Programmation objet II	**420-216-FX**
Travail pratique #2 – Partie #2	**90% du TP2 (16%)**
Prof.: Karine Filiatreault	***Hiver 2016***

Travail en équipe de deux. Le professeur doit approuver les équipes.

Date de remise:

- ***Au plus tard dimanche le 24 avril 2016 avant la fin de la journée sur LÉA.***

Objectif

L'objectif de ce travail est de construire une **application de gestion classique** tout en mettant en pratique les nouveaux concepts appris dernièrement soient les applications *Windows* multi-formulaires, la gestion des exceptions, la validation, les listes, le filtrage (formulaire de recherche), le tri, etc. L'application graphique à construire devra permettre de faire des **recherches avancées de cartes** du **jeu *Hearthstone*** et de **construire des *decks*** personnalisés (*Deck Builder*).

Fonctionnalités offertes par l'application

- L'application doit utiliser un **minimum de deux formulaires** et il doit y avoir du **passage d'information entre ces formulaires**.
- **Au démarrage**, les données sur les **héros** et les **cartes de *Hearthstone*** doivent être chargées dans des tableaux d'objets, et ce, à partir du fichier XML fourni. Attention : aucun *deck* ne doit être chargé à ce moment.
- Le **formulaire principal** devra permettre d'effectuer une **recherche avancée de cartes** en fonction de plusieurs critères (des spécifications précises sont données dans une section ci-dessous). Les cartes correspondant aux critères de recherche doivent être **affichées** soit sous forme **tabulaire** (en utilisant un *DataGridView*) ou bien à l'aide d'**images**; l'utilisateur doit pouvoir choisir entre ces deux modes d'affichage.
- Fonctionnalités liées à la gestion des *decks* :
  - **Créer un nouveau *deck*** en respectant les règles de conception décrites plus loin :
    - **Choisir un héros** pour le *deck*.
    - **Donner un nom** au *deck*.
    - **Ajouter des cartes** dans le *deck* tout en précisant la quantité.
  - **Enregistrer un *deck*** dans un fichier au **format XML**. Les *decks* doivent être enregistrés dans le dossier « *decks* » du répertoire courant. Le nom du fichier doit être créé automatiquement en fonction du nom du *deck* (ne pas utiliser une boîte de dialogue de *Windows* pour l'enregistrement). Si le *deck* existait déjà (avec le même nom), l'ancienne version est écrasée.
  - **Charger un *deck*** en utilisant une boîte de dialogue standard de *Windows* (*OpenFileDialog*).
  - **Modifier un *deck*** :
    - **Modifier** son nom.
    - **Ajouter une nouvelle carte** dans le *deck*.

        - **Supprimer toutes les copies d'une carte** dans le *deck*.
        - **Modifier la quantité d'une carte** dans le *deck*.
        - Note : Il n'est pas possible de changer le héros du *deck*.
    - **Visualiser un *deck*** soit sous forme **tabulaire** ou bien à l'aide d'**images** (de manière similaire au résultat de la recherche de cartes).
    - **Faire une recherche de cartes** en limitant automatiquement celle-ci aux cartes pouvant être incluses dans le *deck* et, à partir du résultat de la recherche, permettre **l'ajout** d'une carte dans le *deck* si c'est encore possible (nombre maximal de copies non atteint).
    - Note : À tout moment, il ne peut y avoir qu'un seul ***deck* chargé en mémoire**.

Règles de conception d'un *deck*

- Un *deck* est constitué d'un **nom** (composé d'un minimum de 3 caractères sans considérer les espaces superflus) déterminé par l'utilisateur, d'un **héros** et d'une **liste d'au plus 30 cartes**.
- Les cartes pouvant être incluses dans un *deck* doivent être soit de la **même classe** que celle du **héros**, soit de la classe **neutre**.
- Il peut y avoir un maximum de **deux copies** d'une même carte dans un *deck* sauf pour les cartes **légendaires** où le nombre maximal de copies est limité à **un**. Vous devez tenir compte du fait que les règles liées au nombre maximal de copies peuvent éventuellement changer; par contre, vous pouvez supposer que pour toutes les raretés sauf légendaire, le nombre maximal permis sera toujours le même.

