Conventions de codage CSharp

Sujet: Conventions de programmation Csharp

Auteur : Enseignants CFPT-i / Microsoft

Date : 22 août 2013

Ecole d'Informatique
10, Gérard-de-Ternier
1213 Petit-Lancy
Tél 00 41 22 388 87 28

1 But des règles de codage

Le coût de la conception de logiciels est très élevé. Pour préserver ses investissements, l'entreprise sérieuse édite des règles de codage motivées par les raisons suivantes :
–un logiciel doit en tout temps rester modifiable, adaptable et compréhensible;
–une fonction réalisée doit pouvoir être récupérée sans difficultés pour un nouveau projet;

–l'indisponibilité d'un programmeur ne doit jamais mettre le projet en péril;
–des projets réalisés en équipe doivent être cohérents.

Exemple du projet *open source WebKit* :
http://www.webkit.org/coding/coding-style.html

2 Documentation

Les parties importantes sont expliquées par un organigramme, un diagramme d'état, un structogramme ou du pseudo-code qui peuvent être manuscrits ou réalisés au moyen d'un logiciel spécialisé.

2.1 En-tête du programme

Chaque programme commence par un en-tête qui comprend au minimum les inforsmations suivantes :
–le nom du projet;
–le nom de l'auteur du programme;
–la description générale du projet;
–la date et la référence de la version de base;
–la date et la référence de chaque version avec la description des modifications.

Exemple :

```
1    /*
2    * Projet :   NPI Calculatrice
3    * Auteur :   GAF
4    * Desc. :    Calculatrice (4 opérations), NotationPolonaiseInverse
5    * Version:   1.0, 2012.09.14, GAF, version initiale
6    */
```

2.2 Commentaires du programme

Les commentaires sont censés aider la personne qui lira votre programme. Ils ne sont pas là pour indiquer ce que le code dit déjà de façon manifeste.

« Ne soulignez pas ce qui est évident. »

Il n'est pas utile de commenter chaque ligne, mais il faut fournir les renseignements clés.

Les éléments suivants doivent obligatoirement être commentés :
–les définitions de classes;
–les déclarations de constantes, d'objets et de variables;
–les déclarations de méthodes;
–les lignes de code complexes.

```
1 compteurEspace = compteurEspace + 1; // increment compteur Espace
2 total = nombreRecu; // initialise total avec nombre Recu
```

Tous les commentaires du listing 2 sont inutiles et devront être supprimés. Ils n'apportent rien à la compréhension du code. Le lecteur lit couramment le langage C#.

2.3 Documentation des méthodes des classes créées

Chaque méthode possède une documentation. Les renseignements permettent à un programmeur étranger au projet d'utiliser la méthode sans devoir analyser son code.
L'en-tête comprend au moins :
–la description du but de la méthode (ce qu'elle fait);
–la description des paramètres d'entrées et de sortie.

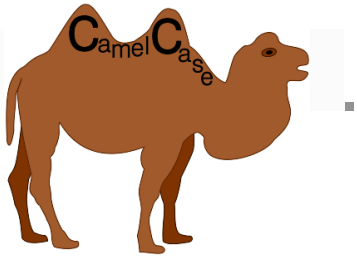
Exemple :

```
1 /*
2 Nom : BeneficeParProduitPeriode
3 Description : Calcule le bénéfice par produit et période
4 Paramètre(s)d'entrée : référence du produit
5                        date de début
6                        date de fin
7 Paramètre(s) de sortie : bénéfice réalisé
8 */
```

3 Identificateurs

Les noms des identificateurs 1 doivent être auto-explicatifs et rédigés en français.

