
Fiche TD N° 06

Exercice 01

Pilote automatique

On s'intéresse à un pilote automatique de voiture. Ce pilote doit adapter sa conduite (réaction aux obstacles, virages, etc.) aux conditions de circulation (pluie, neige, vent, nuit, etc.). Cela se traduit par des définitions différentes des méthodes traiter (obstacle), tourner (angle), etc.

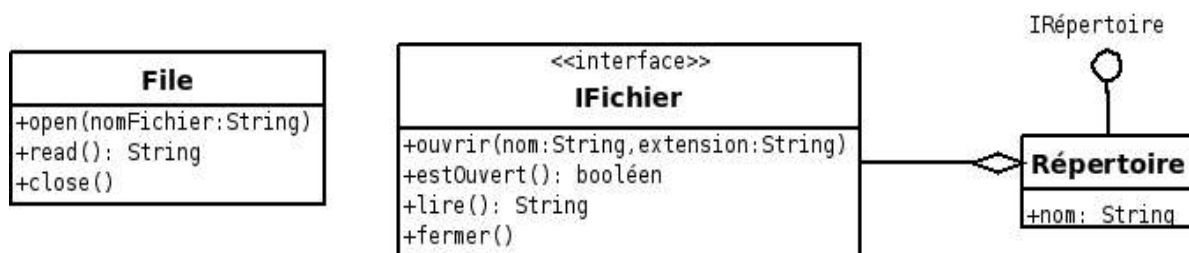
Des capteurs permettent à un composant de savoir dans quelles conditions la voiture évolue à tout instant. Ce composant peut se placer comme écouteur d'événements générés par les capteurs (par exemple de type Neige ou FinNeige), ou encore interroger chacun des capteurs régulièrement (par exemple via le prédicat neige du capteur de neige).

1) Quel(s) design(s) pattern(s) peut-on utiliser pour faire en sorte que le pilote automatique adapte son comportement en fonction des conditions ?

Exercice 02

Répertoires et fichiers

On dispose des classes présentées sur la figure suivante :



1) Une application est écrite en fonction de l'interface IRépertoire. On souhaiterait y intégrer une gestion de répertoires comprenant des fichiers manipulés via la classe File. Proposer une architecture permettant cela en maximisant l'utilisation du code déjà existant. On précisera comment peut être implémentée chaque nouvelle méthode introduite.

2) Modifier cette architecture pour prendre en compte une classe abstraite FichierAbstrait à la place de l'interface IFichier.

Solution

Exercice 01

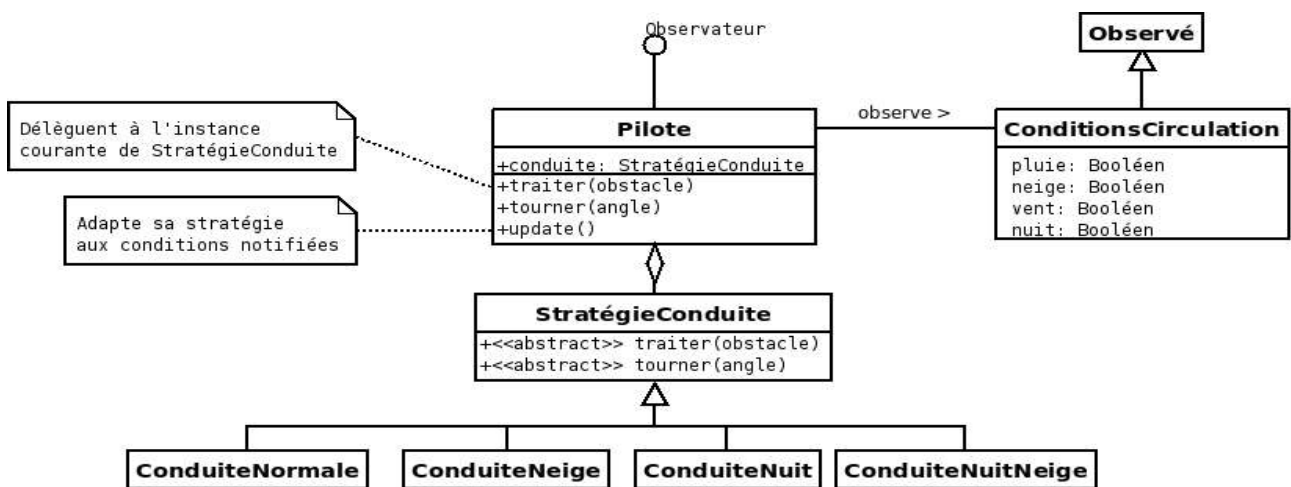
Pattern Stratégie

Problème résolu : permet à un objet de modifier dynamiquement l'implémentation d'un comportement (par exemple, d'algorithme pour un problème) sans changer de classe.

