|  |
| --- |
|  |
| Petits pains |
| Application de rapport pour les CRA |
| *Spécification technique* |

|  |
| --- |
| Benoit Masson-Bedeau  24/10/2017 |

Table des matières

[1 Introduction 5](#_Toc496875758)

[2 Dossier Assets 5](#_Toc496875759)

[3 Dossier Command 5](#_Toc496875760)

[3.1 Fichier CommandHandler.cs 5](#_Toc496875761)

[3.2 Fichier CommandHandlerT.cs 5](#_Toc496875762)

[4 Dossier Converter 5](#_Toc496875763)

[4.1 Fichier BoolToVisibilityConverter.cs 6](#_Toc496875764)

[4.2 Fichier NullToBoolConverter.cs 6](#_Toc496875765)

[4.3 Fichier NullToVisibilityConverter.cs 6](#_Toc496875766)

[4.4 Fichier StringNullOrEmptyToBoolConverter.cs 6](#_Toc496875767)

[4.5 Fichier StringNullOrEmptyToVisibilityConverter.cs 6](#_Toc496875768)

[5 Dossier Data 7](#_Toc496875769)

[5.1 Fichier PetitsPainsStore.cs 7](#_Toc496875770)

[5.1.1 Propriétés 7](#_Toc496875771)

[5.1.2 Méthodes ReadConfig/WriteConfig 7](#_Toc496875772)

[5.1.3 Méthodes WriteCroissantsLines/ReadCroissantsLines 7](#_Toc496875773)

[5.1.4 Méthode ReadPersons 8](#_Toc496875774)

[5.1.5 Méthode GetDefaultPeople 8](#_Toc496875775)

[6 Dossier Resources 8](#_Toc496875776)

[6.1 Fichier EmailTemplate.tt 8](#_Toc496875777)

[6.1.1 Configuration 8](#_Toc496875778)

[6.2 Fichier EmailTemplate.cs 8](#_Toc496875779)

[6.3 Fichier EmailTemplateExtended.cs 8](#_Toc496875780)

[6.3.1 Champs 9](#_Toc496875781)

[6.3.2 Constructeurs 9](#_Toc496875782)

[7 Dossier Utils 9](#_Toc496875783)

[7.1 Fichier Enums.cs 9](#_Toc496875784)

[7.1.1 SortDirection 9](#_Toc496875785)

[7.2 Fichier Extensions.cs 9](#_Toc496875786)

[7.2.1 Méthode BubbleSort<T>(this ObservableCollection<T> collection) where T : IComparable<T> 9](#_Toc496875787)

[7.2.2 Méthode BubbleSort<T>(this ObservableCollection<T> collection, SortDirection? sortDirection) where T : IComparable<T> 10](#_Toc496875788)

[7.3 Fichier ItemsChangeObservableCollection.cs 10](#_Toc496875789)

[8 Dossier Model 10](#_Toc496875790)

[8.1 Fichier Person.cs 10](#_Toc496875791)

[8.1.1 Interfaces 10](#_Toc496875792)

[8.1.2 Propriétés 10](#_Toc496875793)

[8.1.3 Membre 11](#_Toc496875794)

[8.1.4 Constructeur par défaut 11](#_Toc496875795)

[8.1.5 Constructeur Person(string firstName, string lastName) 11](#_Toc496875796)

[8.1.6 Constructeur Person(string firstName, string lastName, string email) 11](#_Toc496875797)

[8.1.7 Constructeur Person(string firstName, string lastName, string persoId, string email) 11](#_Toc496875798)

[8.1.8 Constructeur Person(SerializationInfo info, StreamingContext context) 11](#_Toc496875799)

[8.1.9 Méthode ToString 11](#_Toc496875800)

[8.1.10 Méthode GetObjectData(SerializationInfo info, StreamingContext context) 11](#_Toc496875801)

[8.1.11 Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context) 12](#_Toc496875802)

[8.1.12 Méthode CompareTo(Person other) 12](#_Toc496875803)

[8.1.13 Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null) 12](#_Toc496875804)

