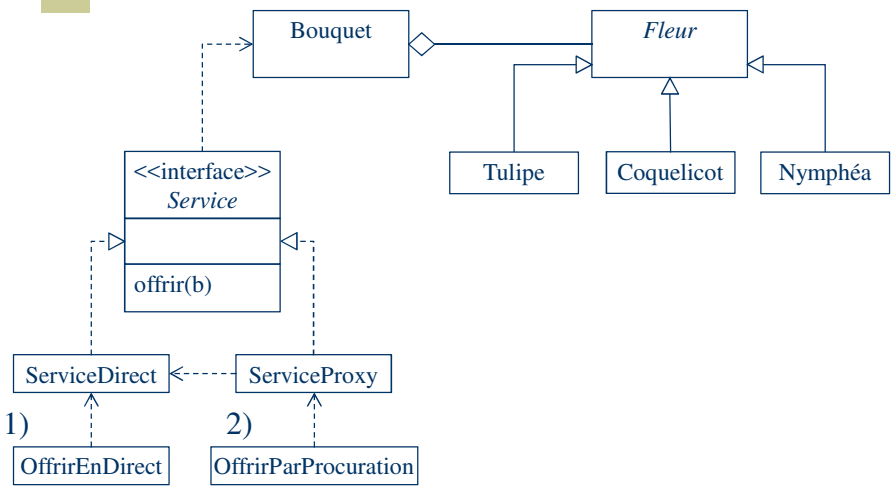
Patron proxy du GoF



Exemple : offrir un bouquet

```
public interface Service{
        /** Offrir un bouquet de fleurs.
         * @param bouquet le bouquet
         * @return accepté ou refusé
         * @exception fleurs fanées, etc…
         */
        public boolean offrir(Bouquet bouquet) throws Exception;
}
```

Offrir un bouquet



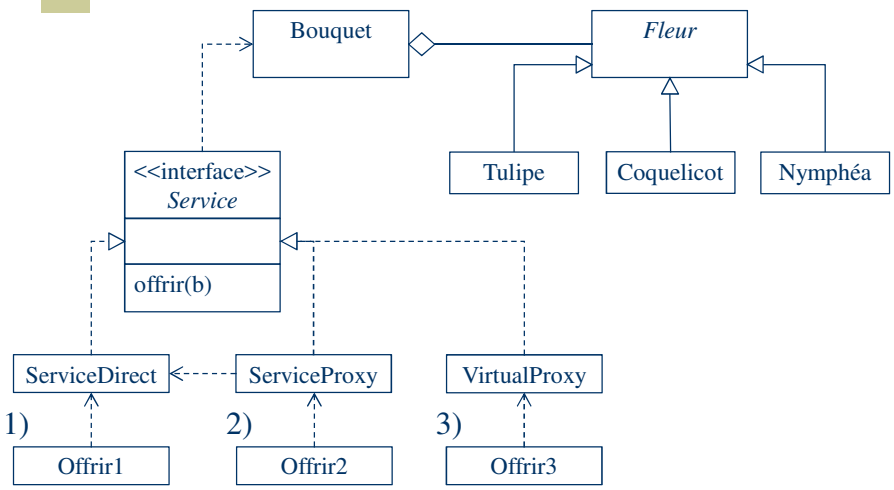
VirtualProxy

- L'objet implantant le service est coûteux (en mémoire par exemple, en temps).
- Chargement de la classe et création d'un objet au moment ultime.

```
public class VirtualProxy implements Service {
    private Service service=null;

    public VirtualProxy(){
    private Service getServiceDirect() {
    public boolean offrir(Bouquet bouquet)
    throws Exception {
        boolean résultat;
        Service service = getServiceDirect();
        résultat = service.offrir(bouquet);
        return résultat;
    }
}
```

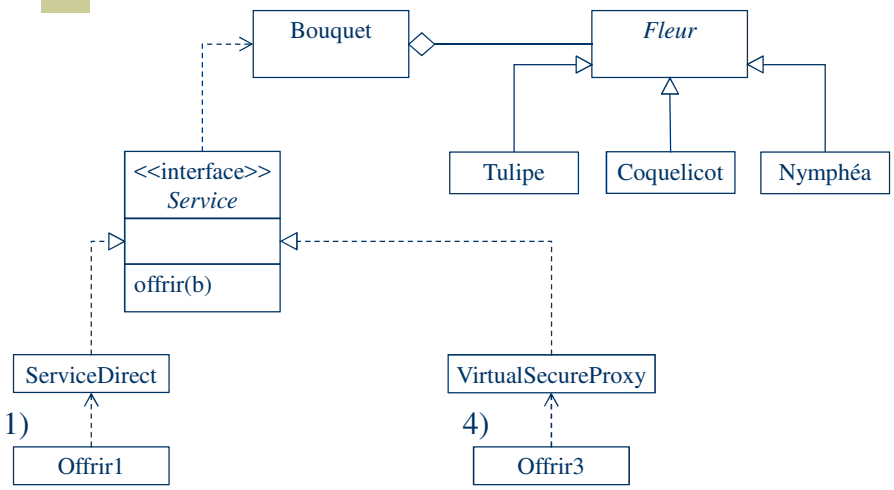
Offrir un bouquet



VirtualSecureProxy

- On sécurise le service par un URLClassLoader.