La technique CamelCase a doit être utilisée pour séparer les mots. fr.wikipedia.org/wiki/CamelCase



3.1 Constantes

Le nom des constantes est exprimé entièrement en lettres majuscules en utilisant le caractère souligné pour séparer les mots.

```
1 const int NB_COLONNES_MAX = 75;
```

3.2 Contrôles Visual Studio

Pour faciliter la compréhension, les identificateurs des objets graphiques générés par Visual Studio doivent être modifiés :
–la numérotation est remplacée par un nom explicite;
–le type du contrôle est conservé et doit être abrégé.

Exemple :

```
1 button1_Click;          // C'est le nom généré par Visual Studio
2 buttonStartClick;       // Le nom du contrôle plus sa fonction
3 btnStartClick;          // Abréviation du contrôle plus sa fonction
```

Exception : pour des raisons internes à Visual Studio (utilisation du nom de la fiche pour nommer les fichiers), la fiche (Form1) ne sera pas renommée.

3.2.1 Abréviations standards

Il est possible d'abréger les noms des contrôles selon les conventions des tableaux 1(a), 1(b) et 1(c).

Nom du contrôle	Abréviation
Button	btn
CheckBox	chk
CheckedListBox	clb
ComboBox	cmb
DateTimePicker	dtp
Label	lbl
LinkLabel	llb
ListBox	lsb
ListView	lsv
MaskTextBox	mtb
MonthCalendar	mca
NotifyIcon	nic
NumericUpDown	nud
PictureBox	pib
ProgressBar	prb
RadioButton	rdb
RichTextBox	rtb
TextBox	tbx
Timer	tmr
ToolTip	tot
TreeView	tv
WebBrowser	web

Tableau 1 (a) Contrôles communs

Nom du contrôle	Abréviation
HScrollBar	hsb
VScrollBar	vsb

Tableau 2 (b) Autres contrôles

Nom du contrôle	Abréviation
FlowLayoutPanel	flp
GroupBox	gb
Panel	pnl
SplitContainer	sc
TabControl	tc
TableLayoutPanel	tlp

Tableau 3 (c) Conteneurs

Nom du contrôle	Abréviation
ContextMenuStrip	cms
MenuStrip	ms
StatusStrip	ss
ToolStrip	ts
ToolStripContainer	tsc
ToolStripMenuItem	tsm
ColorDialog	cd
FolderBrowserDialog	fbd
FontDialog	fd
OpenFileDialog	ofd
SaveFileDialog	sfd
Status Bar	stb

Tableau 4 (c) Menus, barres d'outils

3.3 Classes

Les noms de classes débutent par une majuscule.

Ex : Personne

3.4 Objets

Les noms des objets débutent par une minuscule.

Ex : personne

3.5 Champs privés

Les noms des champs privés débutent par le caractère souligné suivi d'une minuscule. Ex :

```
private double _taux ;
```

3.6 Propriétés

Les propriétés (en C#) sont les méthodes d'accès aux champs privés (getter /setter).

Les noms des propriétés débutent par une majuscule. Ex :

```
1  public double Taux  {
2     get { return _taux;}
3     set { _taux = value;}
4  }
```

3.7 Méthodes

Les noms des méthodes débutent par une majuscule.

Utilisez des verbes actifs, éventuellement suivis par des noms.

Ex :

```
- InitialiserLabyrinthe
- EffacerLabyrinthe
- ResoudreLabyrinthe
- CalculerFactureTotale
- MettreAJourAdresse, ConstruireEtatSelection
- AjouterObjet, LibererObjet
- AjouterInteret, ConvertirEuroEnDollar
```

Voir liste de verbes d'action au paragraphe 6.

