TP2 : Création d'interfaces avec utilisation des Design Pattern Factory et Prototype

1 Le Design Pattern *Factory*

Présentation

Le *Design Pattern* ***Factory*** et de manière plus globale l'***Abstract Factory***, (fig.1). font partie de la famille des *Design Pattern* de Création.

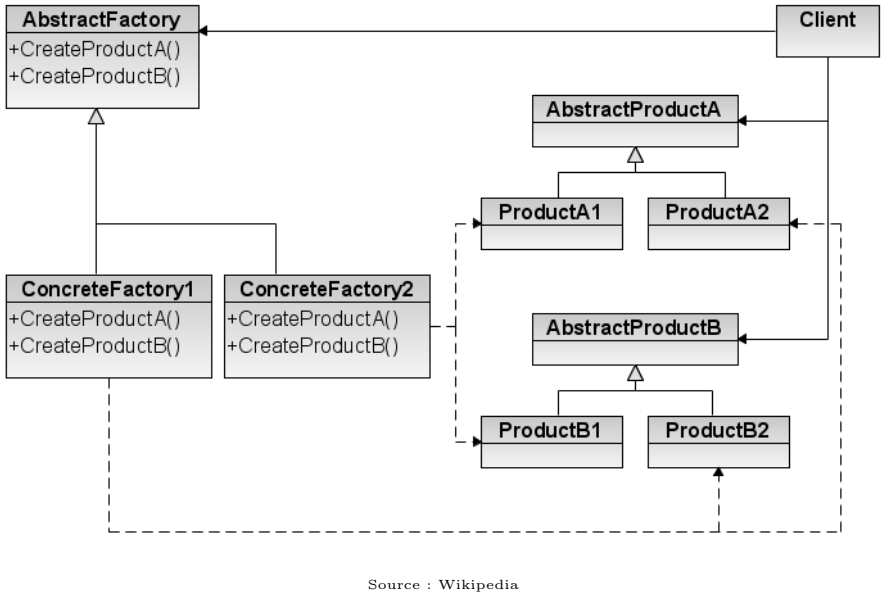

FIGURE 1 – Forme générale du Design Pattern Abstract Factory

Utilisation du Design Pattern *Factory* : exemple

Cas pratique

Prenons comme exemple une application de stockage des ouvrages d'une bibliothèque. Il existe plusieurs manière de stocker les informations qui seront utilisées par l'application : XML, texte brut, MySQL, Access, SQL server, etc. Si nous voulons que cette dernière puisse être utilisée pour plusieurs bibliothèques, cela signifie qu'elle doit permettre d'utiliser différents systèmes d'information et, à fortiori, permettre l'ajout du support d'un autre système d'information.

Schéma succin

Le schéma de la figure 4 présente l'architecture logicielle que nous aimerions avoir : une classe abstraite, `AbsDataManager`, contenant les prototypes des fonctions utilisées dans le reste de l'application. N'importe quelle autre classe de traitement des données peut être ajoutée en plus tant qu'elle implémente la manière d'effectuer chacune des fonctions données. Nous avons par ailleurs le *DataManagerFactory* qui est la *Factory* chargée de créer l'instance du DataManager que nous voulons utiliser.

Ici, nous avons mis comme exemple dans l'interface commune aux *DataManager* les fonctions les plus courantes pour ce type d'application :

Load() Charge les informations

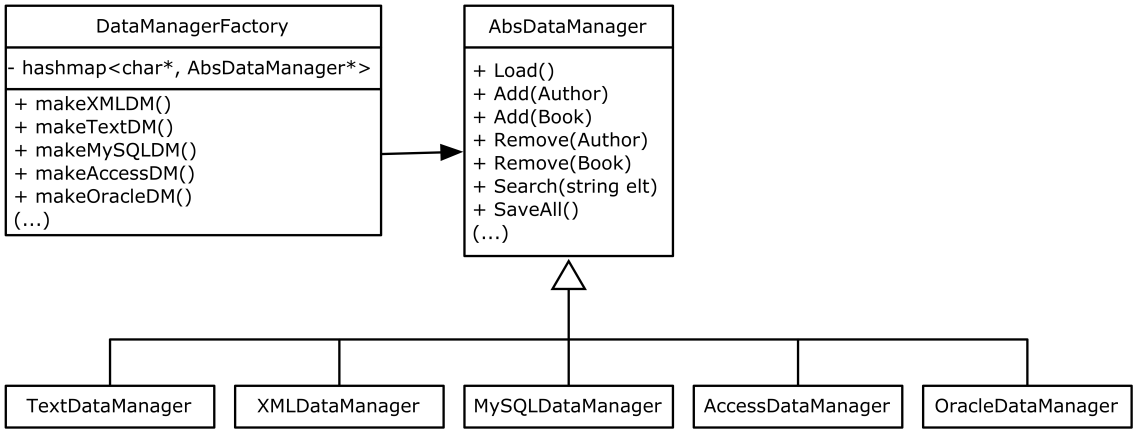

FIGURE 2 – Design Pattern Factory : Chargement de données

Add(Author) Ajoute un auteur.