Critères pour la recherche de cartes

- La méthode « **RechercherCartes** » de la classe « **Hearthstone** » accepte les paramètres suivants dans l'ordre :
    - **`type`** (`CarteType`) : Le type des cartes recherchées.
    - **`nomPartiel`** (`String`) : Le nom partiel des cartes recherchées sans considérer la casse et les espaces superflus. La valeur `null` ou bien une chaîne vide indique qu'il faut ignorer ce critère. Par exemple, une recherche avec le nom partiel « mani » doit retourner les deux cartes suivantes :
        - Amani Berserker
        - Faceless Manipulator
    - **`extensions`** (`List<CarteExtension>`) : La liste des extensions auxquelles les cartes recherchées doivent appartenir. La valeur `null` ou bien la liste vide indique qu'il faut ignorer ce critère.
    - **`rarete`** (`CarteRarete`) : La rareté des cartes recherchées.
    - **`coutMin`** (`sbyte`) : Le coût minimal des cartes recherchées.
    - **`coutMax`** (`sbyte`) : Le coût maximal des cartes recherchées.
    - **`textePartiel`** (`String`) : Les différents mots partiels (séparés par des espaces) qui doivent tous être présents dans le texte des cartes recherchées, et ce, sans considérer leur position, la casse et les espaces superflus. La valeur `null` ou bien une chaîne vide

indique qu'il faut ignorer ce critère. Par exemple, une recherche avec le texte partiel « **Rand add CA**» doit retourner les cinq cartes suivantes :

- Webspinner : Texte = « Deathrattle: **Add** a **rand**om Beast **ca**rd to your hand. »
- Lock and Load : Texte = « Each time you cast a spell this turn, **add** a **rand**om Hunter **ca**rd to your hand. ».
- Burgle : Texte = « **Add** 2 **rand**om class **ca**rds to your hand <i>(from your opponent's class)</i>.»
- Gazlowe : Texte = « Whenever you **ca**st a 1-mana spell, **add** a **rand**om Mech to your hand.»
- Grand Crusader: Texte = « Battlecry: **Add** a **rand**om Paladin **ca**rd to your hand.»

- **`classe`** (`HerosClasse`) : La classe de héros des cartes recherchées.
- **`mecaniques`** (`List<CarteMecanique>`) : La liste des mécaniques que les cartes recherchées doivent avoir; une carte doit avoir toutes les mécaniques de la liste pour être retenue. La valeur `null` ou bien la liste vide indique qu'il faut ignorer ce critère.
- **`attaqueMin`** (`sbyte`) : L'attaque minimale des cartes recherchées.
- **`attaqueMax`** (`sbyte`) : L'attaque maximale des cartes recherchées.
- **`vieMin`** (`sbyte`) : La vie minimale des cartes recherchées.
- **`vieMax`** (`sbyte`) : La vie maximale des cartes recherchées.
- **`race`** (`ServiteurRace`) : La race de serviteur des cartes recherchées.
- **`durabiliteMin`** (`sbyte`) : La durabilité minimale des cartes recherchées.
- **`durabiliteMax`** (`sbyte`) : La durabilité maximale des cartes recherchées.