3.8 Exemple complet

```
1 /*
2 * Projet : 02
3 * Auteur : GAF
4 * Description : Les bases des interfaces , le second projet
5 * Version : 1.0 , 2012.09.14 , GAF , version de base
6 */
7
8 using.namespace M120 ;
9 {
10  public partial class Form1 : Form
11  {
12   private double _taux ;
13   public double Taux {
14      get { return _taux ; }
15      set { _taux = value; }
16  }
17
18  public Form1 () {
19    InitializeComponent();
20  }
21
22  private void btnAfficher_Click ( object sender , EventArgs e ) {
23      // on affiche la valeur qui a été saisi dans le TextBox tbxSaisie
24      string texte = tbxSaisie.Text.Trim();
28      if (texte.Length != 0) {
29          this.Taux = Convert.ToDouble(texte);
30          MessageBox.Show("Texte saisi = " + this.Taux.ToSTring (),
                            "Vérification de la saisie ",
                            MessageBoxButtons.OK, MessageBoxIcon.Information);
31      }
32      else {
33           MessageBox.Show("Saisissez une valeur ...",
                             "Vérification de la saisie",
                             MessageBoxButtons.OK, MessageBoxIcon.Error);
34      }
35  }
36 }
```

3.9 Référence

msdn.microsoft.com/en-us/library/vstudio/ms229002%28v=vs.100%29.aspx

sites.google.com/site/notionscsharpcem/guicontroles/gui-prefixes

4 Valeurs numériques

Le code ne doit contenir aucune valeur numérique à l'exception de 0 et 1. Les autres valeurs sont toutes déclarées sous forme de constantes. Cette méthode facilite la mise à jour et rend les programmes compréhensibles.

Exemple :

```
1 for ( int i =1; i < 15; ++ i )
2   {
3   if ( s > 10)
4      {
5      s = 10;
6      }
7   }
```

4 défauts :

– la valeur 75 n'est pas auto-documentée ;

– si la valeur 10 doit être changée, il faut faire deux modifications ;

– la valeur 10 n'est pas auto-documentée ;

– la variable s n'est pas auto-documentée.

```
1 const int NB_COLONNES_MAX = 75;
2 cont int COMPOSANTE_ROUGE_MAX = 10;
3
4 for ( int i =1; <= NB_COLONNES_MAX ; ++ i )
5     {
6     if ( seuilCouleurRouge > COMPOSANTE_ROUGE_MAX )
7        {
8        seuilCouleurRouge = COMPOSANTE_ROUGE_MAX ;
9        }
```

4 avantages :

– la constante NB_COLONNES_MAX est auto-documentée

– le changement de la limite nécessite une seule modification (COMPOSANTE_ROUGE_MAX) ;

– la constante COMPOSANTE_ROUGE_MAX est auto-documentée ;

– la variable SeuilCouleurRouge est auto-documentée.

5 Raccourcis claviers

Liste exhaustive : http://www.dofactory.com/ShortCutKeys/ShortCutKeys.aspx

5.1 Génération/exécution/débogage

F5	démarre le débogage
SHIFT+F5	arrête le débogage
CTRL+F5	exécute sans débogage
F6	génére la solution
F10/F11	exécute un pas (principal/détaillé)

5.2 Complétion,édition

TAB	insére un modèle de code (if, for, foreach,class...) 2xTAB pour un modèle pré-rempli
CTRL+SPACE	complète le mot (variable, méthode. . .)
CTRL+L	coupe la ligne courante (Line)
SHIFT+ALT+T	déplace la ligne courante vers le bas (Toggle)
SHIFT+ALT+ARROW	sélectionne en mode bloc
CTRL+I	recherche incrémentale (Incremental)
CTRL+SHIFT+R	enregistrement d'une macro (Record)
CTRL+SHIFT+P	exécution de la macro enregistrée (Play)
CTRL+.	- recherche le using de la classe - génère le squelette de la méthode - génère les méthodes d'une interface

5.3 Formatage, insertion

CTRL+K, CTRL+F	auto-indente la sélection
CTRL+K, CTRL+D	auto-indente le fichier
CTRL+K, CTRL+C	commente un bloc (Comment)
CTRL+K, CTRL+U	décommente un bloc (Uncomment)
CTRL+K, CTRL+S	entoure la sélection de : #region (Surround)...
CTRL+K, CTRL+X	insère un snippet

