



Cégep de Saint-Hyacinthe
Département d'informatique

Programmation orientée objet

420-2DP-HY

(3-3-3)

Introduction aux collections et à LINQ #2

(Version 1.0)

3 heures

Préparé par

Martin Lalancette

Comprendre les éléments suivants:

- Opérateur de regroupement (*group by* et *into*)
- Opérateur de jointures (*join* et *on*)
- Expressions lambda

Table des matières

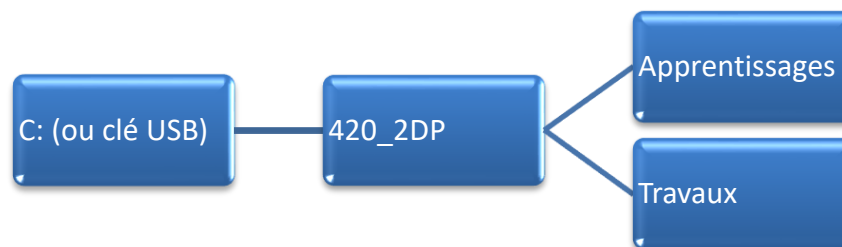
Introduction.....	3
Préparation de base	3
Introduction à LINQ – Suite	3
Opérateur de regroupement (<i>group by</i> et <i>into</i>).....	4
Opérateur de jointures (<i>join</i> et <i>on</i>)	6
Expressions lambda	9
Corps d’une expression lambda – forme abrégée	10
Une expression lambda avec la fonction Count<>	11
Une expression lambda avec les fonctions (Sum, Average, Min, Max).....	13
Une expression lambda avec les fonctions First<> et Last<>.....	14
Une expression lambda avec la fonction Where<>.....	15
Une expression lambda avec la fonction All<>.....	16
Une expression lambda avec la fonction Any<>	16
Autres exercices	16
Bibliographie.....	17

Introduction

Cette séquence a pour but de vous initier aux nécessaires à la programmation à base d'objets et d'événements. Nous commencerons par énoncer les éléments théoriques appuyés d'exemples simples et faciles à reproduire. Afin d'axer l'attention sur la compréhension de ces notions, il y aura des exercices à faire tout au long de cette séquence. **La séquence portera sur une introduction aux collections et à LINQ.**

Préparation de base

Pour bien suivre les instructions qui vont être mentionnées tout au long des séquences d'apprentissage, une préparation de base s'impose. Il est important de créer un répertoire de travail (sur votre C: ou clé USB). Voici une suggestion d'arborescence:



Exercice 1. : S'assurer d'avoir créé l'arborescence ici haut mentionnée sur votre C ou votre clé USB. Récupérer la solution portant le nom de ce document dans le fichier .ZIP. Ouvrir le projet « **Théorie** ».

Introduction à LINQ – Suite

Nous poursuivons l'apprentissage des notions reliées à LINQ en ajoutant le regroupement d'informations et la conception de requêtes interrogeant 2 ou plusieurs collections en même temps.

Opérateur de regroupement (*group by* et *into*)

Les mots clés [group...by](#) et [into](#) permettent d'obtenir une nouvelle collection de valeurs distincte de la catégorie indiquée. Voici la nomenclature :

```
from item in collection
group item by catégorie
into nouvelle_collection
```

Va servir de clé et sera représentée par la propriété Key.

Chaque élément de la nouvelle collection est un objet qui possède :

- Une **propriété Key** qui contient une valeur de la catégorie. Le compilateur déduit le type de la clé.
- Un **indexeur qui encapsule une sous-collection** d'items ayant cette valeur de catégorie en commun. Autrement dit, chaque élément de la nouvelle collection se comporte aussi comme une collection.

« Le mot clé contextuel **into** peut être utilisé pour créer un identificateur temporaire pour stocker les résultats d'une clause **group**, **join** ou **select** dans un nouvel identificateur. Cet identificateur peut lui-même être un générateur pour les commandes de requête supplémentaires. ».

