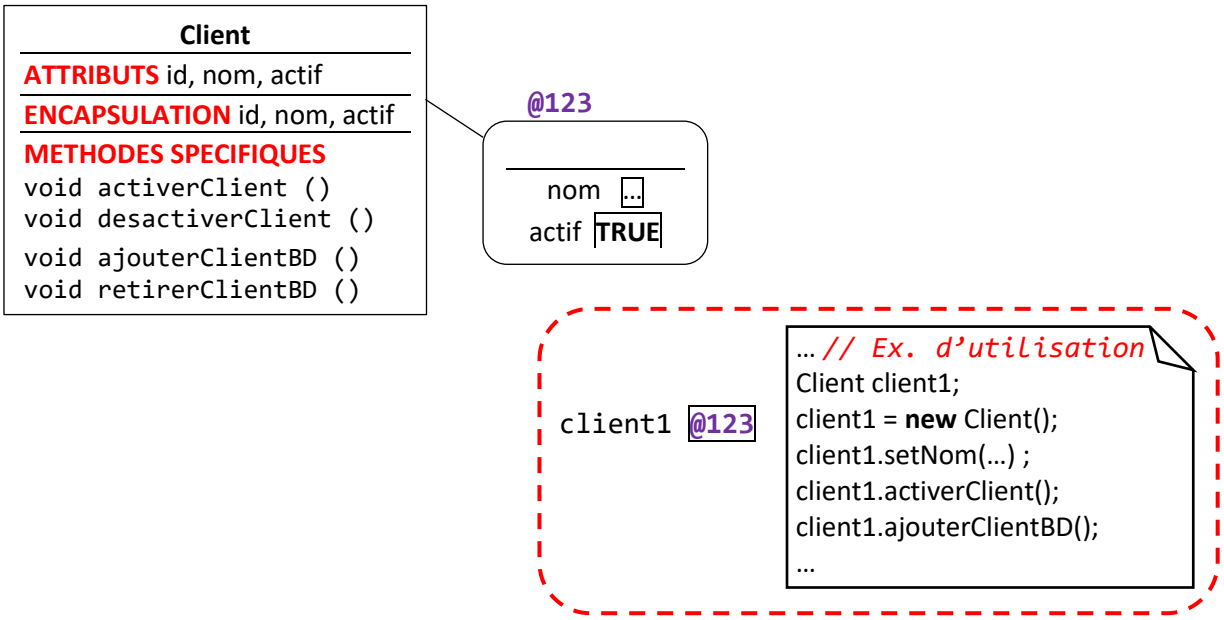
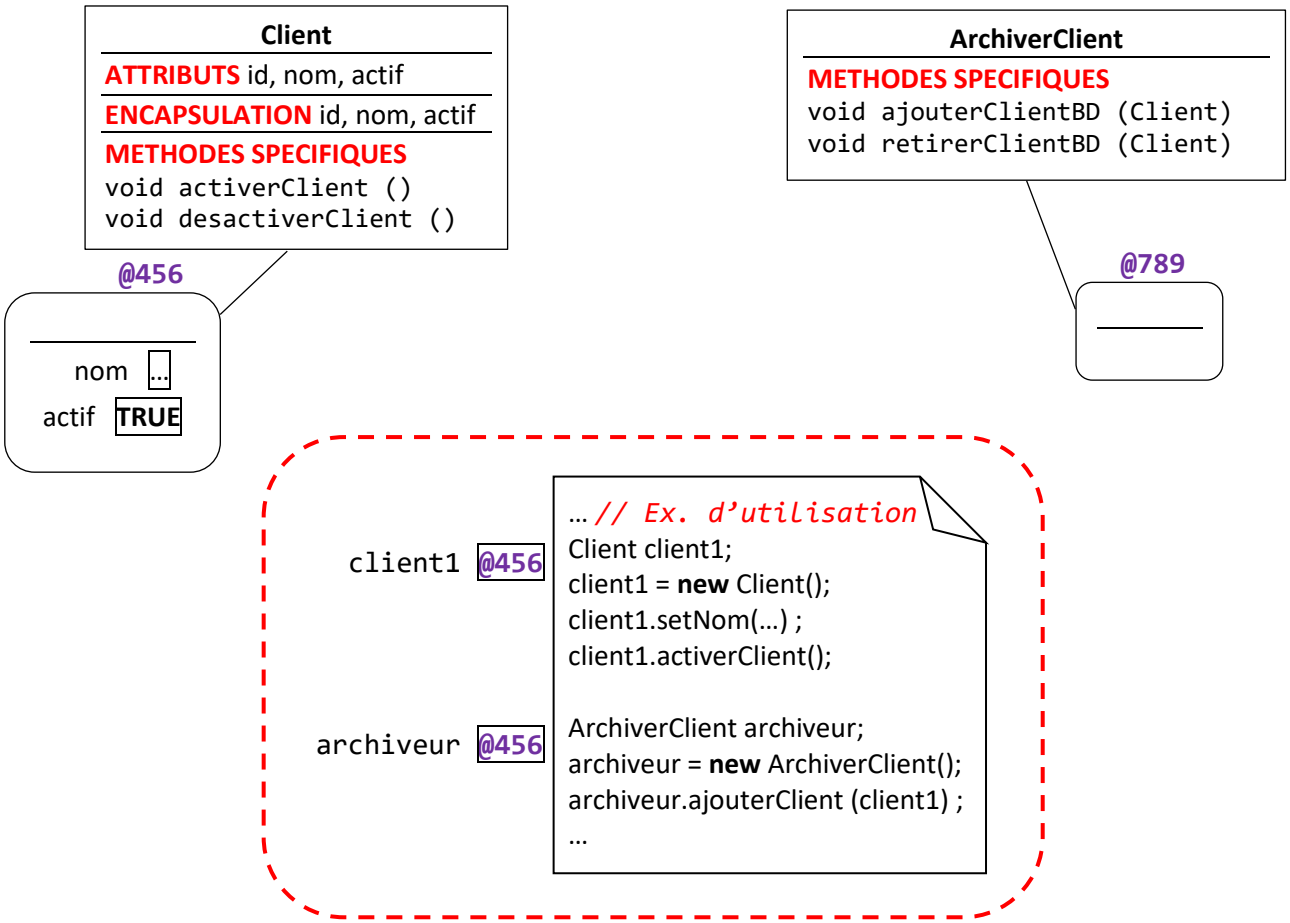
Le principe de **Simple responsabilité** stipule qu'une **classe ne devrait avoir qu'une seule raison de changer** et donc une **seule responsabilité**