5.4 Navigation/fenêtre

F7	affiche la fenêtre de code
SHIFT+F7	affiche la fenêtre d'interface graphique
CTRL+K, CTRL+K	place/retire un signet (booKmark)
CTRL+K, CTRL+N	va au prochain signet (Next)
CTRL+K, CTRL+P	va au précédent signet (Previous)
CTRL+W, D	affiche la fenêtre de définition de code (classe, structure...) (Definition)
CTRL+M, CTRL+M	ouvre/ferme un pli
CTRL+M, CTRL+O	ferme tous les plis

6 Verbes d'action

Add	Confirm	Edit	Hide
Approve	Connect	Enable	Import
Assert	Convert	Enter	Initialize
Backup	ConvertFrom	Exit	Install
Block	ConvertTo	Expand	Invoke
Checkpoint	Copy	Export	Join
Clear	Debug	Find	Limit
Close	Deny	Format	Lock
Compare	Disable	Get	Measure
Complete	Disconnect	Grant	Merge
Compress	Dismount	Group	Mount

Move	Remove	Send	Trace
New	Rename	Set	Unblock
Open	Repair	Show	Undo
Out	Request	Skip	Uninstall
Ping	Reset	Split	Unlock
Pop	Resolve	Start	Unprotect
Protect	Restart	Step	Unpublish
Publish	Restore	Stop	Unregister
Push	Resume	Submit	Update
Read	Revoke	Suspend	Use
Receive	Save	Switch	Wait
Redo	Search	Sync	Watch
Register	Select	Test	Write

Remarque : ces verbes d'action proviennent de la commande Get-Verb du langage de script de Windows : PowerShell 2

Voir msdn.microsoft.com/en-us/library/ms714428.aspx

7 Mots réservés

AddHandler	CType	If	Optional	SyncLock
AddressOf	Date	Implements	Or	Then
Alias	Decimal	Imports	Overloads	Throw
And	Declare	In	Overridable	To
Ansi	Default	Inherits	Overrides	True
As	Delegate	Integer	ParamArray	Try
Assembly	Dim	Interface	Preserve	TypeOf
Auto	Do	Is	Private	Unicode
Base	Double	Let	Property	Until
Boolean	Each	Lib	Protected	volatile
ByRef	Else	Like	Public	When
Byte	ElseIf	Long	RaiseEvent	While
ByVal	End	Loop	ReadOnly	With
Call	Enum	Me	ReDim	WithEvents
Case	Erase	Mod	Region	WriteOnly
Catch	Error	Module	REM	Xor
CBool	Event	MustInherit	RemoveHandler	eval
CByte	Exit	MustOverride	Resume	extends
CChar	ExternalSource	MyBase	Return	instanceof
CDate	False	MyClass	Select	package
CDec	Finalize	Namespace	Set	var
CDbl	Finally	New	Shadows	
Char	Float	Next	Shared	
CInt	For	Not	Short	
Class	Friend	Nothing	Single	
CLng	Function	NotInheritable	Static	
CObj	Get	NotOverridable	Step	
Const	GetType	Object	Stop	
CShort	Goto	On	String	
CSng	Handles	Option	Structure	
CStr			Sub	

Conventions de codage C# (Guide de programmation C#)

Visual Studio 2010 / 2012

La Spécification du langage C# ne définit aucune norme de codage. Toutefois, les conventions de cette rubrique sont utilisées par Microsoft pour développer des exemples et de la documentation.

Les conventions de codage répondent aux objectifs suivants :

- Elles donnent au code un aspect homogène, afin que les lecteurs puissent se concentrer sur le contenu et non pas sur la disposition.
- Elles permettent aux lecteurs de comprendre le code plus rapidement en faisant des hypothèses selon leur expérience précédente.
- Elles facilitent la copie, la modification et la gestion du code.
- Elles illustrent les meilleures pratiques de C#.

