Haute Ecole de la Province de Liège

Programmation .NET et C#

Enoncé de laboratoire Phases I, II et III

2017-2018

François Caprasse
Cécile Moitroux
Alfonso Romio

Table des matières

1 Introduction

Le laboratoire de programmation orientée objet vise à mettre en pratique les différents concepts montrés dans le cadre du cours de théorie.

Il est découpé en trois parties. La première, propose la création d'un ensemble de classes manipulées dans le cadre d'une application console de test. Les deux suivantes sont destinées à la création d'applications fenêtrées.

Echéance des évaluations :

	Sujet	Echéance
Phase I	Développement de classes et outils de base	Semaine du **26/2/2018**
Phase II	Application Winform MyPaint	Semaine du **26/3/2018**
Phase III	Application WPF Home Energy Survey	**Evaluation de juin 2018**

Les fonctionnalités à développer et l'architecture à mettre en place sont précisées de façon très claire et sont à respecter. En cas de retard, des sanctions seront prises.

L'énoncé est à réaliser individuellement. A chaque évaluation, les étudiants délivreront, un fichier archive contenant :

- Les solutions (sans les fichiers exécutables)

Le nom du fichier et l'objet du mail seront obligatoirement constitués de la manière suivante :

Groupe Nom.rar

Celui-ci sera envoyé par mail à l'adresse du titulaire de laboratoire :
francois.caprasse@hepl.be ou
cecile.moitroux@hepl.be ou
alfonso.romio@hepl.be

au plus tard le matin de l'évaluation.

Les outils à utiliser pour les développements sont :

- Visual Studio 2015 minimum en **ANGLAIS**
- Framework .NET 3.5 au minimum

L'ensemble des outils seront installés et configurés **avant** la première séance de laboratoire.

2 Fonctionnalités Phase I (Semaine 4 : 26/2/2018)

Créer un ensemble de classes visant à caractériser des formes géométriques : carré, cercle et rectangle. Ces formes géométriques présentent des éléments communs, tels que le point d'accroche, ou le fait qu'elles puissent se dessiner.

L'ensemble de ces classes est contenu dans un projet librairie de classes situé dans la même solution que le projet console de test. **L'application console est créée puis construite en même temps que les différentes classes**. Les éléments spécifiques de chaque classe sont donc testés au fur et à mesure. Les fonctionnalités à tester sont décrites en fin de paragraphe.

2.1 Librairie de classe MyGraphicComponents

Classe **MyShape**	Créer une classe abstraite **MyShape** qui décrit toute forme géométrique. Elle doit contenir : • Une variable membre et sa propriété associée de type **MyPoint** qui représente le point d'accroche de la forme. • Un constructeur par défaut. • Une méthode abstraite Draw().
Classe **MyPoint**	Créer une classe **MyPoint** décrite par : • Deux coordonnées entières X et Y (variable membre et propriété). • Un constructeur par défaut. • Un constructeur d'initialisation. • La surcharge de la méthode ToString().
Classe **MySquare**	Créer une classe **MySquare** qui hérite de Shape et décrite par : • Une valeur entière représentant la longueur du coté (variable membre et propriété). • Un constructeur par défaut. • Un constructeur d'initialisation. • La surcharge de la méthode ToString(). • La redéfinition de la méthode Draw() qui affiche les informations concernant le carré dans la console.
Classe **MyCircle**	Créer une classe **MyCircle** qui hérite de Shape et décrite par : • Une valeur entière représentant le rayon (variable membre et propriété). • Un constructeur par défaut. • Un constructeur d'initialisation. • La surcharge de la méthode ToString(). • La redéfinition de la méthode Draw() qui affiche les informations concernant le cercle dans la console.