Exemple #1 : Regrouper les étudiants par CodeCours dans une collection.

```
new Etudiant { DA="0981321", Prenom = "Serge", Nom = "Ouellet", CodeCours = "GEA" },
new Etudiant { DA="1234567", Prenom = "Claire", Nom = "Lamarche", CodeCours = "CAC" },
new Etudiant { DA="7654321", Prenom = "Steve", Nom = "Morel", CodeCours = "CAC" },
new Etudiant { DA="0123456", Prenom = "Carl", Nom = "Girard", CodeCours = "GEA" },
new Etudiant { DA="6543210", Prenom = "Dany", Nom = "Girard", CodeCours = "GEA" },
new Etudiant { DA="2345678", Prenom = "François", Nom = "Filion", CodeCours = "GEA" },
new Etudiant { DA="8765432", Prenom = "Henri", Nom = "Fortier", CodeCours = "CAC" },
new Etudiant { DA="3456789", Prenom = "Hugo", Nom = "Gagné", CodeCours = "CAC" },
new Etudiant { DA="9876543", Prenom = "Laurie", Nom = "Turcotte", CodeCours = "GEA" },
new Etudiant { DA="4567890", Prenom = "Thomas", Nom = "Audette", CodeCours = "GEA" },
new Etudiant { DA="0987654", Prenom = "Edouard", Nom = "Parenteau", CodeCours = "CAC" },
new Etudiant { DA="5678901", Prenom = "Michel", Nom = "Turcotte", CodeCours = "CAC" },
new Etudiant { DA="1098765", Prenom = "Zoe", Nom = "Forget", CodeCours = "GEA" }
```

...

```
var req =
from etudiant in _etudiants
// Grouper les étudiants selon le CodeCours et créer la nouvelle collection
group etudiant by etudiant.CodeCours into grpCodeCours
// Le "group by into" complète un groupe avant de poursuivre. Cependant le
// "from" associé à "group" n'est pas atteignable (etudiant n'est pas accessible
// au-delà de "group").
select new
{
    CodeCours = grpCodeCours.Key, // valeur d'un code de cours
    Etudiants = grpCodeCours      // sous-collection d'étudiants
};
```

Items à regrouper

Clé de regroupement

Nouvelle collection

```
// Pour afficher le résultat
foreach (var groupe in req)
{
    txtRésultat.AppendText("Groupe: " + groupe.CodeCours + "\n");
    foreach(var etudiant in groupe.Etudiants)
        txtRésultat.AppendText("Étudiant: " + etudiant + "\n");
}
```

Résultat :

Groupe: **GEA**
 Étudiant: DA = 0981321, Prenom = Serge, Nom = Ouellet
 Étudiant: DA = 0123456, Prenom = Carl, Nom = Girard
 Étudiant: DA = 6543210, Prenom = Dany, Nom = Girard
 Étudiant: DA = 2345678, Prenom = François, Nom = Fillion
 Étudiant: DA = 9876543, Prenom = Laurie, Nom = Turcotte
 Étudiant: DA = 4567890, Prenom = Thomas, Nom = Audette
 Étudiant: DA = 1098765, Prenom = Zoe, Nom = Forget
 Groupe: **CAC**
 Étudiant: DA = 1234567, Prenom = Claire, Nom = Lamarche
 Étudiant: DA = 7654321, Prenom = Steve, Nom = Morel
 Étudiant: DA = 8765432, Prenom = Henri, Nom = Fortier
 Étudiant: DA = 3456789, Prenom = Hugo, Nom = Gagné
 Étudiant: DA = 0987654, Prenom = Edouard, Nom = Parenteau
 Étudiant: DA = 5678901, Prenom = Michel, Nom = Turcotte

Exercice 2. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection **_employees**, va **grouper tous les employés par division** en **ordre décroissant**. Afficher les employés sous chaque division.

