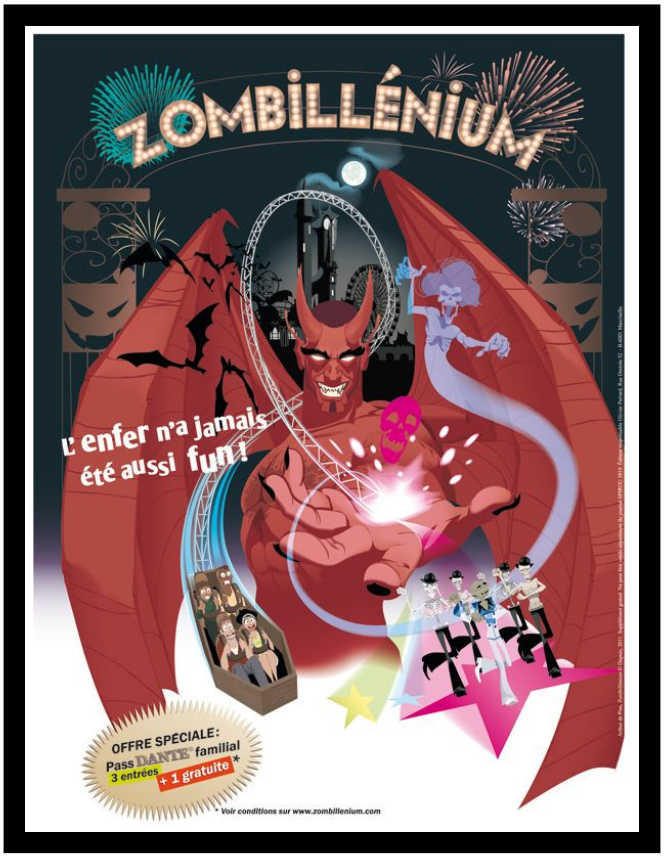
BONCOEUR Sébastien *TD-B*
PERIER Hugo

Problème Fil Rouge - Partie 2



Sommaire

I – Fonctionnement de la Gestion

La gestion a été réalisée dans la classe Administration. Certaines méthodes ont été implémentées dans des classes objets afin d'empêcher la modification d'attributs "délicats" dans la classe Administration. La classe Administration n'a pas été déclarée abstraite car nous l'utilisons comme un objet, certes unique.
En ce qui concerne l'encapsulation des données, la visibilité a été choisie conformément à ce que nous pensions le plus juste, le plus utile pour Mme Von Bloodt. Certains attributs sont accessibles en écriture (maintenance pour une attraction par exemple) dans le cas d'un plausible changement.

A) Du personnel

Deux classes héritent de la classe Personnel : *Sorcier* et Monstre. En effet, leurs attributs et leurs rôles sont très différents. La classe Monstre est également classe mère de 4 autres classes qui correspondent aux différents types de monstres.
En ce qui concerne notre code, l'ensemble du personnel est stocké dans une liste d'Administration : *list*. Par ailleurs, la gestion du personnel a nécessité la création de nouvelles listes : *list_neant* et list_parc. La *list_neant* contient l'ensemble des monstres ne pouvant être affectés à une attraction quelle qu'elle soit, tandis que *list_parc* contient l'ensemble des Monstres affectés au parc et non à une attraction en particulier.

Au détriment de l'espace mémoire, nous avons souhaité simplifier la gestion de l'exception pour les monstres ne pouvant être affectés à une attraction, d'où ces nouvelles listes.
En ce qui concerne le personnel en général, nous avons pu gérer le personnel de façon globale dans certaines situations (changement de fonction, modification de la cagnotte par exemple). Mais il a aussi fallu adapter cette gestion pour les types de personnel (*Sorcier* et *Monstre*).

B) Des attractions

Quatre classes filles correspondant aux types d'attraction du parc héritent de la classe Attraction. L'ensemble des attractions est stocké dans une liste d'Administration : *list_a*. Nous avons autorisé l'accès en écriture à certains paramètres d'une attraction uniquement (maintenance, ouverture, équipe ...).

C) Interactions Monstre-Attraction

L'utilité de la classe Administration est dans le fait qu'elle permet de traiter l'Équipe de monstres d'une attraction aussi bien que l'affectation d'un Monstre. Nous nous servons donc d'Administration pour faire communiquer les deux classes. Chacune délègue une partie de son activité à l'autre.

II - Méthodes implémentées (arguments d'entrées, sorties, fonction)

A) Lecture et écriture de fichier

La lecture du fichier .csv s'effectue grâce à la fonction **Lecture_CSV_initialisation** prenant en paramètres un pathfile de type string : il s'agit tout simplement du nom du fichier à lire situé dans le dossier debug du projet. La première lecture du fichier CSV réalisée avec un StreamReader (obtenu via System.IO) permet d'ajouter uniquement les attractions . Chaque ligne obtenue avec la fonction ReadLine est scindée avec la fonction Split sur le ';' car chaque case du document CSV est séparée par cela.