Classe **MyRectangle**	Créer une classe **MyRectangle** qui hérite de Shape et décrite par : • Une valeur entière représentant la longueur (variable membre et propriété). • Une valeur entière représentant la largeur (variable membre et propriété). • Un constructeur par défaut. • Un constructeur d'initialisation. • La surcharge de la méthode ToString(). • La redéfinition de la méthode Draw() qui affiche les informations concernant le rectangle dans la console.
Interface **IIsPointIn**	Créer une interface **IIsPointIn** implémentée par toutes les formes. Cette interface contiendra une méthode **IsPointIn(MyPoint p)** permettant de déterminer si un point passé en paramètre est ou non dans la forme. L'implémentation de cette méthode pour nos formes est triviale, un peu de math devrait vous permettre d'y arriver. L'objectif de cette interface sera de pouvoir sélectionner une forme en cliquant dans la zone de dessin dans la phase 2 du laboratoire.
Interface **IPointy**	Créer une interface **IPointy** implémentée par les formes délimitées par un ensemble de sommets : • Le carré en a 4, le rectangle en a 4, ils implémentent l'interface. • Le cercle n'est pas concerné car il n'a pas de sommets et n'implémente pas l'interface. Dans cette interface, ajouter une propriété en lecture seule : **Points**. Elle retourne le nombre de sommets de la forme. Implémenter ces méthodes lorsque c'est nécessaire.

2.2 Application console de test

Le projet en mode console permet de tester les différentes fonctionnalités des classes. Celles-ci seront complétées au fur et à mesure si nécessaire.

- Créer et afficher 2 objets de chaque sorte. Mettre en évidence l'utilisation des différents constructeurs et propriétés de chaque classe.
- Ajouter ces objets dans une liste générique d'objets de type **MyShape** (`List<MyShape>`).
- Afficher cette liste en utilisant le mot clé `foreach`.
- Afficher la liste des objets implémentant l'interface `IPointy`.
- Afficher la liste des objets n'implémentant pas l'interface `IPointy`.
- Créer une liste générique de 5 carrés, l'afficher, la trier par ordre de taille croissante, l'afficher à nouveau. Pour trier, la méthode `Sort()` de la classe `List<T>` utilise la

méthode CompareTo() définie grâce à l'implémentation dans la classe **MySquare** de l'interface Icomparable<**MySquare**>.

- Sans modifier la méthode CompareTo(), trier la précédente liste par ordre croissant des abscisses de position des carrés dans le plan. Pour ce faire, il s'agit de créer une classe **MySquareAbscisseComparer** implémentant l'interface IComparer<**MySquare**>.
- Rechercher, parmi les carrés de la liste, ceux qui présentent la même taille qu'un carré de référence. Pour rechercher, la méthode Find() de la classe List<T> utilise la méthode Equals() définie grâce à l'implémentation dans la classe **MySquare** de l'interface IEquatable<MySquare>.
- Rechercher, parmi les carrés de la liste, ceux qui contiennent un point passé en paramètre. Pour ce faire, il s'agit d'utiliser la méthode IsPointIn(…) contenue dans l'interface IIsPointIn.
- Par analogie, effectuer les 4 points précédents pour les autres formes,
- Mettre en place et tester un mécanisme qui permet de classer une liste d'objets **MyShape** sur base de leur surface.