[8.1.14 Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null) 12](#_Toc496875805)

[8.2 Fichier Croissant.cs 12](#_Toc496875806)

[8.2.1 Interfaces 12](#_Toc496875807)

[8.2.2 Enum CroissantState 13](#_Toc496875808)

[8.2.3 Propriétés 13](#_Toc496875809)

[8.2.4 Membre 13](#_Toc496875810)

[8.2.5 Constructeur par défaut 13](#_Toc496875811)

[8.2.6 Constructeur Croissant(DateTime? date) 13](#_Toc496875812)

[8.2.7 Constructeur Croissant(SerializationInfo info, StreamingContext context) 13](#_Toc496875813)

[8.2.8 Méthode ToString 14](#_Toc496875814)

[8.2.9 Méthode GetObjectData(SerializationInfo info, StreamingContext context) 14](#_Toc496875815)

[8.2.10 Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context) 14](#_Toc496875816)

[8.2.11 Méthode CompareTo(Croissant other) 14](#_Toc496875817)

[8.2.12 Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null) 14](#_Toc496875818)

[8.2.13 Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null) 15](#_Toc496875819)

[9 Dossier ViewModel 15](#_Toc496875820)

[10 Dossier View 15](#_Toc496875821)

Historique

|  |  |  |
| --- | --- | --- |
| Date | Version | Commentaire |
| 24/10/2017 | 1.0 | Création du document |

# Introduction

L’application *Petits Pains* consiste en une application permettant de saisir le rapport pour les CRA des personnes d’une équipe puis d’envoyer un email à ces derniers pour les informer de la synthèse.

Le tout, dans la joie et la bonne humeur : on n’est pas la Stasi 😊 !

# Dossier Assets

Dossier des ressources à utiliser dans le projet. Il contient :

* le fichier *credit.txt* est le fichier contenant un lien pointant vers l’auteur de l’image *croissant\_empty* ;
* les fichiers images sont utilisés pour représenter les statuts des emplacements.

# Dossier Command

Dossier dans lequel se trouve les classes de commande. Les classes de commande permettent de réaliser des *bind* sur des éléments de l’interface graphique, comme des boutons.

## Fichier CommandHandler.cs

*Command* qui implémente l’interface *ICommand*. Elle permet d’exécuter une fonction qui ne prend pas de paramètre en fonction d’une autre fonction qui détermine si la première fonction peut s’exécuter.

Le seul constructeur prend donc 2 paramètres :

* *methodToExecute* de type *Action* : méthode à exécuter ;
* *methodToDetermineCanExecute* de type *Func<bool>*: méthode permettant de déterminer si *methodToExecute* peut être exécutée.

## Fichier CommandHandlerT.cs

Même principe que **Fichier CommandHandler**. La seule différence est que *methodToExecute* prend un paramètre en entrée.

Le constructeur prend donc ces 2 paramètres en entrée :

* *methodToExecute* de type *Action<T>* : méthode à exécuter ;
* *methodToDetermineCanExecute* de type *Func<bool>*: méthode permettant de déterminer si *methodToExecute* peut être exécutée.

# Dossier Converter

Dossier contenant les *Converter*, permettant de convertir des objets d’un type particulier vers un type manipulable par l’interface graphique (et vice-versa).

## Fichier BoolToVisibilityConverter.cs

Converti un booléen vers un objet de type *Visibility* :

* booléen vaut *true* : *Visibility* en sortie vaut *Visible* ;
* sinon : *Visibility* en sortie vaut *Collapsed*.

Ceci permet de piloter la visibilité d’éléments de l’interface graphique grâce à un booléen.

## Fichier NullToBoolConverter.cs

Converti un objet vers un booléen.

Dans cette classe, un enum *NullToBoolDirection* est présent, et permet de déterminer le sens de conversion :

* *NullIsTrue* : un objet *null* doit être converti vers une valeur *true* ;
* *NullIsFalse* : un objet *null* doit être converti vers une valeur *false*.

