2018-2019

PROJET CDAA

DEVELOPPEMENT D'UNE APPLICATION POO

ALEXIS GUYOT
UNIVERSITE DE BOURGOGNE
Licence 3 Informatique – Gr TDB TP3

TABLE DES MATIERES

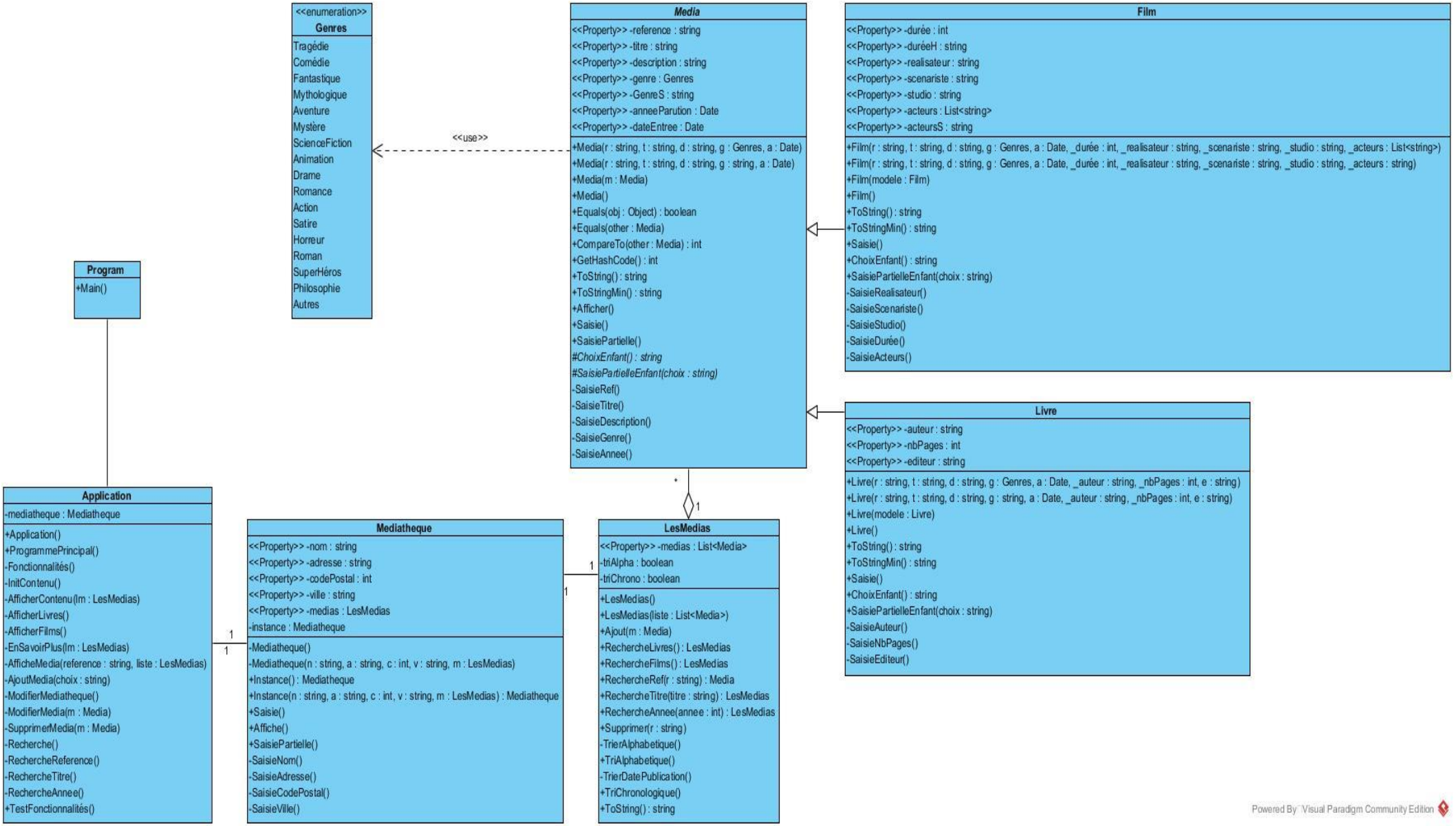
APPLICATION POO

Description de l'application

Dans le cadre de cette première partie du projet de CDAA, j'ai décidé de développer, en m'inspirant de l'application de gestion des inscriptions vue en TD/TP, une application de gestion des médias pour une médiathèque. Celle-ci gère deux types de médias : des livres et des films.