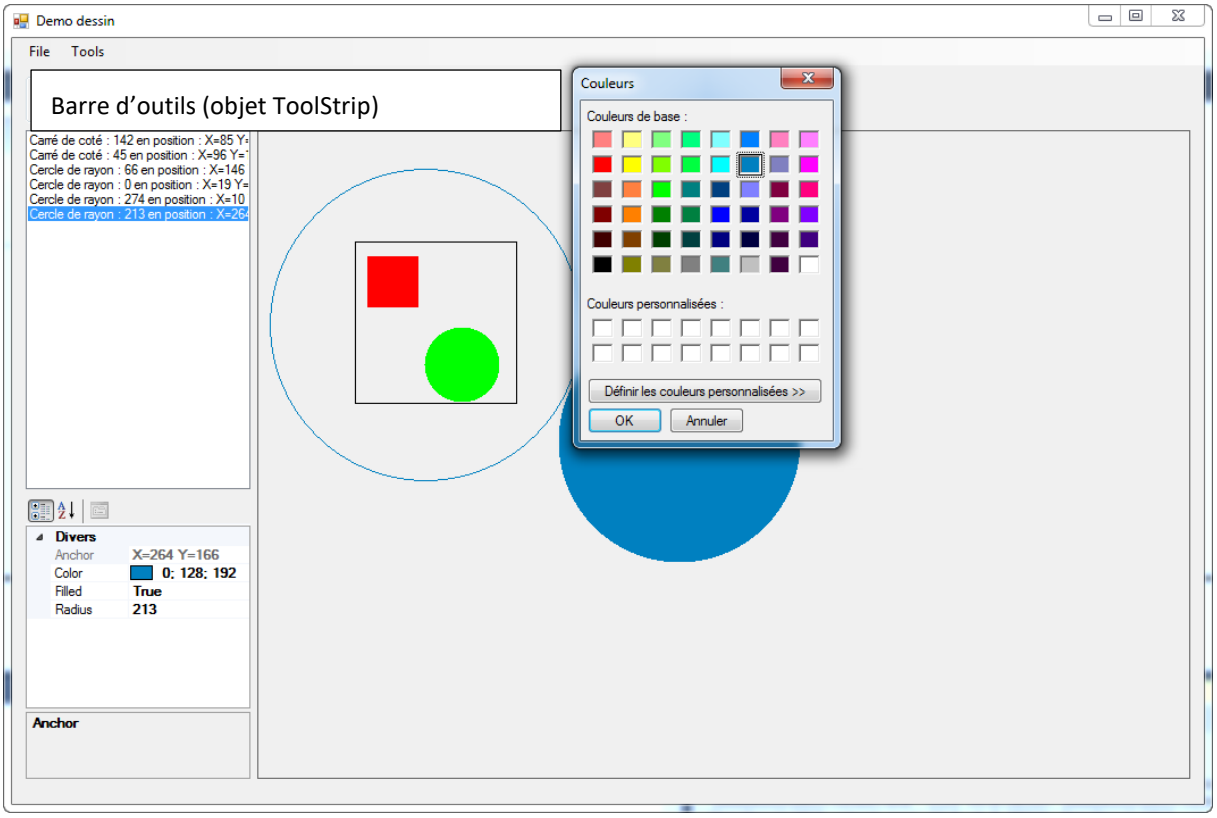
Remarque :

- Toutes les méthodes Draw() utilisent explicitement les méthodes ToString() définies dans chacune des classes.
- **Soyez attentifs à la qualité des messages affichés par la console pour la démonstration**. Par exemple, toutes les données d'un objet ou presque, peuvent être affichées sur une ligne en utilisant des tabulations pour aligner les mêmes types de paramètres.

3 Fonctionnalités Phase II (Semaine du 26/03/2018)

Il s'agit de développer une application permettant de dessiner les formes géométriques précédemment créées dans une fenêtre graphique.

Voici à titre informatif un exemple d'interface utilisateur élémentaire. Celui-ci ne doit pas être reproduit exactement mais vous permet de vous faire une idée du résultat attendu.



Hiérarchie de classes	Les éléments à dessiner sont les formes géométriques décrites dans le précédent dossier. Les éléments "dessinables" sont des formes géométriques simples. Les formes simples sont des carrés, des rectangles et des cercles. Suivant les cas, les éléments "dessinables" sont caractérisés par • Un nom (généré automatiquement à partir d'un compteur static du nombre d'instance de chaque sorte d'objet) puis pouvant être modifié). Modifier la méthode ToString() pour qu'elle retourne cette chaîne de caractères. • Les dimensions des formes (mesure du côté du carré, longueur et largeur du rectangle, rayon du cercle), • Son point d'ancrage dans la fenêtre (Classe ***MyPoint***), • La couleur (classe ***Color***) utilisée pour le dessiner, • Le statut "rempli" (Filled) ou non (booléen), • … Il s'agit donc de compléter et mettre à jour la hiérarchie de classes créée lors du premier dossier. Tous les aspects liés au fait que les objets graphiques se dessinent font partie de l'interface ***IDrawable*** (nouvelle interface que vous devez créer). Cette interface apporte l'implémentation d'un nouvelle méthode Draw(Graphics g) capable de dessiner les composants graphiques dans une **PictureBox**.