Exercice 3. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection **_employees**, va catégoriser les employés par tranche de 10 ans d'ancienneté en **ordre croissant** de tranche. Afficher les employés sous chaque tranche. Exemple :

Ancienneté: 0 an(s) et plus

Employé: E00002 (RVS) Savard Roger (1995-08-10) - Ancienneté: 3
 Employé: E00003 (RVS) Tremblay Lucie (1991-04-09) - Ancienneté: 4
 Employé: E00008 (RVS) Roy Marcel (1997-01-09) - Ancienneté: 8
 Employé: E00010 (RVN) Lavigne Isidore (1997-05-18) - Ancienneté: 4

Ancienneté: 10 an(s) et plus

Employé: E00001 (MTL) Roy Luc (1980-01-23) - Ancienneté: 10
 Employé: E00005 (MTL) Savard Marc (1978-11-07) - Ancienneté: 15
 Employé: E00006 (MTL) Côté Luc (1993-07-21) - Ancienneté: 11
 Employé: E00007 (RVN) Mercure Renée (1981-02-15) - Ancienneté: 10

Ancienneté: 20 an(s) et plus

Employé: E00004 (RVN) Simard Élise (1975-06-13) - Ancienneté: 20

Ancienneté: 30 an(s) et plus

Employé: E00009 (MTL) Turcotte Roger (1954-10-07) - Ancienneté: 31

Exercice 4. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection `_patients`, va **regrouper les dates de visites de chaque patient par mois**. Exemple :

```
Patient: NAM = VEZJ87120299, Prenom = Jacques, Nom = Vézina
    Mois: 03
        Date visite: 2005-03-20
        Date visite: 2006-03-03
Patient: NAM = ROYL65561199, Prenom = Roy, Nom = Lisa
    Mois: 05
        Date visite: 1997-05-13
        Date visite: 2001-05-04
    Mois: 11
        Date visite: 1999-11-03
    Mois: 09
        Date visite: 2000-09-22
Patient: NAM = COTR50081499, Prenom = Côté, Nom = Roger
    Mois: 02
        Date visite: 1980-02-26
    Mois: 07
        Date visite: 1985-07-01
    Mois: 09
        Date visite: 1988-09-12
Patient: NAM = SAVR77022899, Prenom = Savard, Nom = Réjean
    Mois: 02
        Date visite: 1978-02-13
    Mois: 09
        Date visite: 1979-09-01
        Date visite: 2009-09-04
    Mois: 07
        Date visite: 2010-07-21
```

Opérateur de jointures (*join* et *on*)

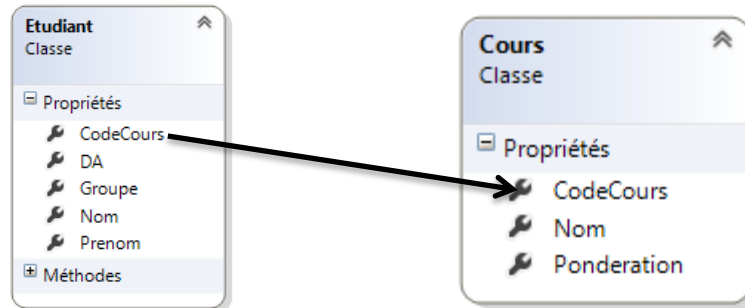
Les mots clés [join...on](#) permettent d'assembler les données complémentaires de 2 collections en se servant d'une donnée commune. Voici la nomenclature :

```
from itemA in collectionA
join itemB in collectionB
on itemA.donnée equals itemB.donnée
```

À savoir :

- Suite à la jointure, les données de l'itemA et l'itemB sont coordonnés comme s'il s'agissait d'un seul item.
- Il faut utiliser itemA pour accéder aux autres données propres à l'itemA. Même chose pour les autres données propres à l'itemB.

Exemple #1 : Joindre les étudiants et les cours selon les codes de cours communs



```

var req =
    from etudiant in _etudiants
    join cours in _cours
    on etudiant.CodeCours equals cours.CodeCours
    select new
    {
        Nom = etudiant.Nom,
        Prenom = etudiant.Prenom,
        Cours = cours.Nom,
        Ponderation = cours.Ponderation
    };

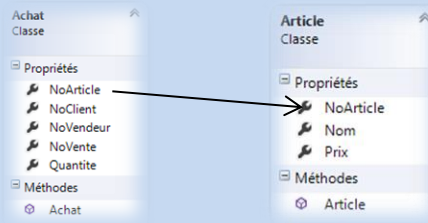
foreach (var item in req)
    txtRésultat.AppendText("Etudiant: " + item.Nom + " " + item.Prenom +
        ", Cours: " + item.Cours + ", Pondération: " +
        item.Ponderation + "\n");
  