Contrairement à l'application développée en TD/TP, j'ai fait le choix de ne pas inclure de classe type « LesAssociations », qui m'aurait permis de gérer le contenu de plusieurs médiathèques depuis une seule et même application. En effet, je voyais vraiment mon programme comme un logiciel de gestion du contenu d'une médiathèque, pas comme un logiciel de gestion de médiathèques au sens large. Comme le sujet n'imposait pas de respecter à la lettre la structure de l'application de gestion des inscriptions, je me suis permis ce petit écart, mais j'en fait mention ici pour qu'on ne pense pas à un oubli par exemple. Pour concrétiser ce choix de conception, j'ai décidé de faire de ma classe Médiathèque un singleton. Il s'agit d'un design pattern qui permet à une instance d'être unique pendant toute l'exécution du programme. Ainsi, il y a toujours une et une seule médiathèque sur laquelle on travaille.

Voici la conception UML de mon programme.


<<enumeration>>
Genres
Tragédie
Comédie
Fantastique
Mythologique
Aventure
Mystère
ScienceFiction
Animation
Drame
Romance
Action
Satire
Horreur
Roman
SuperHéros
Philosophie
Autres
<<use>>
Media
<<Property>> -reference : string
<<Property>> -titre : string
<<Property>> -description : string
<<Property>> -genre : Genres
<<Property>> -GenreS : string
<<Property>> -anneeParution : Date
<<Property>> -dateEntree : Date
+Media(r : string, t : string, d : string, g : Genres, a : Date)
+Media(r : string, t : string, d : string, g : string, a : Date)
+Media(m : Media)
+Media()
+Equals(obj : Object) : boolean
+Equals(other : Media)
+CompareTo(other : Media) : int
+GetHashCode() : int
+ToString() : string
+ToStringMin() : string
+Afficher()
+Saisie()
+SaisiePartielle()
#ChoixEnfant() : string
#SaisiePartielleEnfant(choix : string)
-SaisieRef()
-SaisieTitre()
-SaisieDescription()
-SaisieGenre()
-SaisieAnnee()
Film
<<Property>> -durée : int
<<Property>> -duréeH : string
<<Property>> -realisateur : string
<<Property>> -scenariste : string
<<Property>> -studio : string
<<Property>> -acteurs : List<string>
<<Property>> -acteursS : string
+Film(r : string, t : string, d : string, g : Genres, a : Date, _durée : int, _realisateur : string, _scenariste : string, _studio : string, _acteurs : List<string>)
+Film(r : string, t : string, d : string, g : Genres, a : Date, _durée : int, _realisateur : string, _scenariste : string, _studio : string, _acteurs : string)
+Film(modele : Film)
+Film()
+ToString() : string
+ToStringMin() : string
+Saisie()
+ChoixEnfant() : string
+SaisiePartielleEnfant(choix : string)
-SaisieRealisateur()
-SaisieScenariste()
-SaisieStudio()
-SaisieDurée()
-SaisieActeurs()
Livre
<<Property>> -auteur : string
<<Property>> -nbPages : int
<<Property>> -editeur : string
+Livre(r : string, t : string, d : string, g : Genres, a : Date, _auteur : string, _nbPages : int, e : string)
+Livre(r : string, t : string, d : string, g : string, a : Date, _auteur : string, _nbPages : int, e : string)
+Livre(modele : Livre)
+Livre()
+ToString() : string
+ToStringMin() : string
+Saisie()
+ChoixEnfant() : string
+SaisiePartielleEnfant(choix : string)
-SaisieAuteur()
-SaisieNbPages()
-SaisieEditeur()
Program
+Main()
Application
-mediatheque : Mediatheque
+Application()
+ProgrammePrincipal()
-Fonctionnalités()
-InitContenu()
-AfficherContenu(lm : LesMedias)
-AfficherLivres()
-AfficherFilms()
-EnSavoirPlus(lm : LesMedias)
-AfficheMedia(reference : string, liste : LesMedias)
-AjoutMedia(choix : string)
-ModifierMediatheque()
-ModifierMedia(m : Media)
-SupprimerMedia(m : Media)
-Recherche()
-RechercheReference()
-RechercheTitre()
-RechercheAnnee()
+TestFonctionnalités()
Mediatheque
<<Property>> -nom : string
<<Property>> -adresse : string
<<Property>> -codePostal : int
<<Property>> -ville : string
<<Property>> -medias : LesMedias
-instance : Mediatheque
-Mediatheque()
-Mediatheque(n : string, a : string, c : int, v : string, m : LesMedias)
+Instance() : Mediatheque
+Instance(n : string, a : string, c : int, v : string, m : LesMedias) : Mediatheque
+Saisie()
+Affiche()
+SaisiePartielle()
-SaisieNom()
-SaisieAdresse()
-SaisieCodePostal()
-SaisieVille()
LesMedias
<<Property>> -medias : List<Media>
-triAlpha : boolean
-triChrono : boolean
+LesMedias()
+LesMedias(liste : List<Media>)
+Ajout(m : Media)
+RechercheLivres() : LesMedias
+RechercheFilms() : LesMedias
+RechercheRef(r : string) : Media
+RechercheTitre(titre : string) : LesMedias
+RechercheAnnee(annee : int) : LesMedias
+Supprimer(r : string)
-TrierAlphabetique()
+TriAlphabetique()
-TrierDatePublication()
+TriChronologique()
+ToString() : string
1
1
1
1
*
1
Powered By Visual Paradigm Community Edition