La valeur de l’enum doit être passée dans le paramètre *Parameter*. La sortie est donc conforme à ce paramètre.

Si le paramètre est *null*, on considère que *NullIsFalse*.

## Fichier NullToVisibilityConverter.cs

Converti un objet vers un objet de type *Visibility*.

Dans cette classe, un enum *NullToVisibilityDirection* est présent, et permet de déterminer le sens de conversion :

* *NullIsVisible* : un objet *null* doit être converti vers une valeur *Visible* ;
* *NullIsCollapsed* : un objet *null* doit être converti vers une valeur *Collapsed*.

La valeur de l’enum doit être passée dans le paramètre *Parameter*. La sortie est donc conforme à ce paramètre.

Si le paramètre est *null*, on considère que *NullIsCollapsed*.

## Fichier StringNullOrEmptyToBoolConverter.cs

Converti une chaine de caractère *null* ou vide vers un booléen :

* la chaine est *null* ou vide : la valeur en sortie est *false* ;
* sinon : la valeur en sortie est *true*.

## Fichier StringNullOrEmptyToVisibilityConverter.cs

Converti une chaine de caractère *null* ou vide vers un objet de type *Visibility*:

* la chaine est *null* ou vide : la valeur en sortie vaut *Collapsed* ;
* sinon : la valeur en sortie vaut *Visible*.

# Dossier Data

Dossier contenant les classes permettant la manipulation des données.

## Fichier PetitsPainsStore.cs

Fichier permettant la manipulation des données de l’application.

Il s’agit d’une classe *static* dans laquelle se trouvent les méthodes permettant de :

* stocker les données ;
* lire les données.

### Propriétés

* *RootPath* :
  + disponible en lecture/écriture ;
  + si vide, retourne le chemin de l’application.
* *ConfigFilePath* :
  + disponible uniquement en lecture ;
  + retourne le chemin complet vers le fichier de config :
    - *[chemin de l’application]/petitspains.config*.
* *DefaultPeopleFileName* :
  + disponible uniquement en lecture ;
  + retourne le nom du fichier contenant les personnes par défaut :
    - *ListePersonnesDefaut.json*.
* *CroissantLinesFileName* :
  + disponible uniquement en lecture ;
  + retourne le nom du fichier contenant la sauvegarde du rapport :
    - *ListeLignes.json*.

### Méthodes ReadConfig/WriteConfig

Méthodes *public*.

Ces méthodes permettent de lire et d’écrire le fichier de configuration.

Le fichier de configuration est un fichier XML : il faut donc que ces méthodes écrivent/lisent les paramètres concernés au format XML.

### Méthodes WriteCroissantsLines/ReadCroissantsLines

Méthodes *public*.

Ces méthodes permettent de lire et d’écrire la sauvegarde pour le rapport des personnes suivies.

Le fichier de sauvegarde du rapport est un fichier JSON : il faut donc que les méthodes sérialisent/désérialisent celui-ci.

Dans le cas où le fichier n’existe pas lors de l’appel de *ReadCroissantsLines*, les personnes par défaut sont chargées via la méthode **Méthode ReadPersons**.

### Méthode ReadPersons

Méthode *private*.

Cette méthode permet de lire le fichier des personnes par défaut au format JSON ; il faut donc désérialiser celui-ci.

Si le fichier n’existe pas, une liste vide de personne est retournée.

### Méthode GetDefaultPeople

Méthode permettant de récupérer la liste des lignes contenant les personnes par défaut à charger, lorsqu’un fichier de sauvegarde n’est pas présent par exemple.

Concrètement, cette méthode construit une liste de lignes contenant les personnes récupérées par **Méthode ReadPersons**.

# Dossier Resources

Dossier contenant les fichiers annexes nécessaires au fonctionnement de l’application.

## Fichier EmailTemplate.tt

Fichier permettant la génération d’un texte au format HTML.

Le fichier utilise les champs passés en paramètre du constructeur de **Fichier EmailTemplateExtended.cs** pour générer un email conforme à ce qui est décrit dans la SF.