```

Résultat :

```

Etudiant: Ouellet Serge, Cours: Programmation 2, Pondération: 3-3-3
Etudiant: Lamarche Claire, Cours: Internet, Pondération: 2-2-2
Etudiant: Morel Steve, Cours: Internet, Pondération: 2-2-2
Etudiant: Girard Carl, Cours: Programmation 2, Pondération: 3-3-3
Etudiant: Girard Dany, Cours: Programmation 2, Pondération: 3-3-3
Etudiant: Fillion François, Cours: Programmation 2, Pondération: 3-3-3
Etudiant: Fortier Henri, Cours: Internet, Pondération: 2-2-2
Etudiant: Gagné Hugo, Cours: Internet, Pondération: 2-2-2
Etudiant: Turcotte Laurie, Cours: Programmation 2, Pondération: 3-3-3
Etudiant: Audette Thomas, Cours: Programmation 2, Pondération: 3-3-3
Etudiant: Parenteau Edouard, Cours: Internet, Pondération: 2-2-2
Etudiant: Turcotte Michel, Cours: Internet, Pondération: 2-2-2
Etudiant: Forget Zoe, Cours: Programmation 2, Pondération: 3-3-3
  
```

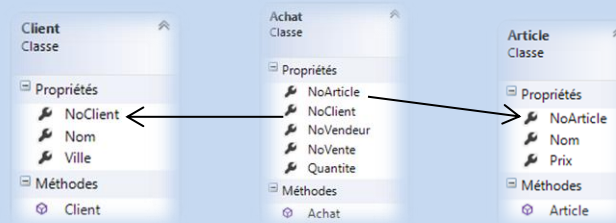
Exercice 5. Dans l'événement Click du bouton associé, ajouter le code qui, à partir des `_articles` et `_achats`, va obtenir le **numéro de vente**, le **nom** de l'article, son **prix**, la **quantité** achetée et le **total de la transaction** pour chaque transaction d'achat.



Résultat :

```
Vente: 3001, Article: PC Portable, Prix: 1200, Qté: 1, Total: 1200
Vente: 3002, Article: PC Bureau , Prix: 1000, Qté: 1, Total: 1000
Vente: 3003, Article: USB 4G, Prix: 25, Qté: 2, Total: 50
Vente: 3004, Article: HDD 1T, Prix: 120, Qté: 1, Total: 120
Vente: 3005, Article: HDD 500G , Prix: 75, Qté: 1, Total: 75
Vente: 3006, Article: Souris, Prix: 30, Qté: 1, Total: 30
Vente: 3007, Article: USB 16G, Prix: 150, Qté: 1, Total: 150
Vente: 3008, Article: USB 4G, Prix: 25, Qté: 20, Total: 500
Vente: 3009, Article: Souris, Prix: 30, Qté: 1, Total: 30
Vente: 3010, Article: HDD 500G , Prix: 75, Qté: 20, Total: 1500
```

Exercice 6. Dans l'événement Click du bouton associé, ajouter le code qui, à partir des `_articles`, `_achats` et `_clients`, va obtenir le **nom** de chaque client et le **nom des articles** achetés et la **quantité**.



Résultat :

```
Client: Jacques, Article: PC Portable, Qté: 1
Client: Jacques, Article: USB 4G, Qté: 2
Client: Jacques, Article: Souris, Qté: 1
Client: André, Article: PC Bureau , Qté: 1
Client: André, Article: HDD 500G , Qté: 1
Client: André, Article: Souris, Qté: 1
Client: Josée, Article: HDD 1T, Qté: 1
Client: Josée, Article: USB 16G, Qté: 1
Client: Aline, Article: USB 4G, Qté: 20
Client: Aline, Article: HDD 500G , Qté: 20
```


Exercice 7. Dans l'événement Click du bouton associé, ajouter le code qui va ajouter une jointure à la collection **_employes**. Faire afficher les mêmes informations (en ajoutant le prénom du vendeur) mais en fonction de l'employé **numéro « E00007 »**. Afficher le résultat dans txtRésultat. Resultat :


Client: Jacques, Article: Souris, Qté: 1, Vendeur: Renée
Client: Josée, Article: HDD 1T, Qté: 1, Vendeur: Renée
Client: Josée, Article: USB 16G, Qté: 1, Vendeur: Renée