Simple responsabilité NON OK



Exemple de proposition **incorrecte** car la classe a **deux responsabilités**, la responsabilité sur la **règle de gestion du client (actif/nonActif)** et la responsabilité de **persistance du client en base de données.**

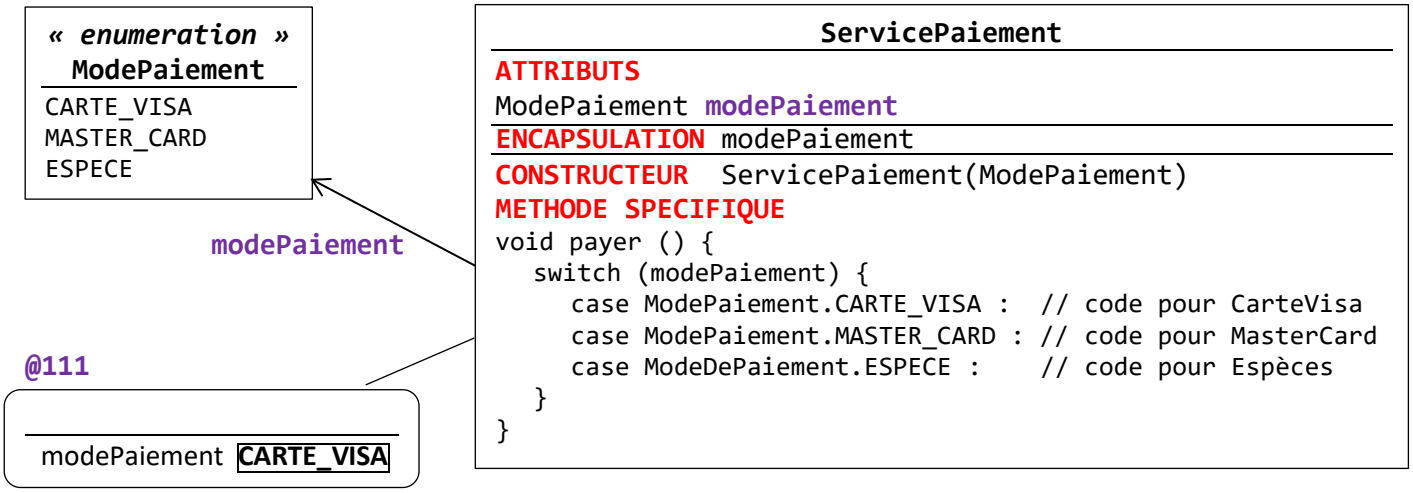
Simple responsabilité **OK**

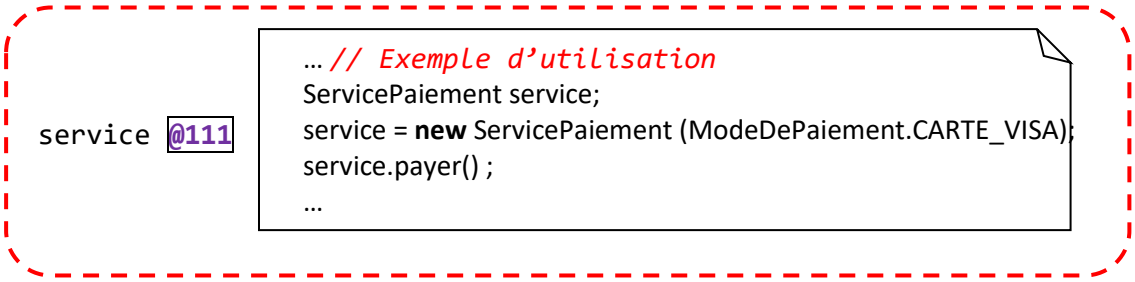


Cette proposition **est correcte** car les responsabilités ont été réparties et **chaque classe n'a qu'une seule raison de changer.**

Le principe **Ouvert-Fermé** affirme qu'**une classe** doit être à la fois **ouverte à l'extension** et **fermée à la modification**.