Application Winform	Créer une application Winform "Inpres-CAD" permettant de créer et dessiner des formes géométriques. Les éléments suivants doivent être présents dans l'interface utilisateur
	
	1. Une fenêtre principale contenant une ***PictureBox*** dans laquelle il est possible de dessiner des formes. Les objets graphiques seront dessinés exclusivement par l'intermédiaire de la méthode PaintPictureBox_Paint() appelée lors de l'apparition d'un événement Paint sur la PictureBox PaintPictureBox. L'appel à la méthode Invalidate() permet de forcer l'apparition de cet événement. Dans une PictureBox, le point de coordonnées (0,0) se trouve dans le coin supérieur gauche de celle-ci. L'axe des x est dirigé vers la droite, l'axe des y est dirigé vers le bas.
	2. Une ***Toolbar*** (composant ToolStrip) permettant de choisir :
	   a. La forme,
	   b. Ses caractéristiques de base (couleur, remplissage),
	   c. La commande : création, sélection, déplacement, suppression.

	Il s'agit bien ici d'utiliser le composant ToolStrip et pas de placer des composants de base (boutons checkbox, radiobutton) dans l'interface utilisateur. 3. Une zone permettant d'afficher les propriétés de la forme courante (***PropertyGrid***) 4. Une liste (***ListBox***) permettant d'afficher le nom (nom unique créé automatiquement) de la forme et de sélectionner les formes déjà créées. 5. Un menu contenant au moins les éléments suivants : a. File → Clear b. File → Save c. File → Load d. File → Exit e. Tools→ Option f. Tools →About box : Créer une « About box » comme il en existe dans de nombreux programmes (Auteur, copyright ☺, date). 6. Fonction sur les formes : a. Les formes sont créées à l'aide de clicks dans la ***PictureBox***, b. Elles doivent être sélectionnables via la liste ou à l'aide d'un click dans la fenêtre. Pour ce faire, on utilisera la méthode IsPointIn(…) développée dans la première partie du dossier. Une forme sélectionnée est affichée dans une couleur particulière ou avec un bord plus épais, c. Elles peuvent être supprimées ou modifiées (***PropertyGrid***), d. Elles peuvent être déplacées à l'aide de plusieurs clicks ou d'un Drag & Drop
Tools→Option	La fenêtre Tools→Option est une boîte de dialogue non modale (non bloquante) permettant de choisir et modifier la couleur utilisée pour écrire le texte affiché dans la ListBox contenant le nom des formes. Elle fonctionne de la même manière que la fenêtre qui permet de gérer la résolution de l'écran sous Windows, elle contient donc 3 boutons : • OK : valide le choix et ferme la fenêtre, • Cancel : ferme la fenêtre sans valider le dernier choix (qui n'a pas encore été appliqué), • Appliquer : valide le choix, ne ferme pas la fenêtre. La mise à jour de l'interface utilisateur principale se fera par la création d'un événement personnalisé envoyé par cette fenêtre à la fenêtre principale au moment opportun.

Informations complémentaires :

- L'ensemble des objets graphiques créés sont mémorisés dans une collection (**BindingList**<…>) gérées en mémoire. Il est conseillé de créer une classe permettant la gestion de cette collection de données ainsi que les différentes méthodes ou propriétés qui peuvent la manipuler. On y trouvera : une propriété de type get qui retourne la collection, des méthodes qui permettent de réinitialiser la collection, la charger ou l'enregistrer. **Vous veillerez à bien séparer le code propre à l'interface utilisateur du code propre à la gestion des données exclusivement effectuée dans cette classe**. De ce fait, la localisation de cette classe sera bien choisie.
- Pour ceux qui ne l'ont pas déjà fait lors de la première partie du labo, il est conseillé de créer une librairie contenant quelques méthodes statiques qui effectuent les calculs mathématiques de base dont vous pourriez avoir besoin. La librairie MyMathLib contient par exemple la classe MathUtil qui contient des méthodes « static » comme IsValueBetweenMinMax(…) qui vérifie si une valeur est comprise entre un minimum et un maximum ou entre deux valeurs (si on ne sait pas au départ quelle est la plus petite des deux) ou la méthode Distance(…) qui calcule la distance entre deux points dont on connait les coordonnées. Bien sûr cette librairies utilise principalement des paramètres de base (int, double) et ne connait pas la classe MyPoint.
- Dès que possible, **les techniques de databinding seront mises en place** (voir séance de théorie 5).
- Le menu File→Clear permet de supprimer l'ensemble des formes actuellement créées.
- La sauvegarde (File→Save) et le chargement (File→Load) doivent permettre de sauvegarder l'ensemble des formes actuellement créées dans un fichier XML (Save) ou de charger un fichier XML contenant des formes précédemment sauvegardées (Load).