Spécification fonctionnelle

Voici les fonctionnalités développées dans l'application.

CREER LA MEDIATHEQUE

Lors du lancement de l'application, il est demandé à l'utilisateur de créer la médiathèque. Celui-ci doit rentrer certaines informations : le nom de la médiathèque et ses coordonnées (adresse, code postal et ville).

L'utilisateur peut ne pas remplir un champ s'il le souhaite en validant avec Entrée sans rien écrire. Dans ce cas, un message apparait pour lui indiquer qu'il pourra modifier cette valeur plus tard. Lorsque l'utilisateur entre un code postal dont le format ne correspond pas (utilisation de lettres, trop de chiffres, pas assez …), un message d'erreur est également affiché.

MODIFIER LES PARAMETRES DE LA MEDIATHEQUE

Après la création de la médiathèque, il est toujours possible pour l'utilisateur de modifier les informations qu'il a entrées au premier lancement. A partir du menu principal, il faut pour cela entrer le numéro « 7 ».

AFFICHER LES MEDIAS CONTENUS DANS LA MEDIATHEQUE

A partir de l'action « 1 », l'utilisateur peut afficher le contenu de la médiathèque. Tous les médias enregistrés sont affichés avec la version réduite de l'affichage (Référence, Titre, Date de parution, Auteur et Nombre de pages/Durée). Depuis cette interface, il peut demander l'affichage complet des informations sur un média en entrant sa référence et en validant avec Entrée. Si la référence ne correspond à aucun média enregistré dans la médiathèque, un message d'erreur est affiché. Si rien n'est entré au moment de la validation, l'utilisateur retourne au menu principal. Il est également possible d'ordonner les résultats.