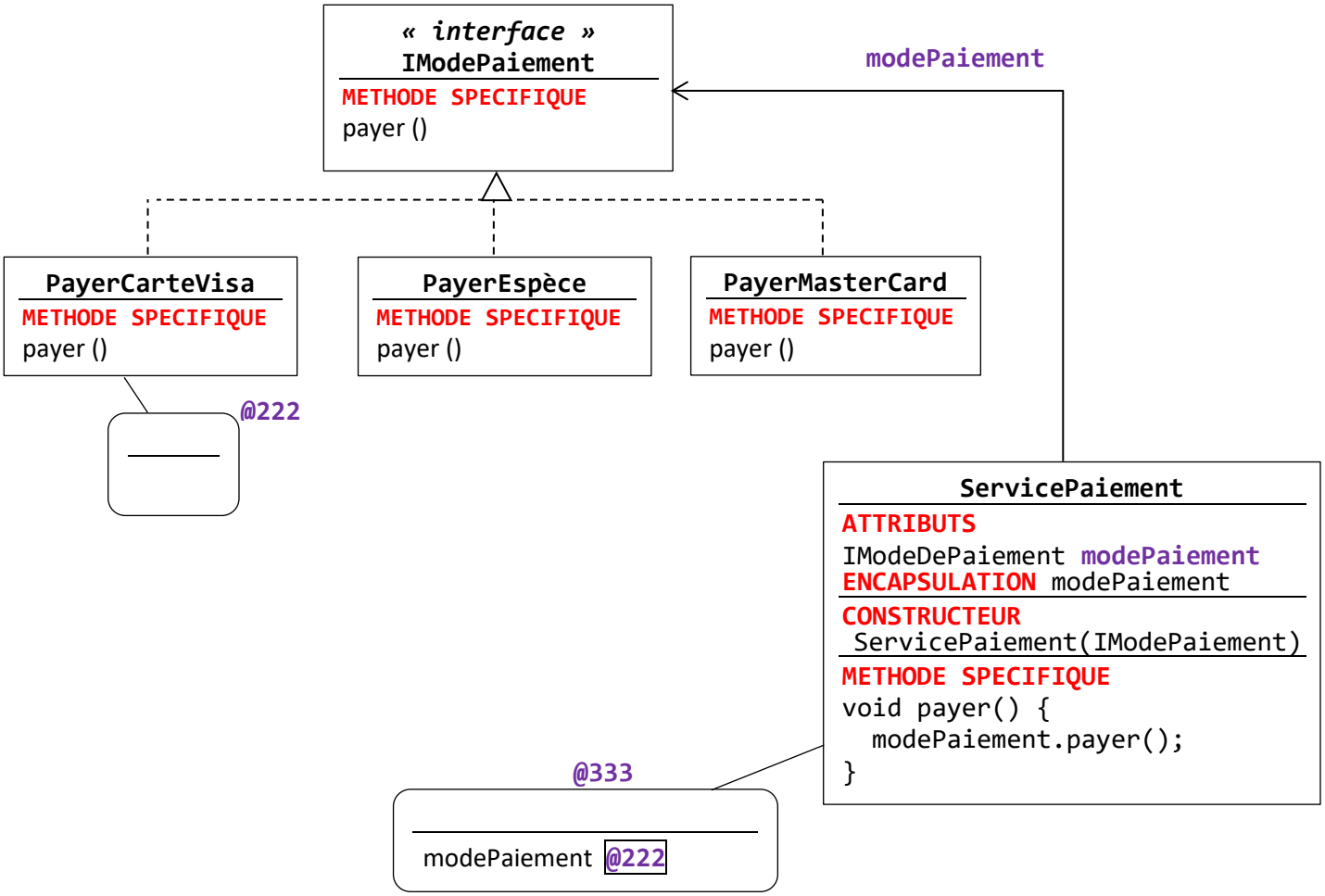
Ouvert Fermé **NON OK**

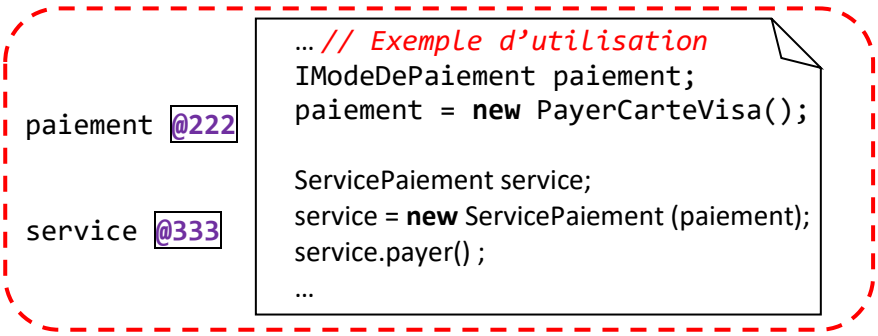




Exemple de proposition **incorrecte** car si un **nouveau mode de paiement** est ajouté, le code de la **classe devra être modifié.** La solution n'est donc pas fermée à la modification.