Add(Book) Ajoute un livre

Remove(Author) Supprime un auteur

Remove(Book) Supprime un livre

Search(string elt) Cherche tout livre/auteur dont le nom contient la chaine de caractère "elt"

SaveAll() Sauvegarde tout le contenu du système d'information.
Selon le type de système d'information, il peut s'agir d'un ajout/modifications en base de donnée (Access, MySQL, Oracle) ou d'écriture dans des fichiers (XML, Text).

En ce qui concerne le *DataManagerFactory*, il contient une *hashtable* qui, pour chaque *type* de *DataManager*, appellera la bonne fonction *make* et renverra une erreur au cas où le *type* n'existe pas.

Et le passage à l'échelle dans tout ça ?

Que se passe t-il dans le cas où on veux ajouter un autre *DataManager*, par exemple pour SQLServer ? Il suffit d'ajouter une classe héritant de `AbsDataManager`, redéfinissant les fonctions nécessaires. Un ajout est nécessaire dans le code de la *Factory* pour gérer cette nouvelle classe concrète et elle peut ensuite être utilisée comme classe chargeant les données depuis l'application.
Dans le cas de l'interface graphique, que se passe-t-il lorsque si vous voulez ajouter une nouvelle interface, par exemple *OpenGL* ? Si vous voulez ajouter un nouveau *Widget* ?

2 Le Design Pattern *Prototype*

Présentation

Le *Design Pattern* ***Prototype*** , (fig.3). font partie de la famille des *Design Pattern* de Création.

Utilisation du Design Pattern *Prototype* : exemple

En reprenant l'exemple précédent, nous voulons pouvoir utiliser l'application sans connaître par avance la source de donnée que nous allons devoir traiter : XML, texte, SQL etc. Les informations seront données dans un fichier à part, chargé au lancement de l'application.

Ici, il n'y a plus de problèmes de modifications du code existant lors de l'ajout de nouvelles classes : lors de l'appel à clone(), la bonne fonction sera appelée et renverra une instance de la classe concrète ainsi construite.