Afficher la description complète d'un média

Comme indiqué ci-dessus, un utilisateur peut obtenir une fiche de description plus complète sur un média en entrant sa référence sur l'écran précédent. Depuis cet écran, il pourra obtenir les informations suivantes : Son titre, un résumé de l'œuvre, sa référence, son genre, sa date de parution, la date de saisie dans l'application et son auteur, son éditeur et son nombre de pages s'il s'agit d'un livre, ou son réalisateur, son scénariste, le studio de production, les acteurs principaux et la durée s'il s'agit d'un film.

Concernant ces attributs, plusieurs choix de conception ont été faits :

- Un média ne possède qu'un seul genre parmi ceux présents dans l'énumération suivante : Tragédie, Comédie, Fantastique, Mythologique, Aventure, Mystère, ScienceFiction, Animation, Drame, Romance, Action, Satire, Horreur, Roman, SuperHéros, Philosophie et Autres.
- Un film ne possède qu'un seul réalisateur, un seul scénariste et est produit par un seul studio de production. S'il y en a plusieurs, seuls les principaux pourront être ajoutés, mais les autres pourront être mentionnés dans le résumé de l'œuvre.

- La durée est exprimée en minutes.

Depuis cet écran, l'utilisateur peut choisir de modifier ou de supprimer le média qu'il découvre en entrant respectivement 1 ou 2 puis en validant avec Entrée. S'il valide sans rien entrer, le programme retourne au menu principal.

Trier la liste par ordre alphabétique

Une fois le contenu de la médiathèque affiché, l'utilisateur peut choisir d'ordonner les résultats pas ordre alphabétique. S'il tape « * » une première fois, la liste va se trier par ordre alphabétique des titres. Pour l'obtenir dans l'ordre inverse, il suffit d'entrer de nouveau « * » alors que le contenu est trié par ordre alphabétique.

Trier la liste par ordre chronologique

Un autre choix d'ordonnancement des résultats est l'ordre chronologique de publication. Pour obtenir cet ordre, l'utilisateur doit entrer « $ » et valider. Tout comme la fonctionnalité précédente, il est possible d'inverser les résultats en entrant cette même clé alors que la liste est déjà triée.

N'AFFICHER QUE LES FILMS / N'AFFICHER QUE LES LIVRES

Cette fonctionnalité est une spécification de la précédente. C'est-à-dire que toutes les sous-fonctionnalités présentées dans la partie précédente sont aussi valables ici. La différence est qu'en choisissant les actions 2 ou 3 depuis le menu principal, l'utilisateur peut choisir de ne travailler que sur la liste des livres ou que sur la liste des films.

RECHERCHER UN MEDIA (LIVRE OU FILM)

En choisissant l'action 6 depuis le menu principal, l'utilisateur peut rechercher dans la médiathèque un média particulier selon certains paramètres et afficher sa fiche description complète.

Tout d'abord, il peut décider d'accéder directement à un média en entrant sa référence. Il faut pour cela taper l'option 1 puis valider. Un nouvel écran apparait alors en demandant la référence à rechercher. Si celle-ci est trouvée dans la médiathèque, la description complète du média associé est affichée, sinon l'application indique qu'elle n'a pas trouvé de correspondance.

Ensuite, l'utilisateur peut décider d'effectuer sa recherche à partir d'un titre ou d'un morceau de titre. En effet, en choisissant l'option 2 et en validant, toujours avec Entrée, l'interface lui propose d'entrer un titre. Ce type de recherche ne retournera pas par défaut qu'un seul titre mais une version filtrée du contenu de la médiathèque dont le titre contient la recherche de l'utilisateur. C'est-à-dire que si celui-ci décide d'entrer « la » dans l'interface de recherche par titre, la fonctionnalité lui retournera tous les titres contenant l'expression « la » dans leur titre. Cependant, si la recherche ne retourne qu'un seul résultat, alors sa fiche description complète sera affichée directement.