Ouvert Fermé **OK**

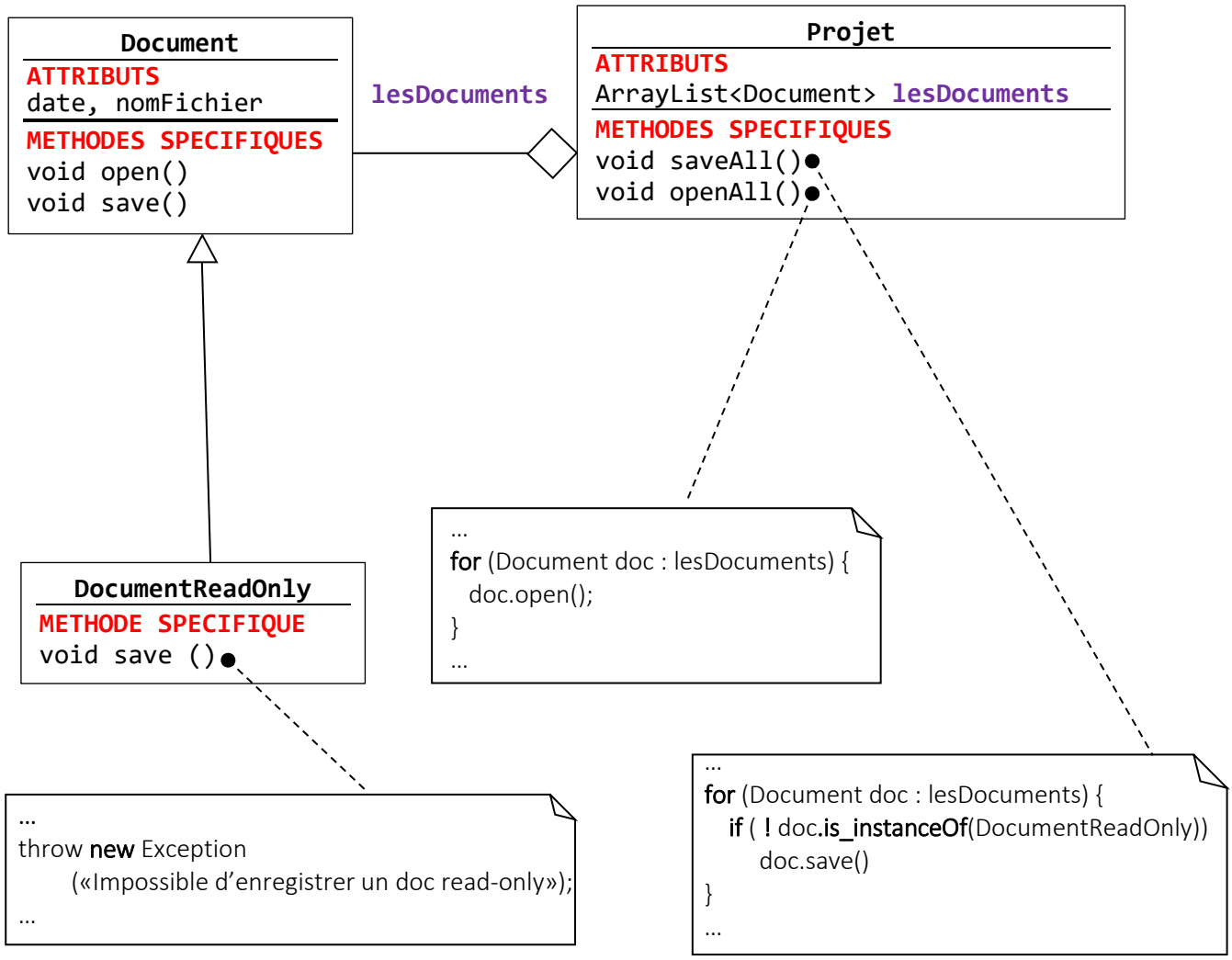




Cette proposition est **correcte** car si un **nouveau mode de paiement** est ajouté, alors la classe ServicePaiement **n'a pas être modifiée.**