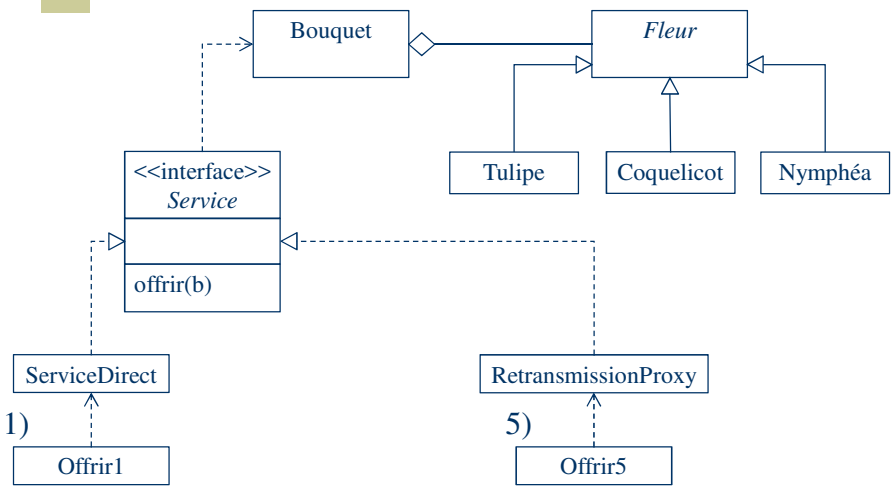
Offrir un bouquet



RetransmissionProxy

- Pour garantir le succès de l'appel au service, on le répète jusqu'à son succès (dans des limites fixées).

Offrir un bouquet

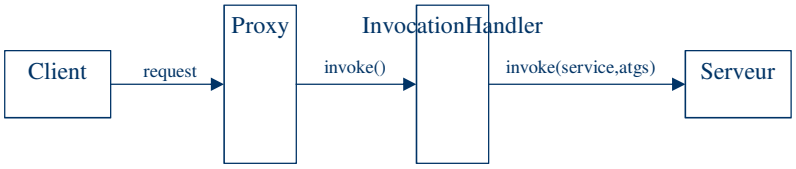


Comment faire si …

- Le service réel n'est connu qu'à l'exécution.
- En fonction du contexte, les méthodes du service ne sont pas toutes les bienvenues.

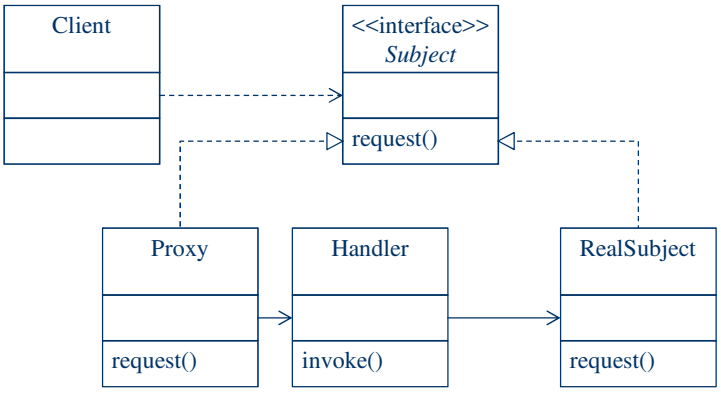
→ Proxy dynamique

Proxy + InvocationHandler



- InvocationHandler
- Création dynamique du Proxy, qui déclenche la méthode invoke de « InvocationHandler »

Patron DynamicProxy



Interface InvocationHandler

```
interface InvocationHandler {
        Object invoke(Object proxy,
                      Methode m,
                      Object[] args)
        throws Throwable;
}
```

Handler

```
public class Handler implements InvocationHandler{
  private Service service;
  public Handler(){
    this.service = new ServiceDirect();
  }
  public Object invoke(Object proxy,
                       Method method,
                       Object[] args)
    throws Exception {
    return method.invoke(service, args);
  }
}
// invoke est déclenchée par le mandataire dynamique
créé par une méthode toute prête newProxyInstance
```

Plus générique encore

- Connaissance préalable du service invoqué
- Ajout nécessire d'un intermédiaire du mandataire (qui implante InvocationHandler)

→Le service devient un paramètre du constructeur (Object target)