Enfin, le dernier type de recherche proposé est par date de publication de l'œuvre à travers l'option 3. Seule l'année de parution est demandée. Si le format de la recherche effectuée par l'utilisateur ne correspond pas à celui d'une année (integer), alors un message d'erreur est affiché. Tout comme pour la

recherche précédente, celle-ci retourne une liste de médias parus l'année recherchée ou affiche directement le résultat sous forme d'une fiche description complète s'il n'y en qu'un seul.

AJOUTER UN FILM / UN LIVRE

En choisissant l'action 4 ou l'action 5 depuis le menu principal, l'utilisateur peut ajouter un nouveau film ou un nouveau livre à la médiathèque. Il devra alors remplir chaque champ. Tous les champs sont facultatifs (l'utilisateur peut laisser le champ vide) sauf la référence. Si la valeur du genre est vide ou qu'elle ne correspond pas à une proposition de la liste ci-dessus, alors la valeur attribuée sera « Autres ». La date de parution sera initialisée au 1er Janvier de l'an 1 si la valeur est laissée vide ou que le format ne respecte pas celui imposé (JJ/MM/YY [HH:MM:SS]). Les attributs « Titre » et « Description » sont quant à eux égaux à « N/A » pour « Not Available » par défaut. Un média ne pourra pas être ajouté à la médiathèque si sa référence est déjà présente dans la liste.

MODIFIER UN MEDIA

Lorsque l'utilisateur se retrouve face à la fiche description complète d'un média, il peut décider de la modifier en tapant 1 puis en validant. Dans ce cas, il peut choisir l'attribut de son choix puis entrer sa nouvelle valeur. Si l'utilisateur souhaite changer la référence d'un média mais que la nouvelle valeur correspond déjà à un autre média présent dans la médiathèque, alors la modification est refusée.

SUPPRIMER UN MEDIA

Toujours depuis la fiche description complète d'un média, un utilisateur peut supprimer l'œuvre courante. Pour cela, il lui suffit de taper 2 et de valider.

SERIALISER/DESERIALISER LA MEDIATHEQUE (PAS IMPLEMENTE)

Je fais mention de cette fonctionnalité car je l'ai envisagée pendant assez longtemps pour cette partie du projet. En effet, j'aurais aimé pouvoir enregistrer avant la fermeture de l'application l'état de la médiathèque pour pouvoir par la suite proposer à l'utilisateur de charger les données plutôt que de tout recréer à chaque fois. J'ai pour cela effectué quelques recherches sur Internet et j'ai notamment lu quelques documentations proposées par Microsoft sur le sujet (voir Bibliographie). Après lecture approfondie des méthodes existantes pour mettre en place cette fonctionnalité, j'ai finalement décidé de ne pas continuer à travailler dessus puisque j'avais un peu de mal à comprendre le fonctionnement des outils du C# pour ce concept.

Points techniques importants

CLASSE AFFICHAGE

La classe Affichage contient un ensemble de fonctions pour faciliter l'affichage dans la console : Centrer un texte, l'encadrer, appliquer un format (pour les erreurs notamment), … Cette classe et ses fonctions sont statiques car leur fonctionnement ne dépend absolument pas d'une instance particulière et qu'elles peuvent être appelées depuis n'importe quelle classe de l'application occasionnellement.

LES FONCTIONS ABSTRAITES DANS MEDIA

La classe Media est une classe abstraite qui correspond à la généralisation du contenu de la médiathèque. Elle contient deux fonctions abstraites, ChoixEnfant() et SaisiePartielleEnfant(). En utilisant ces fonctions, on peut ajouter à la fonction SaisiePartielle(), qui est commune à toutes les sous-classes de Media, des morceaux de code qui vont varier selon le type de média rencontré.

La fonction ChoixEnfant() va ainsi retourner une chaine de caractères correspondant aux attributs propres à la sous-classe (auteur, éditeur, nombre de pages, etc., pour un livre) pouvant être modifiés par la fonction. La fonction SaisiePartielleEnfant() va quant à elle s'occuper de modifier les attributs propres à la sous-classe si ceux-ci sont sélectionnés par l'utilisateur.