Evaluation	L'évaluation portera sur : 1. La qualité de la solution (découpage en projets) 2. Respect des règles de nommage des composants graphiques, des variables membre, propriétés et méthodes 3. Respect de l'énoncé et des fonctionnalités demandées 4. La gestion de la liste des objets graphiques en mémoire et la mise à jour automatique des données lors de modifications 5. L'optimisation de l'utilisation des technologies

4 Fonctionnalités Phase III (juin 2018)

Lisez l'énoncé attentivement jusqu'à la fin

4.1 Mise en situation et objectifs

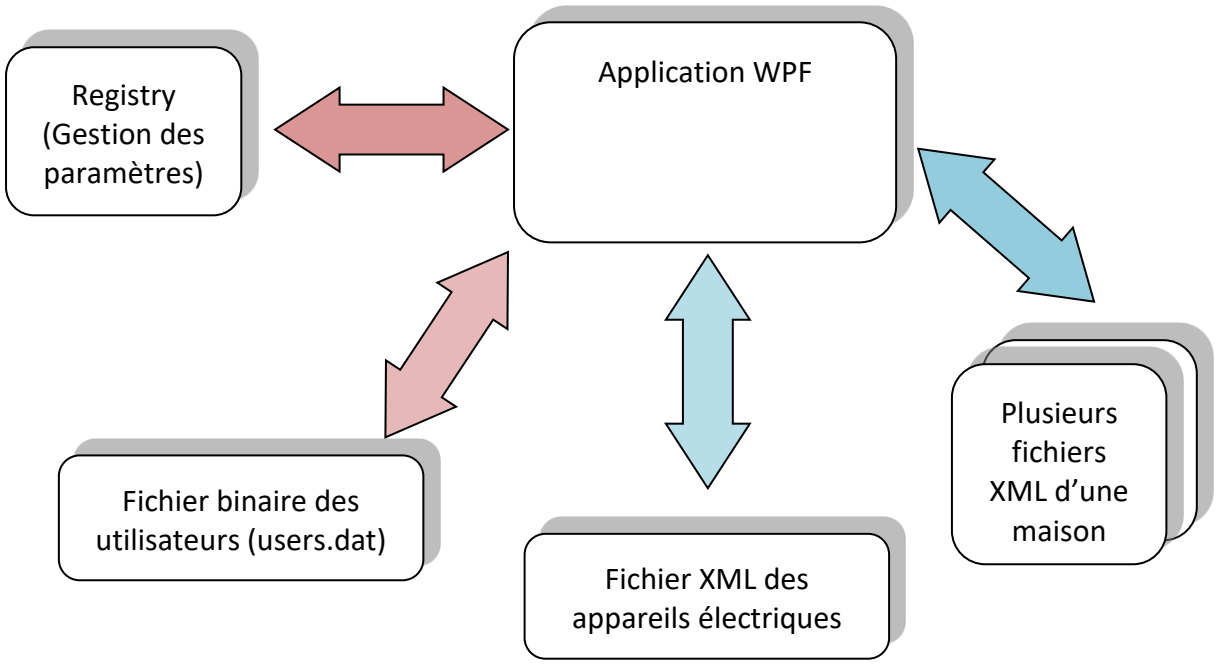
La société *InpresBetesDeCompetEnCsharp* s'est engagée à fournir un logiciel aux organismes régionaux responsables de l'octroi de primes environnementales aux entreprises et aux particuliers. Cette application permet de déterminer la consommation d'énergie moyenne d'une maison ou d'un immeuble quelconque. En fonction des résultats obtenus, l'organisme décide d'octroyer ou non la prime « HomeEnergySurvey ». Notons déjà que ce n'est pas l'application qui prend la décision. Elle la suggère à un utilisateur qui a le loisir ou la responsabilité de l'appliquer telle quelle ou non en fonction de paramètres supplémentaires.