- Si **au moins un critère de recherche** est spécifié mais qu'aucune carte ne correspond aux critères, la méthode « RechercherCartes » doit retourner une **liste vide**, et non pas `null`.
- Si **aucun critère de recherche** n'est spécifié, la méthode « RechercherCartes » doit retourner **`null`**, et non pas une liste vide.
- Si un critère de recherche pour un **type numérique** quelconque (`coutMin`, `coutMax`, etc.) ne doit pas être considéré, on utilisera la **valeur -1** comme paramètre de la méthode « RechercherCartes »
- Si un critère de recherche pour une **énumération** quelconque (`CarteType`, `CarteRarete`, etc.) ne doit pas être considéré, on utilisera la valeur suivante comme paramètre de la méthode « RechercherCartes » :

```
(NomEnum) (-1)
```

Modèle de données (diagramme de classes UML)

- Le **diagramme de classes UML** est donné à l'**annexe I**. Vous devez bien sûr respecter ce diagramme à la lettre.
- Les tests unitaires fournis vous permettront de découvrir certains détails importants quant à l'implémentation des classes. La **compréhension des tests unitaires** fait partie intégrante de la réalisation de ce travail.

Précisions sur le modèle de données

Précisions sur certaines **propriétés** de la classe « **Carte** » :

- **Identifiant** : Chaîne (`String`) respectant le format suivant dans l'ordre :
  - Entre 2 et 6 caractères parmi les lettres (minuscules et majuscules) et les chiffres (de 0 à 9) ;
  - un caractère de soulignement, soit "_" ;
  - entre 1 et 3 chiffres (de 0 à 9) ;
  - une seule lettre minuscule optionnelle.
- **Nom** : Chaîne (`String`) contenant un minimum trois caractères sans les espaces superflus.
- **Coût** : Entier (`byte`) entre **0 et 20** inclusivement.
- **Texte** : Chaîne (`String`) sans aucune restriction (« `null` » est accepté ainsi que chaîne vide). Les espaces superflus doivent toutefois être retirés.
- **Mécaniques** : Liste générique de « `CarteMecanique` ». Si une carte n'a aucune mécanique, cette liste doit être vide et non pas à « `null` ».
- **Points d'attaque** : Entier (`sbyte`) entre **0 et 12** inclusivement pour les cartes de type serviteur et arme. Pour les carte de type sort, la valeur doit obligatoirement être **-1**.
- **Points de vie** : Entier (`sbyte`) entre **1 et 15** inclusivement pour les cartes de type serviteur seulement. Pour les autres types de carte (sort et arme), la valeur doit obligatoirement être **-1**.
- **Race** : La valeur « `ServiteurRace.Aucune` » doit obligatoirement être utilisée pour les serviteurs n'ayant pas de race ainsi que pour les cartes qui ne sont pas de type serviteur.
- **Durabilité** : Entier (`sbyte`) entre **1 et 8** inclusivement pour les cartes de type arme seulement. Pour les autres types de carte (serviteur et sort), la valeur doit obligatoirement être **-1**.

- Précisions sur certaines **propriétés** de la classe « **Heros** » :
  - **Identifiant** : Chaîne (`String`) respectant le format suivant dans l'ordre :
    - La séquence "HERO_" ;
    - exactement 2 chiffres (de 0 à 9) ;
    - une seule lettre minuscule optionnelle.
  - **Nom** : Chaîne (`String`) contenant un minimum de trois caractères sans les espaces superflus.
  - **Classe** : Un héros ne peut pas avoir la classe neutre.
  - **Points de vie** : Entier (`byte`) entre **10 et 100** inclusivement.
- Précisions sur certaines **propriétés** de la classe « **Hearthstone** » :
  - **LesHeros** : Tableau de tous les héros (ne pas utiliser une liste ici).
  - **LesCartes** : Tableau de toutes les cartes (ne pas utiliser une liste ici).