Expressions lambda

La prochaine section sera une initiation aux expressions lambda fortes utiles et servant de complément au LINQ. Les [expressions Lambda](#) sont des **fonctions anonymes (donc ayant des paramètres, du code et un retour) sans identificateur**. « À l'aide d'expressions lambda, vous pouvez écrire des fonctions locales qui peuvent être passées comme arguments ou être retournées sous forme de valeur des appels de fonction. Les expressions lambda sont particulièrement utiles pour écrire des expressions de requête LINQ. » (MSDN, 2013). Voici un lien concernant le [brevet](#).

Pour exécuter, à volonté, les lignes de codes d'une méthode ou d'une fonction, on indique son identificateur et ses paramètres :

Ex. **MessageBox.Show("Bonjour!");**


méthode(paramètre)

Ex. **int valeurPositive = Math.Abs(valeur);**


retour ← fonction(paramètre)

Comme dans le cas des classes, les méthodes possèdent un identificateur afin de les réutiliser au besoin. Quand le besoin est ponctuel et unique, il est possible de recourir à :

- Des classes anonymes;
- Des méthodes anonymes;
- Des fonctions anonymes (expression Lambda);

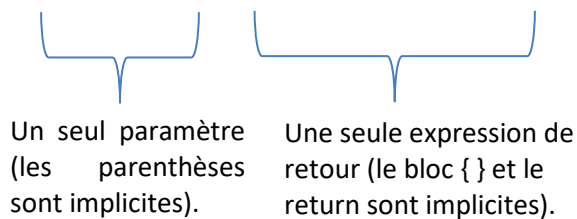
Avant de voir les notions sur les expressions lambda, voici un survol des connaissances associées aux délégués.

Corps d'une expression lambda – forme abrégée

Il existe deux formes associées à une [expression lambda](#): longue ou abrégée. Pour l'instant, voyons la forme abrégée.

Forme abrégée d'une expression lambda

Fonction(paramètre => expression_retour)



Exemple de syntaxe :

```
x => x > 5
```

Ici pour la valeur x contenue dans une liste, on applique une expression pour évaluer si elle est supérieure à 5. La fonction appelée retournera les éléments répondant à cette expression. Exemples :

```
(iNb1, iNb2) => (iNb1 + iNb2); // Expression lambda abrégée (1 seule ligne de code)  
(iNb1, iNb2) => { return iNb1 * iNb2; }; // Expression lambda longue (plusieurs lignes)
```

Pour avoir un aperçu plus concret, travaillons avec des fonctions déjà connues.

Une expression lambda avec la fonction Count<>

Nous savons que la fonction *Count* a comme algorithme de retourner le nombre d'éléments contenu dans une liste. Voici une déclaration et une instantiation d'une liste qui est initialisée avec des valeurs entières :

```
List<int> _nombres = new List<int>() { 40, 5, 90, 123, 2, 33, 210, 101, 78, 21 };
```

Exemple #1 : Combien y a-t-il de nombres?

Pour connaître la réponse à cette question, il suffit d'appeler la fonction *Count* de la façon suivante :

```
int iQuantite = _nombres.Count(); // Réponse: 10
```

Jusqu'ici, pas trop de complexité. Passons au prochain exemple.

Exemple #2 : Combien y a-t-il de valeurs entières supérieures à 100?

Après avoir vu les notions de base sur LINQ, nous pouvons écrire le code suivant :

```
var req = from nombre in _nombres
          where nombre > 100
          select nombre;

int iQuantite = req.Count(); // Réponse: 3
```

Ça fait bien le travail. Cependant, les développeurs de Microsoft voulaient simplifier le tout. Ils ont donc inventé cette syntaxe (lambda) plus simple pour en arriver au même résultat. Maintenant, voici comment simplifier :

```
int iQuantite = _nombres.Count(nombre => nombre > 100); // Réponse: 3
```

Représente un
élément dans la
liste de type **int**

Condition qui agit comme
filtre sur la liste

Exercice 8. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection `_nombres`, va compter les nombres se trouvant entre 20 et 80. Afficher le résultat dans `txtResultat`.