Conventions d'affectation de noms

- Dans les exemples courts qui ne comprennent pas de directive using, utilisez les qualifications d'espace de noms. Si vous savez qu'un espace de noms est importé par défaut dans un projet, vous ne devez pas attribuer des noms qualifiés complets de cet espace de noms. Les noms qualifiés peuvent être interrompus par un point (.) s'ils sont trop longs pour une ligne unique, comme indiqué dans l'exemple suivant.

```
var currentPerformanceCounterCategory = new System.Diagnostics.
     PerformanceCounterCategory();
```

- Vous ne devez pas modifier les noms des objets créés à l'aide de les outils du concepteur de Visual Studio pour les rendre conformes d'autres directives.

Conventions de disposition

Une bonne mise en page utilise la mise en forme pour souligner la structure du code et simplifier la lecture du code. Les exemples et échantillons de Microsoft sont conformes aux conventions suivantes :

- Utilisez les paramètres par défaut de l'éditeur de code (mise en retrait intelligente, mise en retrait de quatre caractères, enregistrement des tabulations en tant qu'espaces). Pour plus d'informations, consultez Options, Éditeur de texte, C#, Mise en forme.
- Écrivez une seule instruction par ligne.
- Écrivez une seule déclaration par ligne.
- Si les lignes continuation ne sont pas mises en retrait automatiquement, mettez-les en retrait d'un taquet de tabulation (quatre espaces).
- Laissez au moins une ligne vide entre les définitions de méthode et de propriété.
- Utilisez des parenthèses pour rendre des clauses apparentes dans une expression, comme illustré dans le code suivant.

```
if ((val1 > val2) && (val1 > val3)) {
     // Take appropriate action.
}
```

Conventions de commentaires

- Placez le commentaire sur une ligne séparée, pas à la fin d'une ligne de code.
- Commencez le texte du commentaire par une lettre majuscule.
- Ajoutez un point à la fin du texte de commentaire.
- Insérez un espace entre le délimiteur de commentaire (//) et le texte de commentaire, comme indiqué dans l'exemple suivant.

```
// The following declaration creates a query. It does not run // the query.
```

- Ne créez pas de blocs d'astérisques mis en forme autour des commentaires.

Indications concernant le langage

Les sections suivantes décrivent les pratiques que l'équipe C# suit pour préparer des exemples de code et des échantillons.

Type de données String

- Utilisez l'opérateur + pour concaténer des chaînes courtes, comme indiqué dans le code suivant.

```
string displayName = nameList[n].LastName + ", " + nameList[n].FirstName;
```

- Pour ajouter des chaînes dans des boucles, en particulier si vous travaillez avec de grandes quantités de texte, utilisez un objet StringBuilder.

```
var phrase = "lalalalalalalalalalalalalalalalalalalalalalalalalalalalalala";
var manyPhrases = new StringBuilder();
for (var i = 0; i < 10000; i++) {
     manyPhrases.Append(phrase);
} //Console.WriteLine("tra" + manyPhrases);
```

Variables locales implicitement typées

- Utilisez les types implicites pour les variables locales lorsque le type de la variable apparaît à droite de l'instruction, ou lorsque le type précis n'est pas important.

```
// When the type of a variable is clear from the context, use var
// in the declaration.
var var1 = "This is clearly a string.";
var var2 = 27;
var var3 = Convert.ToInt32(Console.ReadLine());
```

- N'utilisez pas var lorsque le type n'apparaît pas à droite de l'instruction.

```
// When the type of a variable is not clear from the context, use an
// explicit type.
int var4 = ExampleClass.ResultSoFar();
```

- N'utilisez pas le nom de la variable pour indiquer le type de la variable. Il n'est peut-être pas correct.

```
// Naming the following variable inputInt is misleading.
// It is a string.
var inputInt = Console.ReadLine();
Console.WriteLine(inputInt);
```

- Évitez d'utiliser *var* à la place de dynamique.
- Utilisez les types implicites pour déterminer le type des boucles de variable de boucle dans les boucles for et foreach.