L'objectif du logiciel est de simuler la dépense énergétique moyenne d'une habitation.

L'application est mise à la disposition de plusieurs profils d'utilisateurs :

- Les propriétaires d'habitations
- Les experts prenant la décision d'octroyer ou non la prime
- Les administrateurs de l'application, membre du personnel de la société

4.2 Architecture



4.3 Les utilisateurs

Un seul administrateur existe. Son mot de passe par défaut est « admin ». Il a la possibilité de le changer. Dans ce cas, il est stocké dans la registry (voir remarques concernant la registry au §0)

Les autres utilisateurs doivent s'authentifier pour pouvoir accéder aux différentes fonctionnalités de l'application. Les caractéristiques des comptes des utilisateurs sont

- Un nom (login unique)
- Un mot de passe
- Un profil choisi parmi deux types définis (utiliser une énumération Enum) :
    - Propriétaire
    - Expert

L'ensemble des comptes utilisateurs est stocké dans un fichier binaire « users.dat ».

4.4 Les maisons

Les **maisons** sont identifiées par leur adresse unique réduite à une chaîne de caractères (version minimum) ou représentée par une classe « Adresse » (BONUS).

Les maisons sont composées de **plusieurs niveaux** (étages, rez-de-chaussée, cave). Ils sont caractérisés par un nom et une collection de **pièces** (chambre, cuisine, living, bureau, garage).

Une pièce est caractérisée par un nom et une **collection d'appareils électriques**.

Une maison contient une information représentant son **état de certification** (Certifié ou non certifié) représenté par un booléen.

Les données d'une maison sont enregistrées dans un fichier XML portant un nom unique généré à partir de son adresse. Chacune appartient à un propriétaire et possède donc un lien vers celui-ci (voir utilisateur-propriétaire).

4.5 Les composants électriques

Nos habitations sont inondées d'appareils en tout genre qui consomment de l'électricité. Parmi ceux-ci, on trouvera :

- Les éclairages,
- Les appareils électroménagers
- Le chauffage,
- La TV, la chaîne hifi, l'ordinateur,
- Les appareils autonomes et mobiles qu'il faut charger régulièrement (GSM, appareil photo)

Voyant cette liste, il est assez aisé d'identifier un certain nombre de caractéristiques :

- Le type (éclairage, électroménager, loisir, ...) (nouvelle énumération)
- La marque
- Le modèle
- Le mode de fonctionnement (avec ou sans veille)
- Des données (avec ou sans veille) qui permettront à terme de calculer l'énergie consommée et donc d'évaluer les coûts de consommation (unité = puissance en Watts)
- Un nom d'image du produit
- La durée pendant laquelle il est utilisé pour une période déterminée (jour ou semaine) [valeur initialisée uniquement lorsque l'appareil est ajouté à une maison].

Tous les composants électriques disponibles sont stockés dans un fichier XML. Quelques composants électriques de base sont pré-encodés.

Il serait bien sûr très judicieux de gérer les appareils électriques d'une part et des références vers ceux-ci d'autre part lorsqu'ils sont intégrés dans une maison en y ajoutant les durées d'utilisation spécifique à la personne, la famille qui l'utilise dans sa maison. Tout cela pourrait se faire assez facilement avec une base de données, mais cela ne fait pas partie de la matière de cette année. L'idée simplifiée sera donc la suivante :

1. Gérer d'une part les appareils électriques de façon générale pour tous leurs paramètres sauf celui qui concerne la durée. Ceux-ci sont disponibles dans une GridView affichant l'images et les données pour l'utilisateur qui voudrait les ajouter dans sa maison.
2. Au moment où l'appareil est ajouté dans la maison, l'entièreté de ses données, y compris la durée seront ajoutées à la collection d'appareils d'une pièce.