Exercice 9. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection `_nombres`, va compter les nombres dont le carré est divisible par 3. Afficher le résultat dans `txtResultat`.

Exercice 10. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection `_prenoms`, va compter les prénoms ayant un « a » en deuxième position. Afficher le résultat dans `txtResultat`.

Exercice 11. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection `_mots`, va compter les mots de plus de 3 lettres. Afficher le résultat dans `txtResultat`.

Exercice 12. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection `_mots`, va compter les mots dont la lettre « s » se retrouve plus d'une fois dans ce mot. Afficher le résultat dans `txtResultat`.

Jusqu'à maintenant, nous avons travaillé avec les listes contenant des types de base (comme `int`, `double` et `string`). Voici des exemples comportant des objets plus complexes.

Voici une liste d'étudiants :

```
List<Etudiant> _etudiants = new List<Etudiant>()
{
    #region Données
    new Etudiant { DA="0981321", Prenom = "Serge", Nom = "Ouellet", CodeCours = "GEA" },
    new Etudiant { DA="1234567", Prenom = "Claire", Nom = "Lamarche", CodeCours = "CAC" },
    new Etudiant { DA="7654321", Prenom = "Steve", Nom = "Morel", CodeCours = "CAC" },
    new Etudiant { DA="0123456", Prenom = "Carl", Nom = "Girard", CodeCours = "GEA" },
    new Etudiant { DA="6543210", Prenom = "Dany", Nom = "Girard", CodeCours = "GEA" },
    new Etudiant { DA="2345678", Prenom = "François", Nom = "Filion", CodeCours = "GEA" },
    new Etudiant { DA="8765432", Prenom = "Henri", Nom = "Fortier", CodeCours = "CAC" },
    new Etudiant { DA="3456789", Prenom = "Hugo", Nom = "Gagné", CodeCours = "CAC" },
    new Etudiant { DA="9876543", Prenom = "Laurie", Nom = "Turcotte", CodeCours = "GEA" },
    new Etudiant { DA="4567890", Prenom = "Thomas", Nom = "Audette", CodeCours = "GEA" },
    new Etudiant { DA="0987654", Prenom = "Edouard", Nom = "Parenteau", CodeCours = "CAC" },
    new Etudiant { DA="5678901", Prenom = "Michel", Nom = "Turcotte", CodeCours = "CAC" },
    new Etudiant { DA="1098765", Prenom = "Zoe", Nom = "Forget", CodeCours = "GEA" }
    #endregion
};
```

Exemple #3 : Combien y a-t-il d'étudiants participant au cours GEA?

En utilisant les notions de base LINQ, nous pouvons obtenir une requête de la façon suivante :

```
var req = from etudiant in _etudiants
          where etudiant.CodeCours == "GEA"
          select etudiant;

int iQuantite = req.Count(); // Réponse:7
```

Maintenant la même chose, mais avec une expression lambda :

```
int iQuantite = _etudiants.Count(etudiant => etudiant.CodeCours == "GEA");
```

Exercice 13. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection **_etudiants**, va compter les étudiants faisant partie du **cours « CAC »** dont le nom de famille est « **Turcotte** ». Afficher le résultat dans txtRésultat.

Voyons maintenant d'autres méthodes utiles qui profitent des expressions lambda.

Une expression lambda avec les fonctions (Sum, Average, Min, Max)

Nous venons de voir la fonction *Count*. Les quatre autres fonctions statistiques fonctionnent de la même manière, seulement le résultat diffère.

Sum = **Total** des valeurs basé sur l'expression lambda

Average = **Moyenne** des valeurs basée sur l'expression lambda

Min = **Plus petite** valeur basée sur l'expression lambda

Max = **Plus grande** valeur basée sur l'expression lambda

Exercice 14. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection **_nombres**, va trouver le **total**, la **moyenne**, la valeur **la plus petite** et la **plus grande** des valeurs. Afficher le résultat dans txtRésultat.

Exercice 15. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection **_mots**, va trouver le **nombre total de lettres**, la **moyenne**, le nombre de lettres **la plus petite** et **la plus grande**. Afficher le résultat dans la txtRésultat.