L'exemple suivant utilise les types implicites dans une instruction for.

```
var syllable = "ha";
var laugh = "";
for (var i = 0; i < 10; i++) {
     laugh += syllable;
     Console.WriteLine(laugh);
}
```

L'exemple suivant utilise les types implicites dans une instruction foreach.

```
foreach (var ch in laugh) {
     if (ch == 'h')
         Console.Write("H");
     else
         Console.Write(ch);
}
Console.WriteLine();
```

Type de données non signé

- En général, utilisez int plutôt que des types non signés. L'utilisation de int est commune dans tout le langage C#, et il est plus facile d'interagir avec d'autres bibliothèques lorsque vous utilisez int.

Tableaux

- Utilisez la syntaxe concise lorsque vous initialisez des tableaux sur la ligne de déclaration.

```
// Preferred syntax. Note that you cannot use var here instead of string[].
string[] vowels1 = { "a", "e", "i", "o", "u" };
// If you use explicit instantiation, you can use var.
var vowels2 = new string[] { "a", "e", "i", "o", "u" };
// If you specify an array size, you must initialize the elements one at a time.
var vowels3 = new string[5];
vowels3[0] = "a";
vowels3[1] = "e"; // And so on.
```

Délégués

- Utilisez la syntaxe concise pour créer des instances d'un type délégué.

```
// First, in class Program, define the delegate type and a method that
// has a matching signature.
// Define the type.
public delegate void Del(string message);
// Define a method that has a matching signature.
public static void DelMethod(string str) {
     Console.WriteLine("DelMethod argument: {0}", str);
}
```

```
// In the Main method, create an instance of Del.
// Preferred: Create an instance of Del by using condensed syntax.
Del exampleDel2 = DelMethod;
// The following declaration uses the full syntax.
Del exampleDel1 = new Del(DelMethod);
```

Instructions try-catch et using dans la gestion des exceptions

- Utilisez une instruction try-catch pour la gestion de la plupart des exceptions.

```
static string GetValueFromArray(string[] array, int index) {
     try
     {
         return array[index];
     }
     catch (System.IndexOutOfRangeException ex)
     {
         Console.WriteLine("Index is out of range: {0}", index);
         throw;
     }
}
```

- Simplifiez votre code à l'aide de l'instruction using du langage C#. Si vous avez une instruction try-finally dans laquelle le seul code dans le bloc finally est un appel à la méthode Dispose, utilisez une instruction using à la place.

```
// This try-finally statement only calls Dispose in the finally block.
Font font1 = new Font("Arial", 10.0f);
try {
     byte charset = font1.GdiCharSet;
}
finally {
     if (font1 != null)
     {
         ((IDisposable)font1).Dispose();
     }
}
// You can do the same thing with a using statement.
using (Font font2 = new Font("Arial", 10.0f)) {
     byte charset = font2.GdiCharSet;
}
```

Opérateurs && et ||

- Pour éviter les exceptions et améliorer les performances en ignorant les comparaisons inutiles, utilisez && au lieu de & et || au lieu de | lorsque vous effectuez des comparaisons, comme indiqué dans l'exemple suivant.

```
Console.Write("Enter a dividend: ");
var dividend = Convert.ToInt32(Console.ReadLine());
Console.Write("Enter a divisor: ");
var divisor = Convert.ToInt32(Console.ReadLine());
// If the divisor is 0, the second clause in the following condition
// causes a run-time error. The && operator short circuits when the
// first expression is false. That is, it does not evaluate the
// second expression. The & operator evaluates both, and causes
// a run-time error when divisor is 0.
if ((divisor != 0) && (dividend / divisor > 0)) {
     Console.WriteLine("Quotient: {0}", dividend / divisor);
}
else {
     Console.WriteLine("Attempted division by 0 ends up here.");
}
```