- Précisions sur certaines **méthodes** de la classe « **Hearthstone** » :
  - **RechercherHeroParId** : Prend en paramètre l'identifiant d'un héros et retourne le héros correspondant si l'identifiant est valide; retourne « `null` » si l'identifiant est invalide.
  - **RechercherCarteParId** : Prend en paramètre l'identifiant d'une carte et retourne la carte correspondante si l'identifiant est valide; retourne « `null` » si l'identifiant est invalide.
  - **RechercherCartes** : Prend en paramètre tous les critères de recherche (vous référez à la section « Critères pour la recherche de cartes » plus haut pour l'ordre des paramètres et leur signification). Retourne une **liste générique** de « **`Carte`** » correspondant aux critères de recherche et **triée** premièrement selon le **coût** (en ordre croissant) et, par la suite, selon le **nom** (en ordre croissant et insensible à la casse et aux accents).
- Précisions sur certaines **propriétés** de la classe « **Deck** » :
  - **Nom** : Chaîne (`String`) contenant un minimum de trois caractères sans les espaces superflus.
  - **Heros** : Ne peut pas être à nul.
  - **LstCartesAvecQt** : Liste générique de « **`DeckEntree`** ».
- Précisions sur certaines **méthodes** de la classe « **Deck** » :
  - **AjouterCartes** : Prend en paramètre la carte à ajouter ainsi que le nombre de copies de cette carte (ne retourne rien). Les validations nécessaires doivent être effectuées (référez-vous aux tests unitaires).
  - **RetirerCarte** : Prend en paramètre la carte à retirer ainsi qu'un booléen indiquant si toutes les copies de la carte doivent être retirées (`true`) ou bien seulement une copie (`false`). Retourne le nombre de copies de la carte qui ont été retirées du *deck*. Les validations nécessaires doivent être effectuées (référez-vous aux tests unitaires).

Directives spécifiques

- Toutes les **données** nécessaires pour ce travail proviennent du **fichier** « **cards-collectible.xml** ». Ce fichier contient l'information sur les **cartes** (serviteurs, sorts et armes) ainsi que celle sur les **héros**. Ce fichier doit être incorporé aux **deux projets** de votre solution (projet pour l'application Windows et projet de tests unitaires) dans les dossiers « bin/Debug ». En aucun cas, vous ne devez modifier ce fichier.
- Les **images** pour les **cartes** et les **héros** doivent être téléchargées directement d'Internet à partir de l'URL ci-dessous (`{idCarteHeros}` doit être remplacé par l'identifiant de la carte ou bien du héros). Vous pouvez utiliser la version française ou anglaise pour les images des cartes :
  - Version française :

```
http://wow.zamimg.com/images/hearthstone/cards/frfr/original/{idCarteHeros}.png
```

  - Version anglaise :

```
http://wow.zamimg.com/images/hearthstone/cards/enus/original/{idCarteHeros}.png
```

- Lorsque nécessaire, vous devez **valider** les données dans les **interfaces graphiques**. Pour l'affichage des messages d'erreur vous **ne devez pas utiliser de *pop-up* (MessageBox)**. Les messages doivent être affichés directement dans des étiquettes (**Label**) à proximité des champs en erreur et avoir la couleur **rouge**. De plus, lors de la validation d'un formulaire, **tous les**

messages d'erreur doivent être **affichés en même temps** pour les champs en erreur (et non successivement comme dans les exemples et laboratoires fournis en classe). Il est interdit d'utiliser les méthodes « `TryParse` » pour les conversions de chaîne vers nombre (vous devez utiliser les méthodes « **`Parse`** » avec des blocs **`try-catch`**).

- Vous devez **lancer des exceptions** pour empêcher que les données deviennent corrompues dans vos classes. Référez-vous aux tests unitaires pour les détails.
- Utilisez les **énumérations** qui vous sont fournies.
- Toutes les **interfaces** doivent être **codées en français**; l'anglais est permis seulement pour les **images** et les **mécaniques** des cartes. Ainsi, vous devrez utiliser les descriptions associées aux valeurs des énumérations pour générer vos interfaces (utiliser la classe fournie « `UtilEnum` » à cette fin).
- Tout le **code** doit être en **français**.
- Toutes les **classes de tests unitaires** qui vous sont **fournies** doivent être intégrées à votre solution dans un nouveau projet de tests unitaires que vous devez ajouter vous-même. Bien sûr, votre code doit **passer tous les tests**.
- Vous devez utiliser des **expressions régulières** lorsque c'est approprié.