Exercice 16. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection **_employés**, va trouver le **total**, la **moyenne**, la valeur **la plus petite** et la **plus grande** à partir de **l'ancienneté**. Afficher le résultat dans txtRésultat.

Une expression lambda avec les fonctions **First<>** et **Last<>**

Les fonctions retournent le premier élément ou le dernier qui respecte l'expression lambda. **L'ordre de la collection est considéré.**

Exercice 17. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection **_nombres**, va trouver la **première** valeur et la **dernière** valeur **plus grande que 100**. Afficher le résultat dans txtRésultat.

Exercice 18. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection `_mots`, va trouver le **premier** et le **dernier** mot commençant par « p » et finissant par « t ». Afficher le résultat dans txtRésultat.

Exercice 19. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection `_articles`, va trouver le **premier** et le **dernier** prix de l'article commençant par « USB ». Afficher le résultat dans txtRésultat.

Une expression lambda avec la fonction Where<>

La fonction Where<> retourne la collection d'éléments qui respecte l'expression lambda. Valeurs :

```
List<int> _nombres = new List<int>() { 40, 5, 90, 123, 2, 33, 210, 101, 78, 21 };
```

Exemple #1 : Récupérer les entiers inférieurs à 50.

Voici trois façons de les obtenir avec la même clause Where:

```
var resultat1 = _nombres.Where(e => e < 50);  
int[] resultat2 = _nombres.Where(e => e < 50).ToArray();  
List<int> resultat3 = _nombres.Where(e => e < 50).ToList();
```

Exercice 20. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection `_mots`, va extraire les mots dont la longueur est **plus petite que 8** et qui **ne commencent pas par p**. Afficher le résultat dans la txtRésultat.

Exercice 21. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection `_employés`, va extraire les employés de la Rive-Sud ou bien qui ne fait pas partie de la succursale 0002. Afficher le résultat dans txtRésultat.

Une expression lambda avec la fonction All<>

La fonction All<> retourne true si **tous les éléments respectent l'expression lambda**, sinon retourne false. Valeurs :

```
List<int> _nombres = new List<int>() { 40, 5, 90, 123, 2, 33, 210, 101, 78, 21 };
```

Exemple #1 : Est-ce que tous les entiers sont > 0?

```
bool bResultat = _nombres.All(nombre => nombre > 0); // Réponse: true
```

Exercice 22. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection **_clients**, va indiquer si **tous les clients** demeurent à **Saint-Hyacinthe**. Afficher le résultat dans txtRésultat.

Une expression lambda avec la fonction Any<>

La fonction Any<> retourne true **si au moins 1 élément respecte l'expression lambda**, sinon retourne false. Valeurs :

```
List<int> _nombres = new List<int>() { 40, 5, 90, 123, 2, 33, 210, 101, 78, 21 };
```

Exemple #1 : Existe-t-il au moins un nombre plus grand que 200?

```
bool bResultat = _nombres.Any(nombre => nombre > 200); // Réponse: true
```

Exercice 23. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection **_evaluations**, va indiquer s'il existe un **TP3** dont la **note est supérieure à 90**. Afficher le résultat dans txtRésultat.

Autres exercices

Exercice 24. Dans l'événement Click du bouton associé, ajouter le code qui, à partir de la collection **_employes**, va extraire les **deux premiers** employés faisant partie de la **division MTL** en **ordre d'ancienneté** (utiliser OrderBy) dans un tableau. Afficher le résultat dans txtRésultat.

Bibliographie

Mastriani, R. (2013, 04 05). PowerPoint - LINQ. Saint-Hyacinthe, QC, Canada.

MSDN. (2013, 01 01). *Collections (C# et Visual Basic)*. Consulté le 04 04, 2013, sur MSDN - Visual Studio: <http://msdn.microsoft.com/fr-fr/library/vstudio/ybcx56wz.aspx>

MSDN. (2013, 01 01). *Expressions lambda (Guide de programmation C#)*. Consulté le 04 13, 2013, sur MSDN: <http://msdn.microsoft.com/fr-fr/library/vstudio/bb397687.aspx>

Roberto, M. (2013, 04 13). PowerPoint - Exp Lambda. Saint-Hyacinthe, QC, Canada.