Puisqu'elles sont abstraites, ces fonctions devront obligatoirement être implémentées par toutes les sous-classes qui voudront être instanciables. Ainsi, le code de la modification des attributs communs à toutes celles-ci (ceux de la classe Media) est factorisé dans la classe mère, et seule la partie qui va varier d'un type de média à l'autre sera spécifiée à chaque fois.

GENERICITE

Description de Panier<T>

Panier<T> est une collection générique d'objets de type T. Elle contient trois attributs qui représentent le panier : un tableau « tab » qui contient les éléments de la collection, une constante « max » qui indique le nombre maximum d'éléments pouvant être ajoutés et une variable « nbe » qui représente le nombre d'éléments présents dans la collection. Nbe est différent de tab.Length car cette dernière indique la valeur de « max ». En effet, la variable « tab » est initialisée à une taille égale à cette constante, ce qui veut dire que max éléments sont préparés à l'initialisation pour accueillir des éléments. Les variables sont accessibles depuis l'extérieur grâce aux propriétés « **Items** » pour « tab » et « **Count** » pour « nbe ». En plus de tout cela, deux propriétés supplémentaires sont ajoutées pour représenter l'état du panier : « **Complet** », qui comme son nom l'indique vaut « true » lorsqu'il n'y a plus de place dans la collection et « false » le reste du temps, et « **Total** » qui vaut la valeur monétaire du panier, c'est-à-dire la somme des prix des produits présents dans le panier (en fonction de leur quantité). Panier<T> ne possède qu'un seul constructeur, celui par défaut, qui initialise le tableau en fonction de la valeur de « max ».

La collection possède deux fonctions d'ajout, **Ajout(T elem)** qui ajoute l'élément « elem » à la suite du dernier élément présent dans le tableau, et **Insertion(int i, T elem)** qui insère l'élément « elem » à la position indiquée par le paramètre « i ». A noter que l'ajout n'a lieu que si le panier n'est pas complet et que l'élément qu'on essaye d'ajouter n'est pas déjà présent dans le panier (pas de doublons).

Pour rechercher dans la collection, trois fonctions sont implémentées. **Appartient(T elem)** cherche si l'élément « elem » est présent dans le panier et retourne « true » si oui, « false » sinon. **ElementEnI(int i)** retourne l'élément présent à la position « i » dans le panier. Enfin la fonction **RechercheElem(T elem)** retourne la position de l'élément « elem » dans le panier s'il existe, -1 sinon.

La fonction **Supprimer(int i)** permet de supprimer l'élément à l'indice i dans la collection. Aucun « trou » n'est laissé dans la collection. C'est-à-dire quand un élément est supprimé, tous ceux qui le suivaient sont avancés d'un cran.

Le panier peut-être trié dans l'ordre croissant grâce à la fonction **Tri()** et dans l'ordre décroissant avec **TriInverse().** Enfin, il est possible d'afficher sur la console tous les éléments présents dans le panier grâce à la fonction **Affiche().**

Contraintes d'utilisation

Pour pouvoir utiliser correctement un Panier<T> avec un type donné T, celui-ci doit respecter certaines contraintes, imposées par la section « where ».

Tout d'abord, le type devra implémenter l'interface IEquatable<T>. En effet, sans cette condition les fonctions d'ajout, de tri et certaines de recherche seraient toutes inutilisables puisqu'elles se basent sur la comparaison de deux objets de type T pour déterminer s'ils sont égaux. Une telle comparaison est basée sur le comportement décrit dans la fonction Equals(T objetT), imposée par l'interface citée précédemment.

Ensuite, le type T devra implémenter l'interface IComparable<T>. En effet, les fonctions de tri utilisent la fonction CompareTo(), qui permet d'ordonner deux éléments de type T entre eux. Cette fonction est imposée par l'interface IComparable.