### Configuration

* le HTML généré doit être du HTML5 ;
* le HTML doit être au format UTF-8 ;
* l’accès aux images dans un email se fait par l’attribut *cid:*. Exemple :

<img src="cid:croissantEmpty" />

## Fichier EmailTemplate.cs

Fichier autogénéré lors de l’enregistrement de **Fichier EmailTemplate.tt**.

Ce fichier ne doit donc pas être modifié manuellement.

## Fichier EmailTemplateExtended.cs

Fichier d’extension pour la classe autogénérée **Fichier EmailTemplate.cs**. Permet d’étendre cette classe *partial*.

### Champs

* *date*, de type *DateTime* :
  + *private* ;
  + date à laquelle le rapport est traité.
* lines, de type *ItemsChangeObservableCollection<Line>* :
  + *private* ;
  + liste des lignes à afficher dans le rapport.

### Constructeurs

Un seul constructeur à ajouter : le constructeur prend 2 paramètres :

* *date*, de type *DateTime* :
  + *private* ;
  + date à laquelle le rapport est traité.
* lines, de type *ItemsChangeObservableCollection<Line>* :
  + *private* ;
  + liste des lignes à afficher dans le rapport.

Le constructeur s’occupe juste de valoriser ses champs avec ce qui lui est transmis en paramètre. Ainsi, les paramètres deviennent accessibles dans **Fichier EmailTemplate.tt**.

# Dossier Utils

Dossier contenant les classes utilitaires.

## Fichier Enums.cs

Fichier contenant les types *enum* nécessaires à l’application.

### SortDirection

Enum décrivant le sens de tri pour une collection :

* *Ascending* : le tri se fait de manière croissante ;
* *Descending* : le tri se fait de manière décroissante.

## Fichier Extensions.cs

Fichier contenant les extensions nécessaires à l’application.

### Méthode BubbleSort<T>(this ObservableCollection<T> collection) where T : IComparable<T>

Méthode d’extension pour un type *ObservableCollection<T>*.

Appelle **Méthode BubbleSort<T>(this ObservableCollection<T> collection, SortDirection? sortDirection) where T : IComparable<T>** avec le paramètre *SortDirection.Ascending*.

### Méthode BubbleSort<T>(this ObservableCollection<T> collection, SortDirection? sortDirection) where T : IComparable<T>

Méthode d’extension pour un type *ObservableCollection<T>*.

Tri une collection de type *ObservableCollection<T>* selon l’ordre demandé par *sortDirection*. Si *sortDirection* est null, le tri se fait de manière croissante.

## Fichier ItemsChangeObservableCollection.cs

Fichier permettant de notifier une collection lorsqu’un élément de celle-ci change. En effet, par défaut, lorsque ceci arrive, la collection n’est pas notifiée et l’interface graphique, par exemple, n’est pas mise à jour.

Son implémentation est reprise de ce projet :

* <https://www.codeproject.com/tips/694370/how-to-listen-to-property-chang>.

# Dossier Model

Dossier contenant les modèles.

## Fichier Person.cs

Fichier représentant une personne.

### Interfaces

Cette classe implémentent les interfaces suivantes :

* *ISerializable* ;
* *IComparable<Person>* ;
* *INotifyPropertyChanged*.

### Propriétés

* *FirstName*, de type *string* :
  + prénom d’une personne.
* *LastName*, de type *string* :
  + nom d’une personne.
* *PersoId*, de type *string* :
  + persoid d’une personne.
* *Email*, de type *string* :
  + email d’une personne.

Lors de la mise à jour de chaque propriété, un appel à **Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)** doit être réalisé.

### Évènements

* *event* *PropertyChangedEventHandler* *PropertyChanged* :
  + fait partie de l’implémentation de *INotifyPropertyChanged* ;
  + évènement à déclencher lorsque une personne change.

### Constructeur par défaut

Le constructeur par défaut est obligatoire pour assurer une sérialisation correcte. Ce constructeur est vide.

### Constructeur Person(string firstName, string lastName)

Valorisation des propriétés idoines.