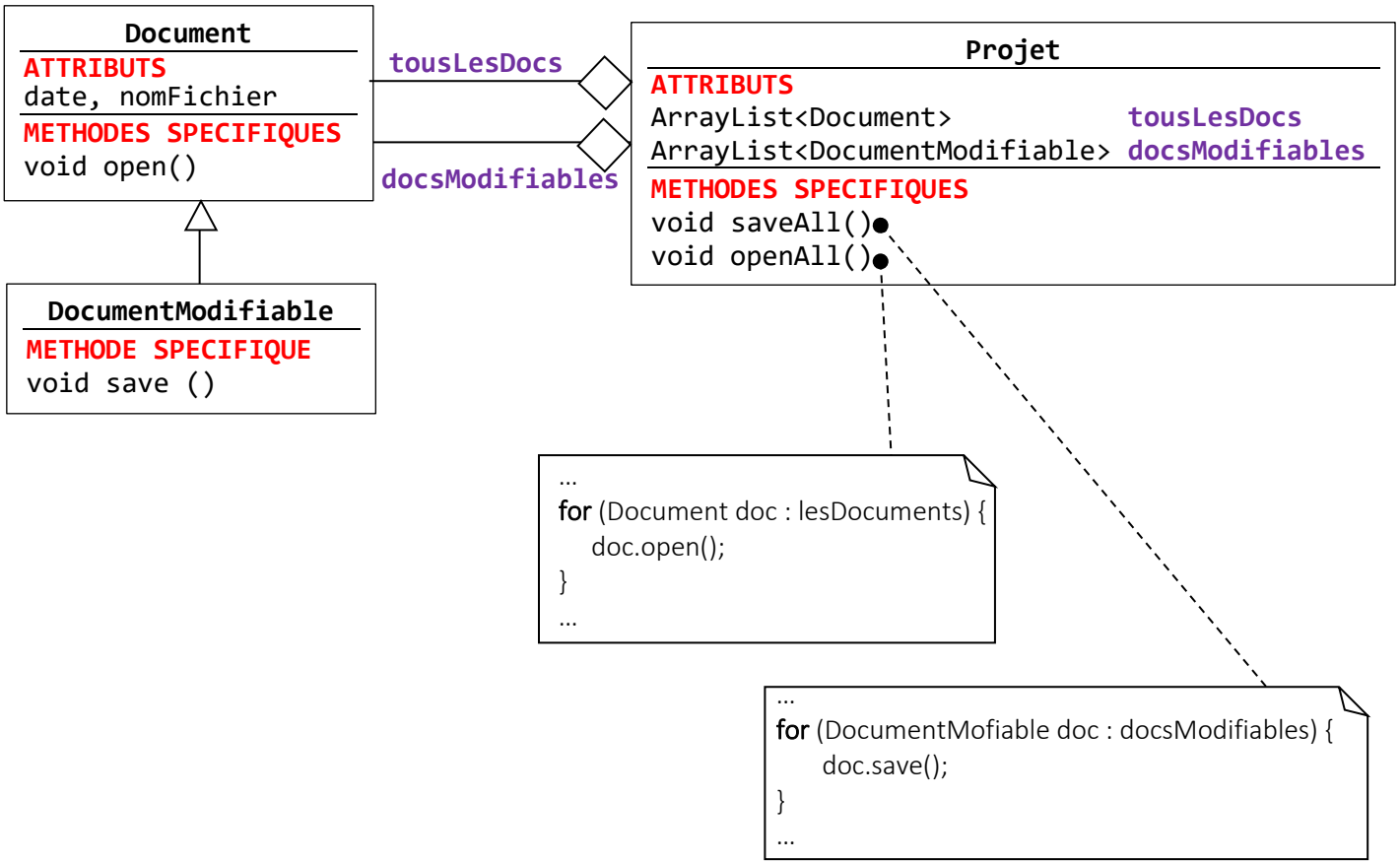
Le principe de **Substitution de Liskov** affirme **que les classes dérivées doivent être substituables à leurs classes de base.**

Substitution NON OK



La sauvegarde **`save()`** n'a pas de sens pour un **doc** en lecture seule, alors la sous-classe essaie de résoudre ce problème en redéfinissant/*annulant* le comportement de base de la méthode ☹