Directives générales

- Vous devez respecter les **conventions** fournies par le professeur pour tous les **identificateurs en C#** (variables, méthodes, classes, contrôles de formulaire, etc.) ainsi que les autres conventions de codage du cours Programmation Objet II (420-216-FX). Voir le fichier "**Conventions pour les identificateurs en C#.pdf**" et "**Normes de codage C#.pdf**". De plus, tout identificateur doit avoir **un nom significatif** considérant le contexte dans lequel il apparaît.
- Tous les **contrôles de formulaire doivent être renommés**; il ne faut pas conserver les noms par défaut tels "label1", "button1". N'oubliez pas de **renommer** aussi les **formulaires** (ne pas utiliser les noms de classe par défaut) et de leur donner un titre.
- Vous devez mettre des **commentaires en format XML** pour tous les éléments suivants : **classes, énumérations, constantes, attributs, constructeurs, propriétés et méthodes**. À noter que pour les classes correspondant à des formulaires, il n'est pas nécessaire de mettre des commentaires XML pour les méthodes liées aux événements dans l'interface graphique.
- À **l'intérieur** des **méthodes**, vous devez **ajouter des commentaires** pour expliquer clairement ce que fait chaque section de code assez complexe. Entre autres, toutes les **variables locales** doivent être accompagnées d'un commentaire expliquant son rôle si le nom de la variable n'est pas suffisamment clair.
- Vous devez vous assurez qu'il n'y ait pas **d'erreurs** (*errors*) selon ***ReSharper*** et la configuration fournie par le professeur. De plus, vérifiez minutieusement les **avertissements** (*warnings*) car il ne devrait presque pas y en avoir. Exécutez la commande suivante : *ReSharper/Inspect/Code Issues in Solution*.
- Les classes doivent toutes être présentées à l'aide du **gabarit de fichier** (*File Layout*) de ***ReSharper*** (Ctrl-E, F) ou (Ctrl-Alt-F).

Précisions sur l'évaluation

L'évaluation portera sur les éléments suivants :

1. (**10 %**) **Modèle objet** proposé (partie 1).
2. (**35 %**) Bon fonctionnement des **tests unitaires** sur les classes :
   - ✓ (20%) Tests unitaires pour les cas normaux.
   - ✓ (15%) Tests unitaires pour les exceptions.
3. (**30 %**) Bon **fonctionnement de l'application** et **interfaces** graphiques.
4. (**15 %**) Codage correct des **classes** et des **formulaires**.
5. (**10%**): **Qualité du code** :
   - ✓ Erreurs et avertissements selon *ReSharper*.
   - ✓ Respect des normes de codage.
   - ✓ Noms significatifs pour les identifiants (en français).
   - ✓ Noms pour les contrôles des formulaires (avec préfixe).
   - ✓ Commentaires XML
   - ✓ Commentaires à l'intérieur des méthodes.
   - ✓ Présentation du code (gabarit *ReSharper*, structure, présentation, etc.)

Outre les éléments à réaliser dans le cadre du travail, les aspects suivants viendront influencer la note finale:

- Retard.
- Remise incorrecte du travail.
- Français.
- Contribution insuffisante au travail d'équipe.
- D'autres éléments selon le besoin.

À remettre

- **Aucune remise papier.**
- Créez le **dossier "TP2_Prenom1_NomFamille1_Prenom2_NomFamille2"** (utilisez vos prénoms et vos noms de famille complets) dans lequel vous devez copier votre dossier pour le projet C# qui doit inclure le **fichier solution (.sln)**. Vérifiez que tout fonctionne bien avant de le remettre en copiant le dossier et en tentant d'ouvrir votre projet à partir du fichier solution copié.

  Note : Il est essentiel de créer ce dossier car autrement je ne peux pas savoir qui est votre coéquipier.

  Compressez le dossier en format ".**zip**". Utilisez un autre format de compression uniquement s'il vous est impossible de le faire en format ".zip".

  Remettez le dossier compressé par LÉA.

Bon travail !

Annexe I : Diagramme de classes UML