### Constructeur Person(string firstName, string lastName, string email)

Valorisation des propriétés idoines.

### Constructeur Person(string firstName, string lastName, string persoId, string email)

Valorisation des propriétés idoines.

### Constructeur Person(SerializationInfo info, StreamingContext context)

Constructeur *protected*.

Désérialisation des propriétés ; cf. <https://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable.getobjectdata(v=vs.110).aspx> pour un exemple.

### Méthode ToString

Permet de générer une représentation d’une personne sous forme de string :

* *FirstName* + " " + *LastName*.

### Méthode GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de désérialiser un objet de type *Person*.

Cette méthode doit être *protected*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de désérialiser un objet de type *Person*.

Cette méthode est l’implémentation explicite de l’interface *ISerializable*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode CompareTo(Person other)

Implémentation de l’interface *IComparable<Person>*.

Permet de comparer une personne par rapport à une autre.

La comparaison doit se faire sur la propriété *FirstName*.

### Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)

Méthode à invoquer lorsqu’une personne change.

Cette méthode doit déclencher l’évènement *PropertyChanged* si des abonnés ont souscrit à celui-ci (c’est-à-dire que l’évènement n’est pas *null*).

### Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)

Méthode permettant de mettre à jour une propriété :

* si la valeur de la propriété n’a pas changée, rien ne doit être fait ;
* sinon, appel de **Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)**.

## Fichier Croissant.cs

Fichier représentant un croissant (un emplacement sur une ligne).

### Interfaces

Cette classe implémentent les interfaces suivantes :

* *ISerializable* ;
* *IComparable<Person>* ;
* *INotifyPropertyChanged*.

### Enum CroissantState

Enum décrivant l’état d’un croissant :

* *IsAvailable* : le croissant n’a pas reçu de pénalité et peu en recevoir une ;
* *IsUsed* : le croissant possède une pénalité ;
* *IsDeactivated* : le croissant est désactivé : il n’est pas possible de positionner une pénalité dessus.

### Propriétés

* *State*, de type *CroissantState* :
  + état du croissant.
* *Date*, de type *DateTime?* :
  + date à laquelle une pénalité a été positionnée sur le croissant ;
  + lors de sa valorisation, si celle-ci est différente de *null*, la propriété *State* doit être positionnée à *State.IsUsed*. Sinon (si la valeur est *null*), si le croissant n’est pas désactivé, State doit être positionné à *State.IsAvailable*.

Lors de la mise à jour de chaque propriété, un appel à **Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)** doit être réalisé.

### Évènements

* *event* *PropertyChangedEventHandler* *PropertyChanged* :
  + fait partie de l’implémentation de *INotifyPropertyChanged* ;
  + évènement à déclencher lorsque un croissant change.

### Constructeur par défaut

Le constructeur par défaut est obligatoire pour assurer une sérialisation correcte. Ce constructeur se contente de positionner la propriété *State* à *CroissantState.IsAvailable*.

### Constructeur Croissant(DateTime? date)

Constructeur permettant de valoriser la date du croissant.

### Constructeur Croissant(SerializationInfo info, StreamingContext context)

Constructeur *protected*.

Désérialisation des propriétés ; cf. <https://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable.getobjectdata(v=vs.110).aspx> pour un exemple.

### Méthode ToString

Permet de générer une représentation d’une personne sous forme de string :

* Si le croissant a une date de positionnée :
  + « Croissant : date :  » + [date au format dd/MM/yyyy] + *State*.
* sinon :
  + « Croissant : pas de pénalité ;  » + *State*.

### Méthode GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de désérialiser un objet de type *Croissant*.

Cette méthode doit être *protected*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de désérialiser un objet de type *Croissant*.

Cette méthode est l’implémentation explicite de l’interface *ISerializable*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode CompareTo(Croissant other)

Implémentation de l’interface *IComparable<Croissant>*.