4.6 Les calculs

L'application permet d'effectuer différents calculs de consommation pour une période donnée de :

- 1 jour (uniquement les appareils ayant une consommation journalière fixée)
- 1 semaine (tous les appareils y compris ceux ayant une consommation hebdomadaire comme le fer à repasser (5h par semaine) ou l'aspirateur (1h30 par semaine)).

Les résultats des calculs peuvent être affichés pour :

- Un appareil
- Tous les appareils d'une pièce
- *Tous les appareils de toutes les pièces d'une maison (BONUS)*

4.6.1 Les formules

Au-delà des calculs de l'énergie consommée, vous utiliserez une facture d'électricité récente pour calculer les coûts en euros. La simulation sera réalisée sur base d'un tarif unique.

Exemples de calcul :

1. Une ampoule de 50W allumée pendant 6h par jour. Energie consommée : 50W * 6h = 300Wh par jour.
2. Un fer à repasser 2200 W allumé pendant 5h par semaine. Energie consommée : 2200W * 5h = 11000 Wh = 11 KWh par semaine.
3. Un ordinateur portable utilise 4,74A à 19V pendant 8h par jour. Energie consommée : 4,74A * 19V * 8h = 720Wh par jour.

Nous espérons obtenir des résultats plausibles issus de vos calculs.

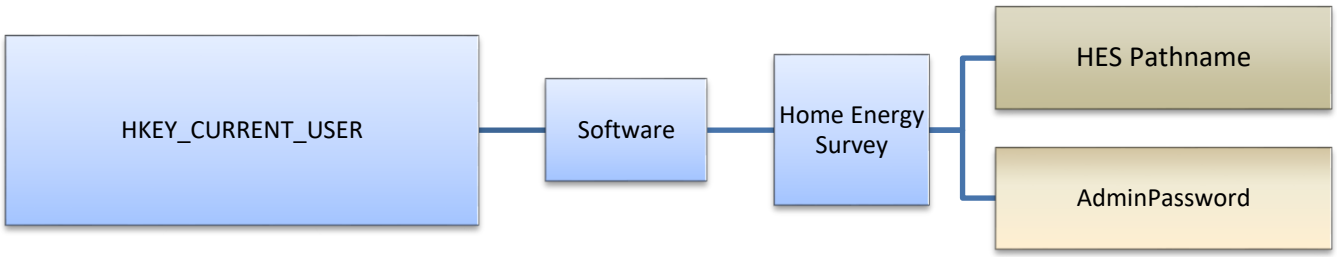
4.7 La certification d'une maison

Les experts ont l'opportunité de certifier une maison. Pour cela, après s'être connectés, ils peuvent cocher la case prévue à cet effet.

Lorsqu'une maison est certifiée, l'interface utilisateur affichant les détails de la maison change de couleur. Cette opération se fera grâce à l'utilisation du databinding et d'une classe implémentant l'interface IValuerConverter (voir dernière séance de théorie).

4.8 Données de la registry

La registry permet de mémoriser trois paramètres de l'application à l'aide de paires de clé/valeurs.



Dans la hiérarchie décrite ci-dessus, elle contient :

- « HES Pathname » : Le chemin d'accès aux fichiers de l'application. Ce chemin contient :
  - Un répertoire « Maisons » contenant les fichiers de description des maisons (un fichier par maison),
  - Un fichier « Appareils.xml » contenant les appareils,
  - Un répertoire « Images » contenant les images des appareils,
  - Un fichier « users.dat » contenant les utilisateurs.
- « AdminPassword » : Le mot de passe de l'administrateur.

Il s'agit donc, à l'aide d'une nouvelle classe que l'on pourrait appeler HomeEnergySurveyManager de proposer une gestion complète d'accès aux paramètres de l'application.
Cette classe offre des propriétés qui permettent d'accéder directement aux données de la Registry ou qui passent pas l'utilisation de variables membre et leurs propriétés associées initialisées à partir de méthodes telles que LoadRegistryParameter() ou SaveRegistryParameter(). Les deux méthodes ont leurs avantages et leurs inconvénients.
Cette classe s'occupe également de créer les sous-dossiers « Maisons », « Appareils » qui doivent être créés dans le dossier choisi pour la clé « HES Pathname ».