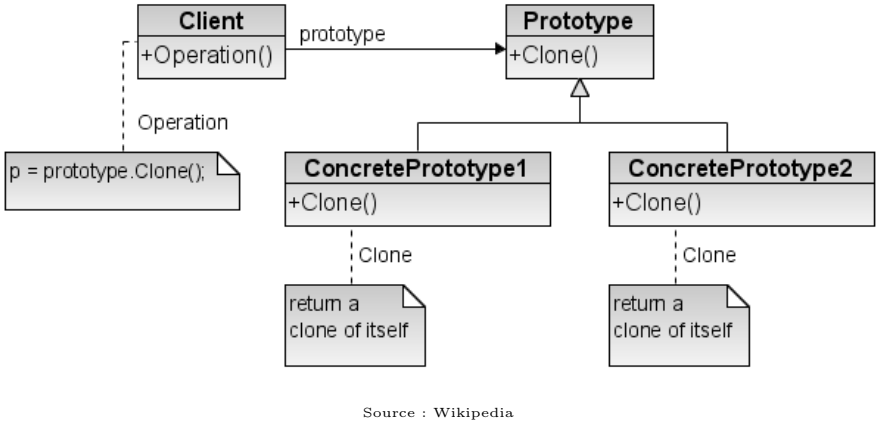

FIGURE 3 – Forme générale du Design Pattern Prototype

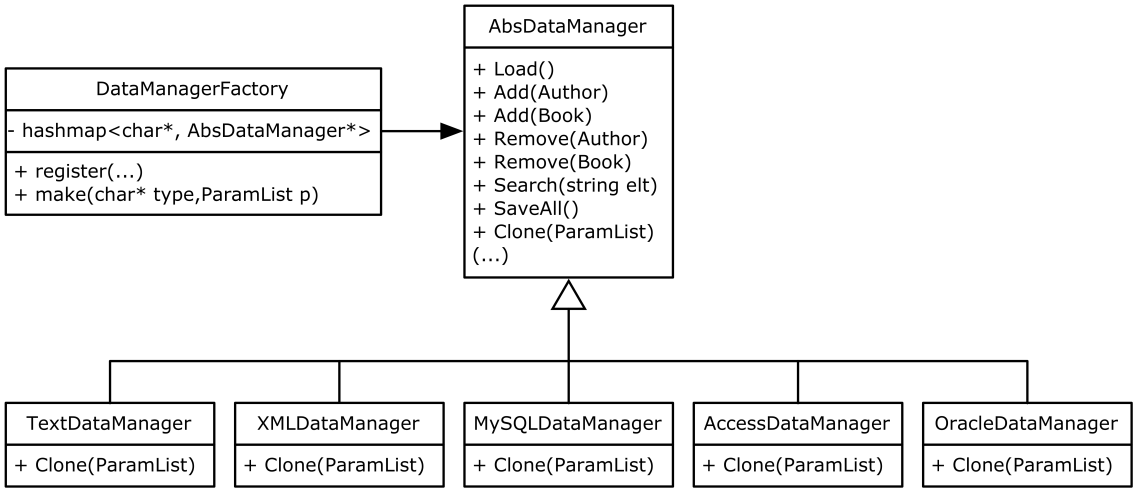

FIGURE 4 – Design Pattern Prototype : Chargement de données

Travail à rendre

Vous définirez une interface (ex. formulaire) dont vous donnerez le layout sous forme de schéma.
Rappel : on ne demande en aucun cas la gestion des évènements de clic de souris par exemple, tout ce qui nous intéresse ici est l'affichage de l'interface.

Partie 1 : Factory

Dans un premier temps, vous allez devoir implémenter les *Factory* pour la création des *Widgets* en SDL et en mode texte.
Vos fonctions auront un nom explicite et chaque *Widget* aura une chaîne d'héritage propre (ex : SDLButton et TxtButton héritent de *AbstractButton* qui contient la fonction make_button). Quant à l'*Abstract Factory*, elle contiendra une fonction *register* qui enregistrera les *Factory* dans une *hashtable*, ce qui permettra d'avoir l'ensemble des *Factory* présentes dans l'application et les fonctions permettant de les créé (make_sdl_interface etc.).

Partie 2 : Prototype

On vous demande ici de repartir du travail fait au TP précédent et d'y ajouter le *Design Pattern Prototype* pour gérer les *Widgets*. Les *Widget* seront désormais des *Prototype* qu'il sera possible de *cloner*.

Modalités de rendu

Vous devrez envoyer votre travail par mail à florence.laguzet@lri.fr et en copie à joel.falcou@lri.fr

(pensez à respecter le formatage de l'objet !!). Devront être présents : une archive du code source avec makefile, un document présentant l'analyse de l'implémentation effectuée des *Design Pattern* (avec mention explicite des modifications apportées pour le passage du TP1 au TP2) et un document présentant les choix éventuels d'implémentations. **Les deux premiers TP seront à rendre pour le dimanche 25 septembre 23 heure au plus tard (il faut dormir la nuit !).**

Annexe : Fonctionnement de la SDL

Pour implémenter la fenêtre en mode graphique, nous allons utiliser une bibliothèque externe nommée SDL (Simple DirectMedia Layer). Afin de faciliter l'utilisation de cette dernière, un code présentant les principales fonctions vous sera fourni.
Voici quelques notions de base concernant la SDL :

Initialisation

`SDL_Init(<Options>)` est la fonction permettant d'initialiser le moteur graphique et vérifier que toutes les options nécessaires fonctionnent correctement. Ici, nous n'utilisons que l'option `SDL_INIT_VIDEO` car nous n'avons besoin que de l'affichage. D'autres options sont disponibles pour, par exemple, permettre l'utilisation de sons (`SDL_INIT_AUDIO`), la gestion des manettes (`SDL_INIT_JOYSTICK`), etc.

Création de la fenêtre

En SDL, il est nécessaire de créer une surface correspondant à la fenêtre que nous allons utiliser, un genre de canevas sur lequel tout le reste sera collé. La fonction `SDL_SetVideoMode` initialise cette surface en renvoyant un pointeur sur cette dernière, de type `SDL_Surface*`.

Création de surfaces

Pour créer par exemple les boutons, il faut créer des surfaces, de type `SDL_Surface*`, afin de les coller sur la surface principale. C'est le travail de la fonction `SDL_BlitSurface` qui va dessiner le contenu d'une surface sur une autre.
Attention : lors de l'ajout de surfaces, la position du point haut gauche vous est demandé. En SDL, **le point en haut à gauche de la fenêtre possède les coordonnées (0,0)**. De plus, toute surface créée explicitement (hors la surface principale) **doit être explicitement effacée**.

Pour aller plus loin

La documentation complète de la SDL est disponible sur le site http ://www.libsdl.org/ et une quantité phénoménale de tutoriels sont à votre disposition sur le net.
Cependant, pensez à finir la partie concernant l'analyse et l'implémentation des Design Pattern avant de jouer avec les possibilités de la SDL ;-)