Permet de comparer une croissant par rapport à un autre :

* si la date du croissant est *null* et que la date de l’autre croissant ne l’est pas, retourne 1 (un croissant ne possédant pas de date vient après un croissant avec) ;
* sinon si la date du croissant est non *null* et que la date de l’autre croissant l’est, retourne -1 (un croissant possédant une date vient avant un croissant sans) ;
* sinon si les deux dates sont *null*, retourne 0 (les croissants sont égaux) ;
* sinon, on compare les 2 dates.

### Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)

Méthode à invoquer lorsqu’un croissant change.

Cette méthode doit déclencher l’évènement *PropertyChanged* si des abonnés ont souscrit à celui-ci (c’est-à-dire que l’évènement n’est pas *null*).

### Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)

Méthode permettant de mettre à jour une propriété :

* si la valeur de la propriété n’a pas changée, rien ne doit être fait ;
* sinon, appel de **Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)**.

## Fichier Line

Fichier représentant une ligne. Une ligne possède une personne et les emplacements correspondants à cette personne.

### Interfaces

Cette classe implémentent les interfaces suivantes :

* *ISerializable* ;
* *INotifyPropertyChanged*.

### Propriétés

* *Person*, de type *Person* :
  + personne sur la ligne ;
  + lors du *set* de cette propriété, Méthode HandlePersonChanged(object sender, EventArgs e) doit souscrire à l’évènement *PropertyChanged* de *Person*.
* *Croissants*, de type *ItemsChangeObservableCollection<Croissant>* :
  + liste des croissants (emplacement) sur la ligne.
* *SelectedCroissant*, de type *Croissant* :
  + croissant sélectionné sur la ligne.
* *CroissantsSlots*, de type *int* :
  + en lecture seule ;
  + nombre de pénalités admissible pour la ligne.
* *PenaltiesAdded*, de type *ObservableCollection<DateTime*> :
  + liste des pénalités qu’une personne a eu pendant l’exécution courante de l’application.
* *HasToBringCroissant*, de type *bool* :
  + booléen indiquant si la personne doit ramener les croissants : cela signifie que la personne a dépassé le nombre de pénalités autorisé pendant l’exécution courante de l’application.
* *CroissantsSlotsDefault*, de type *int* :
  + propriété *static* ;
  + en lecture seule ;
  + retourne systématiquement 10 ;
  + nombre de pénalités par défaut pour une ligne.

Lors de la mise à jour de chaque propriété, un appel à **Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)** doit être réalisé.

### Évènements et delegates

* *event* *PropertyChangedEventHandler* *PropertyChanged* :
  + fait partie de l’implémentation de *INotifyPropertyChanged* ;
  + évènement à déclencher lorsque une ligne change.
* *delegate void PenaltyAlreadyExistsAtThisDateHandler(object sender, EventArgs e)* :
  + *handler* pour l’évènement *PenaltyAlreadyExistsAtThisDate*.
* *event PenaltyAlreadyExistsAtThisDateHandler PenaltyAlreadyExistsAtThisDate* :
  + évènement à déclencher lorsque le nombre de pénalités autorisées pour une personne est dépassée.

### Méthode GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de désérialiser un objet de type *Line*.

Cette méthode doit être *protected*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de désérialiser un objet de type *Line*.

Cette méthode est l’implémentation explicite de l’interface *ISerializable*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)

Méthode à invoquer lorsqu’un croissant change.

Cette méthode doit déclencher l’évènement *PropertyChanged* si des abonnés ont souscrit à celui-ci (c’est-à-dire que l’évènement n’est pas *null*).

### Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)

Méthode permettant de mettre à jour une propriété :

* si la valeur de la propriété n’a pas changée, rien ne doit être fait ;
* sinon, appel de **Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)**.

### Méthode HandlePersonChanged(object sender, EventArgs e)

Méthode permettant de gérer l’évènement *PropertyChanged* pour une personne.

Cette méthode est privée.

Cette méthode réalise une seule chose :

* si *PropertyChanged* n’est pas *null*, l’évènement est déclenché avec pour paramètre la ligne (*this*) et « Person » dans l’objet *PropertyChangedEventArgs*.

# Dossier ViewModel

# Dossier View