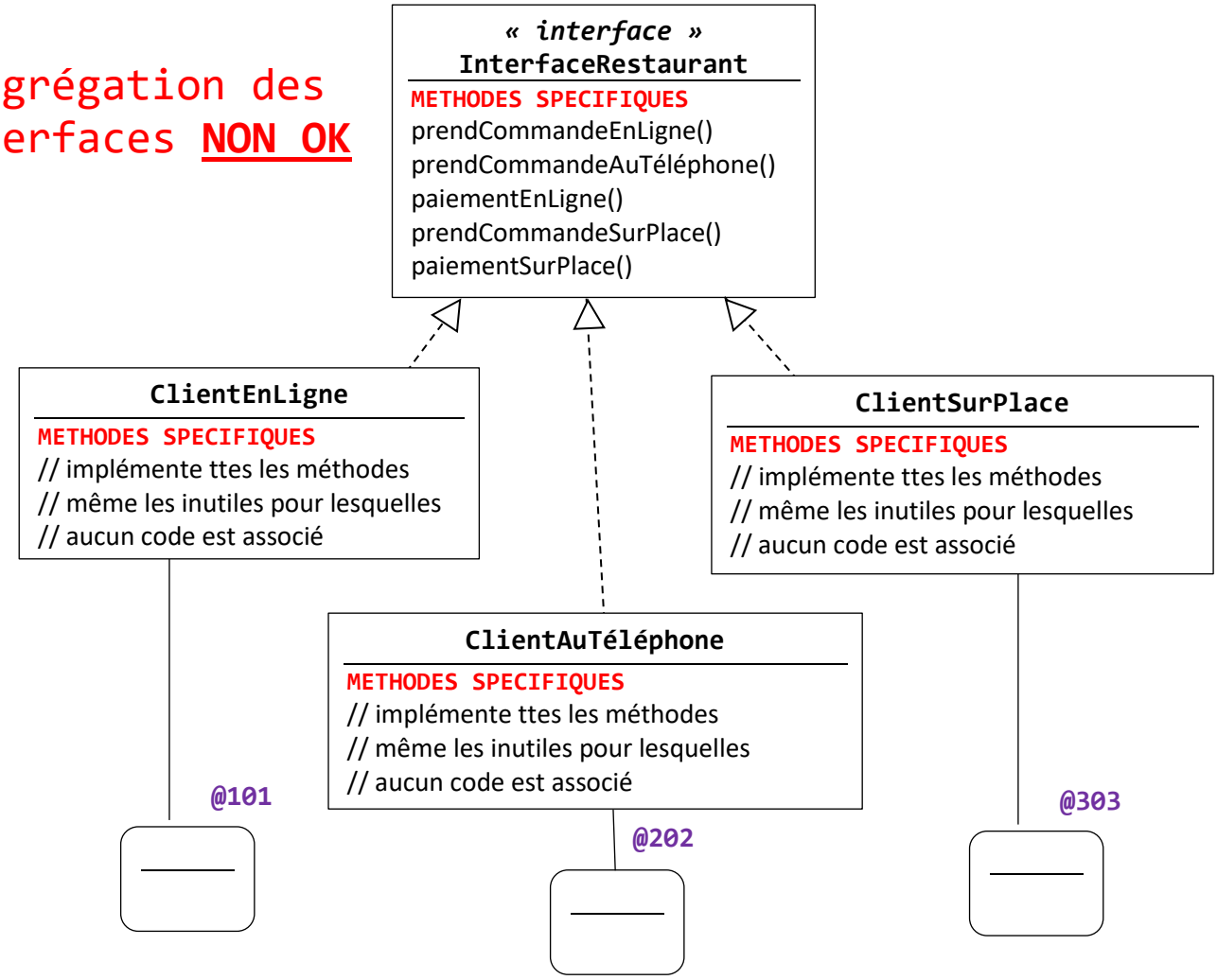
Substitution OK



La solution a consisté à retirer la capacité de sauvegarde **`save()`**, de la super classe **`Document`** (cad de la classe situé à la base de la hiérarchie).
Avec **`DocumentModifiable`** chacune de ses instances peut réaliser les traitements de la super classe **`Document`**, à savoir **`open()`**.
Elle peut donc **être substituée** par une instance de la super classe.

Le principe de **Ségrégation d'Interface** affirme qu'**il vaut mieux plusieurs interfaces spécifiques** à une classe, **plutôt qu'une grosse interface générique** ; car il ne faut pas obliger celui qui implémente une interface, à implémenter des méthodes qui ne l'intéresse pas.