Enfin, le type T devra implémenter l'interface IValeurNum, qui est une interface créée spécialement pour le panier. Celle-ci impose à toutes les classes qui l'implémentent de posséder une fonction ValeurNumerique() qui correspond au prix du produit ajouté au panier. Cette fonction est utilisée pour évaluer la valeur du panier. Ceci implique aussi que tous les objets qui utilisent un panier possèdent d'une manière ou d'une autre une valeur qui lui est propre (monétaire, des points pour un jeu, …)

Usage du panier pour la classe ProduitAcheté

La classe ProduitAcheté décrit une commande faite pour un produit. Celui-ci est associé à un prix unitaire et à une quantité. A part cela, son implémentation respecte les différentes obligations apportées par les interfaces IEquatable, IComparable et IValeurNum. La valeur numérique d'un produit acheté est le prix total de la commande, soit son prix unitaire multiplié par sa quantité. La comparaison entre deux produits se fait sur leur référence.

COLLECTIONS

List<T>

La collection List<T> permet de stocker un ensemble homogène d'objets (des objets de même type, par polymorphisme ou non). La liste peut être ordonnée d'une manière précise (croissant/décroissant) mais par défaut ne l'est pas (le deuxième est simplement celui qui est ajouté après le premier). On accède à un élément dans la liste en précisant sa position dans celle-ci. Elle correspond bien à son nom puisque le moyen de le plus simple se représenter la collection est d'imaginer une liste dans la vraie vie. Pour cet exercice, j'ai pris l'exemple d'une liste d'étudiants mais la classe LesMedias de la première partie aurait pu faire un autre bon exemple également. Pour pouvoir utiliser comme on le souhaite les fonctions de tri ou de recherche dans la liste, il faut que les objets de type T implémentent les interfaces IEquatable et IComparable. Dans mon exemple de la liste des étudiants, la comparaison entre deux se fait par leur nom puis par leur prénom si ce dernier est identique.

SortedList

La collection SortedList permet de stocker un ensemble d'objets associés à une clé. La sortedList est un mélange entre le dictionnaire (avec un format clé/valeur) et la liste (qui s'accède par index et qui propose des fonctions de tri). On peut récupérer les objets grâce à leur position dans la SortedList mais aussi grâce à la valeur de leur clé. Les clés possèdent toutes le même type puisqu'elles doivent toutes pouvoir se comparer les unes avec les autres. En effet, la spécificité de la sortedList par rapport à la Hashtable ou au Dictionary, qui sont aussi des dictionnaires, est d'être toujours ordonnée par ordre croissant des clés. Celles-ci doivent donc par conséquent forcément implémenter l'interface IComparable<T>. L'accès aux données pouvant aussi se faire à travers la clé, il faut également qu'elles implémentent la fonction Equals de IEquatable<T>. Le type des valeurs peut en revanche changer d'un indice à l'autre et aucune contrainte sur celles-ci n'est imposée.

Plusieurs exemples peuvent bien représenter une sortedList. J'ai décidé dans le cadre de l'exercice de représenter un dictionnaire comme on l'entend chez le commun des mortels, c'est-à-dire une liste de mots associés à un ensemble d'informations (sa définition, sa nature, sa traduction en telle langue, …). J'étais au départ parti sur l'exemple de l'annuaire, où la clé n'était pas une chaine de caractères mais la classe Etudiant utilisée avec la List<T>. L'exemple était aussi intéressant (peut-être même plus que celui-ci d'un point de vue « pédagogique » puisque les clés n'étaient pas d'un type de base), mais je trouvais que l'intérêt de la sortedList était plus mis en avant avec l'exemple du dictionnaire. J'ai toutefois recyclé l'exemple de l'annuaire pour tester le Dictionary. Quoi qu'il en soit, il est surtout important de se souvenir que le type utilisé comme clé pour une sortedList doit pouvoir être trié facilement grâce à l'interface IComparable.