Une boîte de dialogue permet d'initialiser ces paramètres. Elle est accessible à partir du menu de l'application de l'administrateur.

Ressources : Voir « Etude C# et prog .NET - Part 6 V1.0.pdf » dans le centre de ressources de Cécile Moitroux ou notes de cours.

4.9 Fonctionnalités

Les fonctionnalités à développer seront accessibles aux utilisateurs en fonction de leur profil.

Deux boîtes de dialogues doivent être créées et exécutées en fonction du type de l'utilisateur connecté :

1. L'administrateur

2. Le propriétaire ou l'expert qui ont des interfaces très proches (seules quelques fonctionnalités diffèrent).

Fonctionnalité	Propriétaire	Expert	Administrateur
Fenêtre 1 : Gestion d'une ou plusieurs maisons			
1. Création d'une maison, définition du nombre de niveaux ainsi que des pièces faisant partie des niveaux.	X		
2. Visualisation de la structure de la maison dans une TreeView (§4.11). Un propriétaire y voit toutes ses maisons. Un expert y voit toutes les maisons qu'il doit traiter.	X	X	
2.1. La sélection de la maison provoque l'affichage de l'adresse de celle-ci ainsi que son niveau de certification (voir Fonctionnalité 4)	X	X	
2.2. La sélection d'un niveau affiche la liste des pièces qu'il contient			
2.3. La sélection d'une pièce permet :	X	X	
2.3.1. L'affichage des composants électriques insérés dans la pièce.	X	X	
2.3.2. L'ajout de nouveaux composants et donc l'encodage de ceux-ci ainsi que la configuration des données d'utilisation.	X		
3. Calcul des énergies consommées	X	X	
3.1. Les résultats des calculs des énergies consommées s'affichent automatiquement et en rapport avec l'élément sélectionné dans la TreeView.			
4. Certification d'une maison		X	
5. Gestion des utilisateurs (§4.3) :			X
5.1. Affichage de la liste dans une GridView			
5.2. Ajout / modification / suppression d'un utilisateur			
6. Ajout d'appareils électriques à la liste des appareils disponibles pour tous les utilisateurs (affichage dans une GridView montrant l'image de l'appareil)	X		X
7. Modification des paramètres de l'application (§0)			X

4.10**Méthodologie**

Identifiez les différentes parties sur lesquelles vous allez travailler.

- Identifiez les fenêtres et boîtes de dialogue.
- Identifiez les différents types d'objets à créer ainsi que le rôle qu'ils devront avoir.
- Identifiez les librairies (donc les nouveaux projets) à créer qui sont assez indépendantes des interfaces utilisateur et qui pourront être réutilisées dans d'autres applications manipulant les mêmes objets.

L'objectif est ici de découper votre projet en différentes parties isolées et performantes qui interagissent entre elles au service de l'application à développer.

Enfin, il va de soi que la solution proposée doit présenter une interface graphique WPF attrayante, intuitive et pratique pour l'utilisateur (ex : copier-coller pour dupliquer la configuration d'une habitation ou drag and drop pour DEPLACER un élément d'une pièce à une autre, etc …). Elle doit se mouler dans la mouvance des interfaces graphiques des applications Windows actuelles. La solution doit être codée de manière orientée objets. On y trouvera :

- Des classes,
- Des Interfaces,
- Des événements,
- Des collections,
- Du Databinding, des « Converter »
- …

4.11 **Ressources**

Pour la TreeView

https://www.codeproject.com/Articles/124644/Basic-Understanding-of-Tree-View-in-WPF

http://www.wpf-tutorial.com/treeview-control/treeview-data-binding-multiple-templates/

https://www.codeproject.com/Articles/55168/Drag-and-Drop-Feature-in-WPF-TreeView-Control

the best one … (MAJ le 17/05/2018)

https://blogs.msdn.microsoft.com/mikehillberg/2009/10/30/treeview-and-hierarchicaldatatemplate-step-by-step/