Opérateur New

- Utilisez le formulaire concis d'instanciation d'objets, avec des types implicites, comme indiqué dans la déclaration suivante.

```
var instance1 = new ExampleClass();
```

La ligne précédente est équivalente à la déclaration suivante.

```
ExampleClass instance2 = new ExampleClass();
```

- Utilisez des initialiseurs d'objets pour simplifier la création d'objets.

```
// Object initializer.
var instance3 = new ExampleClass {
     Name = "Desktop", ID = 37414,
     Location = "Redmond", Age = 2.3
};
// Default constructor and assignment statements.
var instance4 = new ExampleClass();
instance4.Name = "Desktop";
instance4.ID = 37414;
instance4.Location = "Redmond";
instance4.Age = 2.3;
```

Gestion des événements

- Si vous définissez un gestionnaire d'événements que vous n'avez pas besoin de supprimer ultérieurement, utilisez une expression lambda.

```
public Form2() {
     // You can use a lambda expression to define an event handler.
     this.Click += (s, e) =>
         {
             MessageBox.Show(
                 ((MouseEventArgs)e).Location.ToString());
         };
}
```

```
// Using a lambda expression shortens the following traditional definition.
public Form1() {
     this.Click += new EventHandler(Form1_Click);
}
void Form1_Click(object sender, EventArgs e) {
     MessageBox.Show(((MouseEventArgs)e).Location.ToString());
}
```

Membres statiques

- Appelez des membres statiques à l'aide du nom de la classe : ClassName.StaticMember. N'accédez pas à un membre statique qui est défini dans une classe de base d'une classe dérivée.

Requêtes LINQ

- Utilisez des noms explicites pour les variables de requête. L'exemple suivant utilise seattleCustomers pour des clients qui se trouvent à Seattle.

```
var seattleCustomers = from cust in customers
                       where cust.City == "Seattle"
                       select cust.Name;
```

- Utilisez des alias pour vous assurer que les noms de propriété des types anonymes sont correctement capitalisés à l'aide de la casse Pascal.

```
var localDistributors =
     from customer in customers
     join distributor in distributors on customer.City equals distributor.City
     select new { Customer = customer, Distributor = distributor };
```

- Renommez les propriétés lorsque les noms de propriété dans le résultat sont ambigus. Par exemple, si votre requête retourne un nom de client et un ID du serveur de distribution, au lieu de les laisser en tant que Name et ID dans le résultat, renommez-les pour clarifier que Name correspond au nom d'un client, et que ID désigne l'ID du serveur de distribution.

```
var localDistributors2 =
    from cust in customers
     join dist in distributors on cust.City equals dist.City
     select new { CustomerName = cust.Name, DistributorID = dist.ID };
```

- Utilisez les types implicites dans la déclaration des variables de requête et de portée.

```
var seattleCustomers = from cust in customers
                       where cust.City == "Seattle"
                       select cust.Name;
```

- Alignez les clauses de requête sous la clause de , comme indiqué dans les exemples précédents.
- Utilisez les clauses where avant les autres clauses de requête afin de vous assurer que les clauses de requête ultérieures s'appliquent correctement au groupe de données réduit et filtré.

```
var seattleCustomers2 = from cust in customers
                        where cust.City == "Seattle"
                        orderby cust.Name
                        select cust;
```

- Utilisez plusieurs clauses from au lieu d'une clause join pour accéder aux collections internes. Par exemple, chaque élément d'une collection d'objets Student peut contenir une collection de notes d'examens. Lorsque la requête suivante est exécutée, elle retourne chaque note qui est supérieure à 90, avec le nom de l'étudiant qui l'a reçue.

```
// Use a compound from to access the inner sequence within each element.
var scoreQuery = from student in students
                 from score in student.Scores
                 where score > 90
                 select new { Last = student.LastName, score };
```