Hashtable

La Hashtable est une collection de type dictionnaire, c'est-à-dire qu'elle fonctionne sur le principe de l'association entre une clé et sa valeur. Les dictionnaires ne possèdent pas de fonctions de tri prédéfinies comme les listes et l'accès aux valeurs se fait à travers la clé. La spécificité de la Hashtable est que les clés

de la collection peuvent posséder des types différents. C'est-à-dire qu'une entrée de la table peut posséder une clé d'un type T1, et la suivante une clé d'un type T2. Puisque l'accès se fait par clé, celles-ci doivent implémenter la fonction Object.Equals() pour pouvoir comparer deux clés.

J'ai fait le choix dans cet exercice de représenter la hashtable à travers un inventaire de jeu. Dans celui-ci, on doit pouvoir placer des objets de types très différents, ici les clés, et chaque objet est associé à une fiche de description qui doit pouvoir être affichée quand l'utilisateur sélectionne l'objet.

Dictionary<TKey, TValue>

Comme son nom l'indique, la collection Dictionary<TKey, TValue> est une collection de type dictionnaire, où une clé est associée à une valeur. Ici, le type des clés et le type des valeurs sont imposés dès la création du dictionnaire. Comme pour la hashtable, le dictionary ne possède pas de fonctions de tri et l'accès à ses valeurs se fait à travers les clés. On peut considérer le dictionary comme une hashtable où les types seraient fixes et identiques pour toutes les clés et toutes les valeurs (ce qui ne veut pas dire que le type de la clé doit être identique au type de la valeur).

L'exemple d'utilisation le plus basique pour moi du dictionary<tkey, tvalue> est d'utiliser la collection lorsqu'on travaille sur des objets qui ne sont pas des types de base et une interface graphique qui elle travaille essentiellement avec des chaines de caractères. Cela revient à ajouter une sorte de libellé à l'objet pour pouvoir avoir accès à celui-ci facilement en entrant une chaîne de caractères. Dans le cadre de l'exercice, j'ai utilisé l'exemple d'un annuaire comme je l'ai déjà mentionné plus tôt. Un étudiant, avec toutes les informations qui lui sont propres (son nom, son prénom, son adresse, son numéro d'étudiant …) est associé à un numéro de téléphone (on aurait aussi pu utiliser une adresse mail par exemple). Ici, la classe Etudiant implémente l'interface IEquatable, ce qui permet de comparer facilement deux étudiants en cas de recherche.

BIBLIOGRAPHIE

Partie I :

Tri à bulles : https://www.gladir.com/CODER/CSHARP/tribubble.htm

Sérialisation : https://docs.microsoft.com/fr-fr/dotnet/framework/wcf/feature-details/serialization-and-deserialization

Sérialisation : https://docs.microsoft.com/fr-fr/dotnet/csharp/programming-guide/concepts/serialization/

Partie II :

Type générique : https://openclassrooms.com/fr/courses/2818931-programmez-en-oriente-objet-avec-c/2819081-les-generiques

Génériques : https://docs.microsoft.com/fr-fr/dotnet/csharp/programming-guide/generics/

Contraintes génériques : https://docs.microsoft.com/fr-fr/dotnet/csharp/language-reference/keywords/where-generic-type-constraint

Partie III :

Documentation List<T> : https://docs.microsoft.com/fr-fr/dotnet/api/system.collections.generic.list-1?view=netframework-4.7.2

Documentation SortedList : https://docs.microsoft.com/fr-fr/dotnet/api/system.collections.sortedlist?view=netframework-4.7.2

Documentation Hashtable : https://docs.microsoft.com/fr-fr/dotnet/api/system.collections.hashtable?view=netframework-4.7.2

Hashtable : https://www.dotnetperls.com/hashtable

Hashtable : http://www.tutorialsteacher.com/csharp/csharp-hashtable

Documentation Dictionary : https://docs.microsoft.com/fr-fr/dotnet/api/system.collections.generic.dictionary-2?view=netframework-4.7.2

Dictionary : https://www.dotnetperls.com/dictionary

Dictionary : http://www.tutorialsteacher.com/csharp/csharp-dictionary