Solution : l'algorithme n'est pas implémenté dans la classe de l'objet. L'objet contient une stratégie, dont il peut changer dynamiquement l'instance concrète. Les différentes stratégies sont donc implémentées dans différentes classes concrètes.

On peut utiliser le design pattern stratégie : les comportements possibles du pilote automatique, en fonction des conditions de circulation, seront implémentés dans autant de sous-classes concrètes d'une classe abstraite. Le pilote changera alors sa stratégie courante en fonction des conditions.

Afin que le pilote soit maintenu au courant des changements de conditions, on pourra utiliser le pattern observateur/observé pour en faire un observateur du composant résumant les conditions. Ce composant notifiera ses observateurs à chaque fois qu'il changera d'état. On obtient alors l'architecture suivante :



Notons enfin que si de trop nombreuses combinaisons de conditions sont à envisager, et donc trop de stratégies, on peut envisager d'utiliser un pattern décorateur pour définir les différentes stratégies concrètes. Par exemple, la stratégie correspondant à la conduite de nuit par temps de neige pourra être obtenue dynamiquement en associant la conduite de nuit et la conduite par temps de neige.

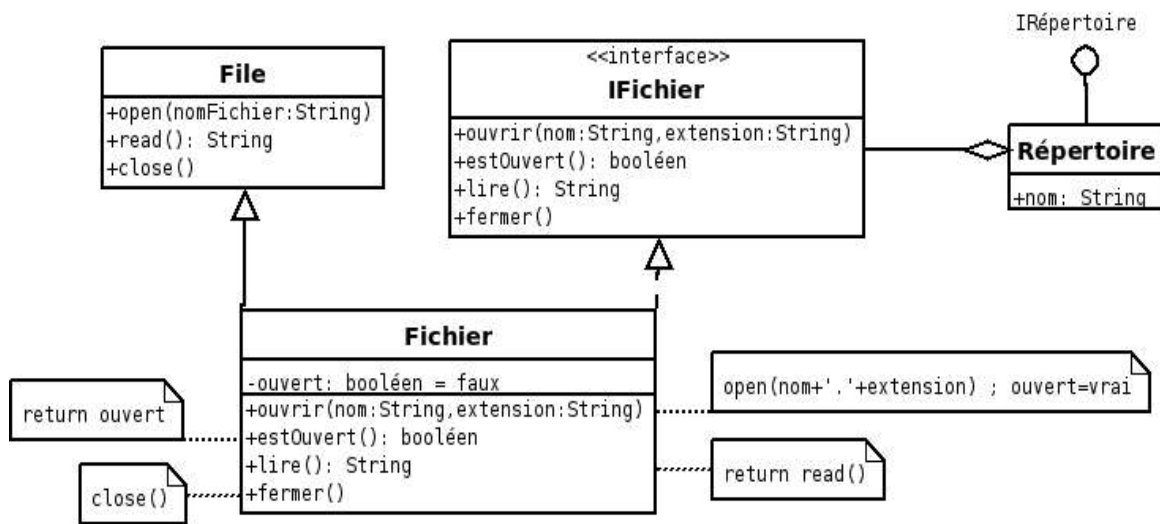
Exercice 02

Pattern Adaptateur

Problème résolu : permet d'« unifier » une interface requise par un client et l'interface d'une classe fournissant les services attendus. Autrement dit fournir une interface stable (Adaptateur) à un composant dont l'interface peut varier (Adapté).

Solution : ce motif se décline en deux variantes. L'adaptateur de classe consiste à faire hériter une nouvelle classe de la classe fournissant les services, en lui faisant implémenter l'interface requise tout en utilisant les méthodes de la surclasse. L'adaptateur d'objet consiste à écrire une nouvelle classe, implémentant l'interface requise et utilisant une instance du fournisseur par délégation.

1) En utilisant le pattern adaptateur pour « adapter » la classe `File` à l'interface `IFichier`, on peut utiliser l'architecture suivante :



Notons que l'on a utilisé ici un adaptateur de classe.

2) On peut utiliser exactement le même principe que précédemment, mais afin d'éviter l'héritage multiple on utilisera un pattern d'adaptateur d'instance (d'objet) : on introduira une classe **Fichier** héritant de la classe **FichierAbstrait** et utilisant une instance de la classe **File** par délégation.