La fonction **Ajout_attraction** prend en paramètre un tableau de string contenant tous les éléments des cases d'une ligne du fichier CSV. Elle vérifie si la première case correspond bien à un des types Attraction (c'est à dire s'il s'agit d'une Boutique/Spectacle/DarkRide/RollerCoaster) et en crée un nouvel élément correspondant au type détecté avec les informations contenues dans le tableau de string. Ceci s'effectue si et seulement si l'Attraction (identifiée par son identifiant) n'est pas déjà présente dans la *list_a* de l'administration : verification grâce à la fonction **Attra_est_présente** qui balaye la *list_a* avec un foreach pour detecter la presence de la dite attraction.

De la même manière, la fonction **Ajout_personnel p**rend en paramètre un tableau de string contenant tous les éléments des cases d'une ligne du fichier CSV. Elle vérifie si la première case correspond bien à un des types Personnel (Sorcier/ Monstre/ Démon/ Fantôme/ Vampire/ Zombie/ Loup Garou) et en crée un nouvel élément correspondant au type détecté avec les informations contenues dans le tableau de string. Ceci s'effectue si et seulement si le *Personnel* (identifiée par son *matricule*) n'est pas déjà présente dans la *list_a* de l'administration : verification grâce à la fonction **Perso_est_present** qui balaye la *list* avec un foreach pour detecter la presence de la dite attraction.

Vient ensuite l'utilisation de la fonction **CSV_Affectation_Attraction**, prenant en entré le monstre dont on doit modifier l'affectation et un string contenant le contenu de la case du fichier CSV réservé à l'affectation (cette dernière peut être vide, contenir "neant" ou "parc" ou contenir l'identifiant de l'attraction correspondant à son affectation). Ainsi en fonction de ces info, on affecte null ou on retrouve l'attraction correspondante dans la list_a grâce à la fonction **TrouverAttraction** et on l'affecte au Monstre.

Enfin avec la fonction **RemplirEquipes** on créé les équipes de Monstre qui sont sur la même attraction.

B) Création d'objets et ajouts à la liste

L'implémentation devant se faire sans l'intervention humaine au sein de la console, on ne considère qu'un élément string "ligne" (pouvant être créé à la suite de plusieurs Console.Read) ce qui permet de réutiliser simplement les méthodes précédentes : **Ajout_personnel** et **Ajout_attraction**.

C) Evolution de caractéristiques d'objets

Comme nous l'écrivons plus tôt, certains attributs sont importants et susceptibles de changer au cours du temps. Nous avons donc réalisé des méthodes pouvant faire évoluer ces paramètres dans Administration. Nous recensons des attributs pour le personnel (fonction), puis plus spécifiquement pour les sorciers (*pouvoirs*, *grade*) ou pour les monstres (*affectation*). Il en est de même pour les attractions (*maintenance*, *ouverture*).

D) Recherche d'objets selon une/des caractéristiques

Nous employons l'opérateur « is » pour transtyper Personnel en *Monstre* et *Sorcier*.
Dans la boucle foreach nous vérifions ensuite par un booléen les conditions souhaitées avant de renvoyer l'objet correspondant.

E) Tri

Nous avons trié les objets en fonction d'une caractéristique grâce à la fonction « OrderBy » dans un souci de performance et d'efficacité du code.

L'implémentation d'une interface <IComparable> n'était pas nécessaire dans la mesure où les paramètres qui nous ont été données à trier étaient des paramètres numériques.

III – Gestion des erreurs

De multiples dispositifs try/catch (finally non-utile) de gestion des erreurs ont été mises en place :
- ligne 172 dans Administration pour éviter le cas où le type d'attraction proposé ne ferait pas partie des 4 types prédéfinis.
-ligne 471 dans Administration pour prendre en compte le cas où l'affectation souhaitée serait nulle.
-ligne 976 dans Administration pour ne pas voir de paramètres numériques illogiques. Par exemple la cagnotte qui ne doit pas descendre au-dessous de zéro.
(définition d'une propre classe d'exception : ExceptionCagnotteHorsLimite)
Nous n'en avons toutefois pas abusé car en trop grand nombre ils ne permettent pas de savoir précisément d'où vient l'erreur.

IV – Fonctionnement de la démonstration

Nous n'avons pas ajouté d'instances d'objets au csv préexistant. Nous utilisons ainsi simplement les données qui nous ont été données à traiter.

Les résultats de notre script sont présentés sous forme d'un menu déroulant dont l'utilisateur pourra vérifier l'exactitude en se déplaçant dans la console.

FIN