Ségrégation des Interfaces NON OK

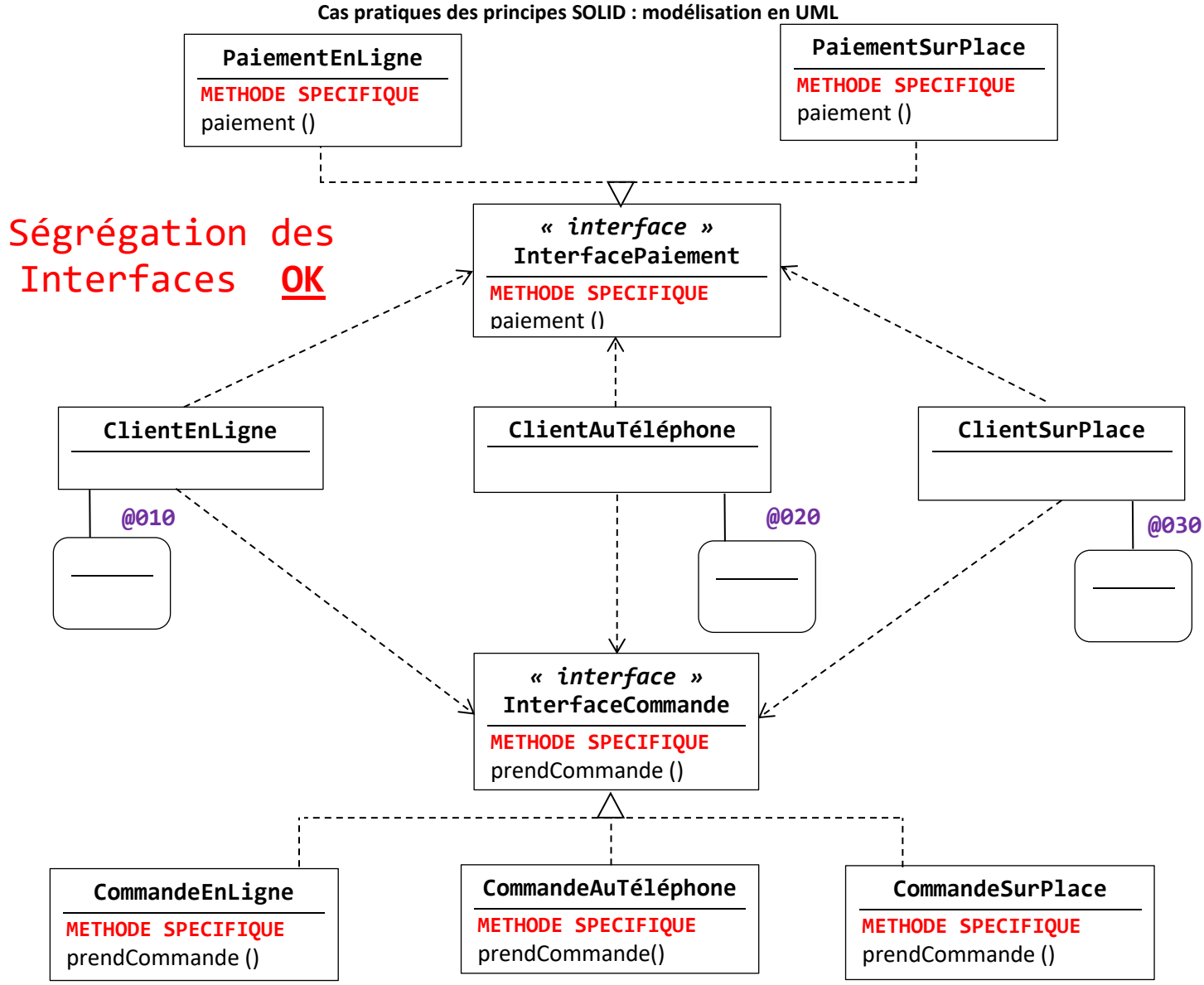



```
class ClientSurPlace implements InterfaceRestaurant {
class ClientAuTéléphone implements InterfaceRestaurant {
class ClientEnLigne implements InterfaceRestaurant {
   prendCommandeEnLigne() { // Comportement spécifique 😊 }
   paiementEnLigne()      { // Comportement spécifique 😊 }
   prendCommandeAuTéléphone() { // Ne rien faire ☹ }
   prendCommandeSurPlace()    { // Ne rien faire ☹ }
   paiementSurPlace()         { // Ne rien faire ☹ }
}
```

client1enLigne	@101
client2auTéléphone	@202
client3surPlace	@303

```
… // Exemple d'utilisation
InterfaceRestaurant  client1EnLigne;
client1enLigne     = new ClientEnLigne ();
InterfaceRestaurant  client2AuTéléphone;
client2AuTéléphone = new ClientAuTéléphone ();
InterfaceRestaurant  client3SurPlace;
client3SurPlace    = new ClientSurPlace ();
…
```

Cas pratiques des principes SOLID : modélisation en UML




```
class CommandeSurPlace implements InterfaceCommande {
class CommandeAuTéléphone implements InterfaceCommande {
class CommandeEnLigne implements InterfaceCommande {
class PaiementSurPlace implements InterfacePaiement {
class PaiementEnLigne implements InterfacePaiement {
   paiement () {
      // Comportement que j'attribue au paiement en ligne
   }
}
Class ClientSurPlace {
class ClientAuTéléphone {
class ClientEnLigne {
   InterfaceCommande commnd = new CommandeEnLigne();
   InterfacePaiement paie   = new PaiementEnLigne();
   // Suite du comportement que j'attribue au client
   // en ligne
```

clientAenLigne @010

clientBauTéléphone @020

clientCsurPlace @030

```
… // Exemple d'utilisation
ClientEnLigne      clientAenLigne;
clientAenLigne     = new ClientEnLigne ();
ClientAuTéléphone  clientBAuTéléphone;
clientBAuTéléphone = new ClientAuTéléphone ();
ClientSurPlace     clientCSurPlace;
clientCSurPlace    = new ClientSurPlace ();
…
```